A Python programozási nyelv

Takács Gábor

1. Bevezetés

Programozási alapfogalmak

- **Algoritmus**: Valamely feladat megoldására alkalmas véges hosszú lépéssorozat. A fogalom hétköznapi feladatokra is alkalmazható (pl. Sacher-torta készítés, könyvespolc takarítás :-).
- **Adatszerkezet**: Adatelemek tárolására és hatékony használatára szolgáló séma (példa: lista).
- **Programozási nyelv**: Szigorú szabályokra épülő nyelv, melynek segítségével az ember képes a számítógép felé kommunikálni az utasításait.
- **Programozás**: Algoritmusok és adatszerkezetek megtervezése illetve megvalósításuk valamilyen programozási nyelven.

A Python nyelv jellemzői

+ szintaxisa tömör, elegáns
+ könnyen tanulható ("brain-friendly")
+ több 10 ezer külső csomag érhető el hozzá (https://pypi.org/)
+ erős közösség, évente PyCon konferenciák
+ szabadon használható
+ platformfüggetlen
+ értelmezett nyelv, típusai dinamikusak
+ többparadigmás nyelv
− bizonyos feladatokhoz lassú lehet
− többszálú lehetőségei korlátozottak

2. A Jupyter Notebook környezet

A Jupyter Notebook egy böngésző alapú interaktív munkakörnyezet. Elsődlegesen a Python nyelvhez fejlesztették ki, de más programozási nyelvekkel is használható. Egy notebook cellákból áll, a cellák lehetnek szöveges (Markdown) vagy kód típusúak. A kódcellákat le lehet futtatni, akár többször is egymás után. A futtatás eredménye megjelenik az adott kódcella utáni kimenetben. A notebook használata kétféle üzemmódban történik:

- Parancsmódban tudjuk elvégezni a cellaszintű műveleteket (pl. új cella beszúrása, cella törlése, cellák mozgatása, lépegetés a cellák között stb). Néhány billentyűparancs:
  - `b`: Új kódcella beszúrása az aktuális cella után.
  - `m`: Az aktuális cella típusának átállítása szövegesre.
  - `dd`: Az aktuális cella törlése.
  - `Enter`: Átlépés szerkesztőmódba (az aktuális cella tartalmának szerkesztése).
- Szerkesztőmódban tudjuk szerkeszteni a cellák tartalmát. Néhány billentyűparancs:
  - `Ctrl+Enter`: Az aktuális cella futtatása.
  - `Esc`: Visszalépés parancsmódba.
- A billentyűparancsokról a Help / Keyboard Shortcuts menü ad részletesebb leírást.

3. Egyszerű adattípusok

3.1. Egész szám

```
In [1]: # A számok között a szokásos módon végezhetünk műveleteket.
        1 + 1

        # Megjegyzések:
        # - A szóközök nem számítanak, a fenti írásmód a PEP 8 kódolási stílust követi.
        # - A Jupyter a futtatás után megjeleníti a cella utolsó kifejezését.

Out[1]: 2

In [2]: # Ha szeretnénk felülbírálni a precedenciát, használjunk zárójelezést!
        (2 + 3) * 4

Out[2]: 20

In [3]: # A Python képes tetszőleges hosszúságú egész számokkal dolgozni,
        # nincsen túlcsordulási hiba.
        111111111111111111111111111111111111111111111111111111111111111111111111111111 + 1

Out[3]: 1111111111111111111111111111111111111111111111111111111111111111111111111112

In [4]: # Hozzunk létre egy i nevű változót, és tegyük bele a 11 értéket!
        i = 11

        # Megjegyzések:
```

```
        # - Az = az értékadás műveleti jele.
        # - i felveszi a megadott értéket, de magának az értékadásnak nincs eredménye.
        # - Emiatt a cella kimenete üres.
```

manók

```
In [5]: # A változóra a továbbiakban is lehet hivatkozni.
        2 * i
```

```
Out[5]: 22
```

```
In [6]: # A változó értéke természetesen változtatható.
        i = 42
        i
```

```
Out[6]: 42
```

```
In [7]: # Az értékadást lehet kombinálni a többi művelettel.
        i += 1 # ekvivalens az i = i + 1 értékadással
        i
```

dry bean

```
Out[7]: 43
```

```
In [8]: # Lebegőpontos osztás.
        7 / 3
```

```
Out[8]: 2.3333333333333335
```

```
In [9]: # Egészosztás (levágja a törtrészt).
        # Sok hibalehetőséget megelőz, hogy külön műveleti jele van.
        7 // 3
```

dice

```
Out[9]: 2
```

```
In [10]: # Maradékképzés.
         7 % 3
```

```
Out[10]: 1
```

```
In [11]: # Van hatványozás is, ** a műveleti jele.
         2**10
```

```
Out[11]: 1024
```

3.2. Lebegőpontos szám

A lebegőpontos számábrázolás lehetővé teszi a valós számokkal történő, közelítő számolást. A Python lebegőpontos típusa az IEEE-754 szabvány dupla pontosságú (64 bites double) típusát valósítja meg.

```
In [12]: # Lebegőpontos állandókat a tizedespont használatával tudunk megadni.
         1.23 * 4.56
```

```
Out[12]: 5.6088

In [13]: # Gyök kettő (közelítő) kiszámítása.
         2**0.5

Out[13]: 1.4142135623730951

In [14]: # Hozzunk létre egy f nevű, lebegőpontos típusú változót!
         f = 1.5
         f

Out[14]: 1.5

In [15]: # A type függvénnyel tudjuk lekérdezni f típusát.
         type(f)

Out[15]: float

In [16]: # ...vagy bármely más érték típusát.
         type(2 * 3)

Out[16]: int

In [17]: # Tegyünk most f-be egy int típusú értéket!
         # Pythonban ez minden probléma nélkül megtehető.
         f = 100
         type(f)

Out[17]: int
```

3.3. Komplex szám

A Python támogatja a komplex számokkal való számolást, külső könyvtárak használata nélkül.

```
In [18]: # Osztás algebrai alakban.
         (2 + 3j) / (5 - 3.5j)

Out[18]: (-0.013422818791946262+0.5906040268456377j)

In [19]: # A képzetes egység hatványozása.
         1j**2019

Out[19]: (8.77583695147045e-14-1j)
```

3.4. Sztring

A sztring adattípus szöveges értékek tárolására szolgál. Pythonban a sztring nem más mint Unicode szimbólumok (másnéven Unicode karakterek) nem módosítható sorozata.

```
In [20]: # A sztringállandót ' jelekkel határoljuk.
         'alma'
```

```
Out[20]: 'alma'

In [21]: # ...de lehet használni " jeleket is.
         "körte"

Out[21]: 'körte'

In [22]: # Megjegyzés: Az előző cellák kimenetében a ' nem a sztring része, csak az
         # adattípust jelzi. Írjuk ki a sztring tartalmát, határoló jelek nélkül!
         print('alma')
```

```
alma
```



```
In [23]: # A type függvény most is működik.
         type('alma')

Out[23]: str

In [24]: # A sztringben természetesen használhatunk Unicode szimbólumokat.
         'π ∞ ℕ'

Out[24]: 'π ∞ ℕ'

In [25]: # A kétféle határoló értelme:
         print("asd'fgh")
         print('asd"fgh')
```



```
asd'fgh
asd"fgh
```

```
In [26]: # ...egyébként le kéne védeni az ' ill. ' karaktert.
         print('asd\'fgh')
         print("asd\"fgh")
```

```
asd'fgh
asd"fgh
```



```
In [27]: # Hozzunk létre egy s nevű sztringváltozót!
         s = 'sör'

In [28]: # s karaktereinek kinyerése.
         # Megjegyzés: Az indexelés 0-tól indul.
         s[0]

Out[28]: 's'
```

```
In [29]: # A kinyert karaktert egy 1 hosszú sztring formájában kapjuk vissza.
         type(s[0])

Out[29]: str

In [30]: # Túlindexelés esetén hibaüzenetet kapunk.
         s[3]
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-11-da58443a8b01> in <module>()
  1 # Túlindexelés esetén hibaüzenetet kapunk.
----> 2 s[3]


IndexError: string index out of range
```

```
In [31]: # A sztring karaktereit nem lehet módosítani!
         # (Majd később meglátjuk, hogy miért.)
         s[0] = 'x'
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-12-a489aa6d5eb9> in <module>()
  1 # A sztring karaktereit nem lehet módosítani!
  2 # (Majd később meglátjuk, hogy miért.)
----> 3 s[0] = 'x'


TypeError: 'str' object does not support item assignment
```

```
In [32]: # Természetesen s-nek adhatunk új értéket.
         s = 'bor'

         # Megjegyzés: Az értékadás megtörténik, de magának az értékadó kifejezésnek
         # nincs eredménye. Emiatt a cellának nincsen kimenete.

In [33]: # Írjuk ki s tartalmát!
         print(s)
```

```
bor
```

nevek

```
In [34]: # A sztring hossza (Unicode szimbólumok száma):
         len('Béla∞')

Out[34]: 5

In [35]: # Sztringek összefűzése.
         'sör' + 'bor'

Out[35]: 'sörbor'

In [36]: # Tartalmazásvizsgálat.
         'ab' in 'abrakadabra'

Out[36]: True

In [37]: # Sztringből a kódolás műveletével képezhetünk bájtsorozatot.
         b = 'Géza'.encode('utf-8')
         b

Out[37]: b'G\xc3\xa9za'

In [38]: # Az eredmény típusa.
         type(b)

Out[38]: bytes

In [39]: # A bájtok száma nagyobb lehet, mint a Unicode szimbólumok száma!
         len(b)

Out[39]: 5

In [40]: # Feladat:
         # Hány bájton tárolódnak a magyar ábécé ékezetes kisbetűi UTF-8 kódolás esetén?

         print(len('á'.encode('utf-8')))
         print(len('é'.encode('utf-8')))
         print(len('í'.encode('utf-8')))
         print(len('ó'.encode('utf-8')))
         print(len('ö'.encode('utf-8')))
         print(len('ő'.encode('utf-8')))
         print(len('ú'.encode('utf-8')))
         print(len('ü'.encode('utf-8')))
         print(len('ű'.encode('utf-8')))

         # Megjegyzés: A fenti kód tele van ismétléssel.
         # Hamarosan megtanuljuk, hogy hogyan lehet elegánsabbá tenni.
```

ipcim

```
2
2
2
2
2
2
2
2
2
```

```
In [41]: # Hány bájton tárolódik a π és a ∞ szimbólum?
         print(len('π'.encode('utf-8')))
         print(len('∞'.encode('utf-8')))
```

```
2
3
```

```
In [42]: # Bájtsorozatból a dekódolás műveletével képezhetünk sztringet.
         b.decode('utf-8')

Out[42]: 'Géza'

In [43]: # Üres sztring létrehozása.
         ''

Out[43]: ''

In [44]: # Fehér karakterek (szóköz, tabulátor, sortörés) eltávolítása
         # a sztring elejéről és végéről.
         '  \talma\n'.strip()

Out[44]: 'alma'

In [45]: # Megadott karakterek eltávolítása a sztring elejéről és végéről.
         '---alma+++'.strip('+-')

Out[45]: 'alma'
```

3.5. Logikai érték

A logikai igaz értéket a *True*, a hamisat a *False* jelöli. A nagy kezdőbetű fontos, a Python különbözőnek tekinti a kis- és nagybetűket.

```
In [46]: # Hozzunk létre logikai típusú változót!
         x = True
         x
```

Out[46]: True

```
In [47]: # Logikai ÉS művelet.
         print(True and False)
         print(True and True)
```

```
False
True
```

```
In [48]: # Logikai VAGY művelet.
         print(False or False)
         print(False or True)
```

```
False
True
```

```
In [49]: # Logikai tagadás.
         not x
```

Out[49]: False

```
In [50]: # Az összehasonlító műveletek eredménye logikai érték.
         print(2 <= 3)
         print(5 > 10)
```

```
True
False
```

```
In [51]: # Pythonban az egyenlőségvizsgálat műveleti jele ==.
         print('alma' == 'alma')
         print('alma' == 'körte')
```

```
True
False
```

3.6. None

A szó jelentése *semmi* vagy *egyik sem*. A Pythonban a None értéknek helykitöltő szerepe van. Ezzel jelölhetjük pl. a hiányzó vagy érvénytelen eredményt vagy az alapértelmezett beállítást.

```
In [52]: # A None érték típusa.
         type(None)
```

Out[52]: NoneType

```
In [53]: # Ha a cella utolsó kifejezése None értékű, akkor nincs kimenet.
         1 + 1
         None
```

4. Kollekciók

4.1. 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.

```
In [54]: # Hozzunk létre egy t nevű, 3 elemű tuple változót!
         t = (10, 20, 30)

In [55]: # Ellenőrizzük t típusát!
         type(t)

Out[55]: tuple

In [56]: # Az elemek számát a len függvénnyel kérdezhetjük le.
         len(t)

Out[56]: 3

In [57]: # Tuple elemeinek elérése (az indexelés 0-tól indul).
         t[1]

Out[57]: 20

In [58]: # Az elemeken nem lehet módosítani!
         t[1] = 200
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-37-d2bcc5d6cf25> in <module>()
  1 # Az elemeken nem lehet módosítani!
----> 2 t[1] = 200


TypeError: 'tuple' object does not support item assignment
```

```
In [59]: # Az elemeknek nem kell azonos típusúnak lenniük.
         t = (1, 2.5, 'alma')

In [60]: # Tartalmazásvizsgálat.
         2.5 in t

Out[60]: True
```

```
In [61]: # Amennyiben nem okoz kétértelműséget, a ( és ) határoló elhagyható!
         a = 2, 3, 4, 'alma'
         type(a)

Out[61]: tuple

In [62]: # Üres tuple létrehozása.
         ()

Out[62]: ()

In [63]: # Egy elemű tuple létrehozása.
         (42,)

Out[63]: (42,)
```

4.2. Lista

A lista a tuple módosítható változata. Új elemet is hozzá lehet adni, illetve meglévő elemeken is lehet módosítani. Az indexelés $O(1)$, a tartalmazásvizsgálat $O(n)$ időben fut le itt is.

```
In [64]: # Hozzunk létre egy l nevű, 4 elemű listaváltozót!
         # Az elemeknek nem kell azonos típusúnak lenniük.
         l = [2, 3, 4, 'sör']
         l

Out[64]: [2, 3, 4, 'sör']

In [65]: # Ellenőrizzük l típusát, és kérdezzük le az elemek számát!
         type(l), len(l)

Out[65]: (list, 4)

In [66]: # Lista elemeinek elérése (az indexelés 0-tól indul).
         l[1]

Out[66]: 3

In [67]: # Listaelem módosítása.
         l[1] = 30
         l

Out[67]: [2, 30, 4, 'sör']

In [68]: # Listába elemként beágyazhatunk másik listát.
         [[1, 2], [3, 4], [5, 6]]

Out[68]: [[1, 2], [3, 4], [5, 6]]

In [69]: # Elem beszúrása a lista végére.
         l.append('bor')
         l
```

```
Out[69]: [2, 30, 4, 'sör', 'bor']

In [70]: # Elem beszúrása a lista közepére.
         l.insert(3, 42)
         l

Out[70]: [2, 30, 4, 42, 'sör', 'bor']

In [71]: # Tartalmazásvizsgálat.
         30 in l

Out[71]: True

In [72]: # Első előfordulás indexének meghatározása.
         l.index('sör')

Out[72]: 4

In [73]: # Egy szekvencia összes elemének hozzáfűzése a listához.
         l.extend([22, 23, 24])
         l

Out[73]: [2, 30, 4, 42, 'sör', 'bor', 22, 23, 24]

In [74]: # Az extend különbözik az append-től!
         l.append([22, 23, 24])
         l

Out[74]: [2, 30, 4, 42, 'sör', 'bor', 22, 23, 24, [22, 23, 24]]

In [75]: # Adott indexű elem törlése.
         l.pop(2)

Out[75]: 4

In [76]: # Két lista összefűzése egy új listába.
         [10, 20] + [30, 40, 50]

Out[76]: [10, 20, 30, 40, 50]

In [77]: # Utolsó elem törlése.
         l.pop()

Out[77]: 24

In [78]: # Nézzük meg, hogy mi maradt az l listában!
         l

Out[78]: [2, 30, 42, 'sör', 'bor', 22, 23]

In [79]: # Üres lista létrehozása.
         []

Out[79]: []
```

4.3. Halmaz

A halmaz adattípus a matematikai halmazfogalom számítógépes megfelelője. Halmazt indexelni nem lehet, a tartalmazásvizsgálat $O(1)$ időben fut le.

```
In [80]: # Hozzunk létre egy s nevű halmazváltozót!
         s = {2, 3, 4}

In [81]: # Ellenőrizzük s típusát és elemszámát!
         type(s), len(s)

Out[81]: (set, 3)

In [82]: # Tartalmazásvizsgálat.
         4 in s

Out[82]: True

In [83]: # Elem hozzáadása a halmazhoz.
         s.add(42)
         s

Out[83]: {2, 3, 4, 42}

In [84]: # Halmazműveletek.

         {1, 2, 3} | {3, 4, 5} # unió

Out[84]: {1, 2, 3, 4, 5}

In [85]: {1, 2, 3} & {3, 4, 5} # metszet

Out[85]: {3}

In [86]: {1, 2, 3} - {3, 4, 5} # kivonás

Out[86]: {1, 2}

In [87]: # Az elemek típusa nem feltétlenül azonos.
         {1, 2, 'Móricka'}

Out[87]: {'Móricka', 2, 1}

In [88]: # A halmazba bármilyen nem módosítható típusú elemet be lehet tenni.
         {1, 2, (3, 4)}

Out[88]: {1, 2, (3, 4)}

In [89]: # ...módosíthatót viszont nem lehet!
         {1, 2, [3, 4]}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-db01a652620f> in <module>()
----> 1 {1, 2, [3, 4]}

TypeError: unhashable type: 'list'
```

```
In [90]: # Elem eltávolítása.
         s.remove(4)
         s

Out[90]: {2, 3, 42}

In [91]: # Üres halmaz létrehozása.
         set()

Out[91]: set()
```

4.4. Szótár

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

```
In [92]: # Hozzunk létre egy d nevű szótárváltozót!
         d = {'a': 10, 'b': 20, 100.0: 30}

In [93]: # Ellenőrizzük le d típusát és elemszámát!
         type(d), len(d)

Out[93]: (dict, 3)

In [94]: # Létező kulcshoz tartozó érték lekérdezése.
         d['a']

Out[94]: 10

In [95]: # Nem létező kulcshoz tartozó érték lekérdezése.
         d['aa']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
```

```
    <ipython-input-16-e373c19b8509> in <module>()
----> 1 d['aa']


    KeyError: 'aa'
```

```
In [96]: # Kulcshoz tartozó érték módosítása.
         d['a'] = 42
         d

Out[96]: {'a': 42, 'b': 20, 100.0: 30}

In [97]: # Új kulcs-érték pár beszúrása.
         d['Józsi'] = 101
         d

Out[97]: {'a': 42, 'b': 20, 100.0: 30, 'Józsi': 101}

In [98]: # Kulcs-érték pár törlése.
         del d['Józsi']
         d

Out[98]: {'a': 42, 'b': 20, 100.0: 30}

In [99]: # Benne van-e egy kulcs a szótárban?
         'a' in d

Out[99]: True

In [100]: # Üres szótár létrehozása.
          {}

Out[100]: {}
```

5. Konverzió

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

```
In [101]: int(2.7) # float => int

Out[101]: 2

In [102]: float('10') # str => float

Out[102]: 10.0
```

```
In [103]: str(20) # int => str

Out[103]: '20'

In [104]: tuple([1, 2, 3]) # list => tuple

Out[104]: (1, 2, 3)

In [105]: list((4, 5, 6)) # tuple => list

Out[105]: [4, 5, 6]

In [106]: set((7, 8, 9)) # tuple => set

Out[106]: {7, 8, 9}

In [107]: dict([('a', 1), ('b', 2)]) # párok listája => dict

Out[107]: {'a': 1, 'b': 2}

In [108]: list({'a': 1, 'b': 2}.items()) # dict => párok listája

Out[108]: [('a', 1), ('b', 2)]

In [109]: # Üres lista létrehozása (alternatív megoldás).
          list()

Out[109]: []
```

6. 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 érkezhet egy fájlból vagy egy másik programból, a standard kimenet és hibakimenet pedig íródhat fájlba vagy továbbítódhat másik programnak.

6.1. Standard bemenet

A standard bemenetről adatokat bekérni az input függvény segítségével lehet. Az eredmény sztring típusú. Ha más adattípusra van szükség, akkor konvertálni kell.

```
In [110]: # Sztring típusú adat beolvasása.
          x = input('Kérek egy szöveget: ')
          x
```

```
Kérek egy szöveget: Géza, kék az ég.
```

```
Out[110]: 'Géza, kék az ég.'
```

```
In [111]: # Egész típusú adat beolvasása.
          y = int(input('Kérek egy egész számot: '))
          y
```

```
Kérek egy egész számot: 42
```

```
Out[111]: 42
```

6.2. Standard kimenet és hibakimenet

A standard kimenetre és hibakimenetre kiírni a print függvény segítségével lehet.

```
In [112]: # Kiírás a standard kimenetre.
          print('hello')
          print('bello')
```

```
hello
bello
```

```
In [113]: # Kiírás soremelés nélkül.
          print('hello', end='')
          print('bello', end='')
```

```
hellobello
```

```
In [114]: # Egyetlen soremelés kiírása.
          print()
```

```

```

```
In [115]: # Kiírás a standard hibakimenetre.
          import sys
          print('Hiba történt.', file=sys.stderr)
```

```
Hiba történt.
```

6.3. Formázott kiírás

```
In [116]: # Formázott kiírás format metódussal.
          x1 = 10.23
          x2 = 3.56
          print('Az első megoldás {}, a második megoldás {}.'.format(x1, x2))
```

```
Az első megoldás 10.23, a második megoldás 3.56.
```

```
In [117]: # Kiírás 1 tizedesjegy pontossággal.
          print('Az első megoldás {:.1f}, a második megoldás {:.1f}.'.format(
              x1, x2
          ))
```

```
Az első megoldás 10.2, a második megoldás 3.6.
```

```
In [118]: # Egész szám ill. sztring kiírása.
          print('egész szám: {}, sztring: {}'.format(42, 'Móricka'))
```

```
egész szám: 42, sztring: Móricka
```

```
In [119]: # Formázott kiírás % operátorral.
          print('Az első megoldás %f, a második megoldás %f.' % (x1, x2))
```

```
Az első megoldás 10.230000, a második megoldás 3.560000.
```

```
In [120]: # Kiírás 1 tizedesjegy pontossággal.
          print('Az első megoldás %.1f, a második megoldás %.1f.' % (x1, x2))
```

```
Az első megoldás 10.2, a második megoldás 3.6.
```

```
In [121]: # Egész szám ill. sztring kiírása.
          print('egész szám: %d, sztring: %s' % (42, 'Móricka'))
```

```
egész szám: 42, sztring: Móricka
```

```
In [122]: # Megjegyzés: A format metódus ill. a % operátor
          # kiírás nélkül is alkalmazható, sztringműveletként.
          x = 'Józsi'
          'Helló, {}!'.format(x)

Out[122]: 'Helló, Józsi!'
```

7. Vezérlési szerkezetek

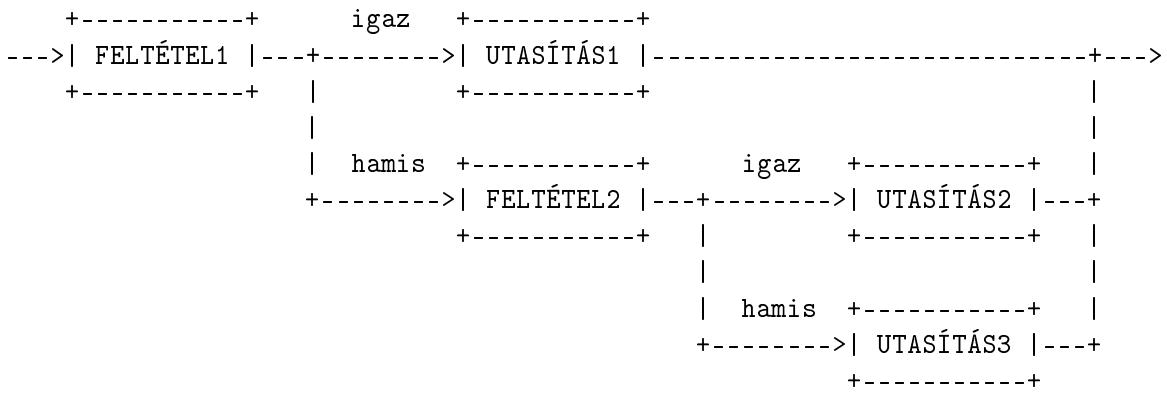
Pythonban a vezérlési szerkezetek belsejét indentálással (azaz beljebb írással) kell jelölni. Emiatt garantált, hogy a program megjelenése és logikai jelentése összhangban van.

7.1. if utasítás

Szintaxis:

```
if FELTÉTEL1:
    UTASÍTÁS1
elif FELTÉTEL2:
    UTASÍTÁS2
else:
    UTASÍTÁS3
```

Működés:



Megjegyzések:

- Több elif ág is szerepelhet.
- Az elif ágak és az else ág is elhagyható.
- Ha az utasítás 1 soros, akkor írható az if-fel elif-fel ill. az else-zel azonos sorba.

```
In [123]: # Példa: Kérsz sört?
          x = int(input('Hány éves vagy? '))
          if x >= 18:
              print('Kérsz sört?')
          else:
              print('Nem adhatok sört.')
```

```
Hány éves vagy? 17
Nem adhatok sört.
```

```
In [124]: # Példa: Másodfokú egyenlet megoldó.
          a = float(input('a: '))
```

```
b = float(input('b: '))
c = float(input('c: '))

d = b**2 - 4 * a * c
if d > 0:
    x1 = (-b + d**0.5) / (2 * a)
    x2 = (-b - d**0.5) / (2 * a)
    print(x1, x2)
elif d == 0:
    x1 = -b / (2 * a)
    print(x1)
else:
    print('Nincs megoldása!')
```

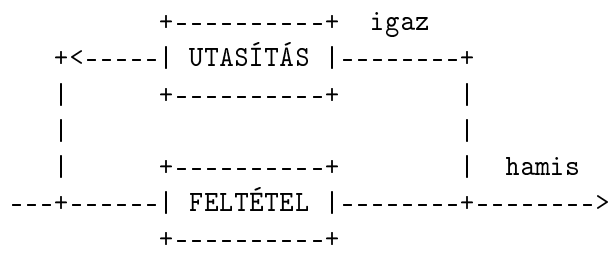
```
a: 1
b: 3
c: 2
-1.0 -2.0
```

7.2. while utasítás

Szintaxis:

```
while FELTÉTEL:
    UTASÍTÁS
```

Működés:



Megjegyzések:

- Egy jól megírt program esetén az utasítás a feltételt előbb-utóbb hamisra állítja. (Ellenkező esetben végtelen ciklus keletkezik.)
- Akkor érdemes while ciklust alkalmazni, ha a ciklus elején még nem tudjuk pontosan az iterációk számát.

```
In [125]: # Példa: Móricka a programozásvizsgán.

          while int(input('Hány pontot értél el? ')) < 16:
              print('Tanulj még!')
          print('Gratulálok, átmentél!')
```

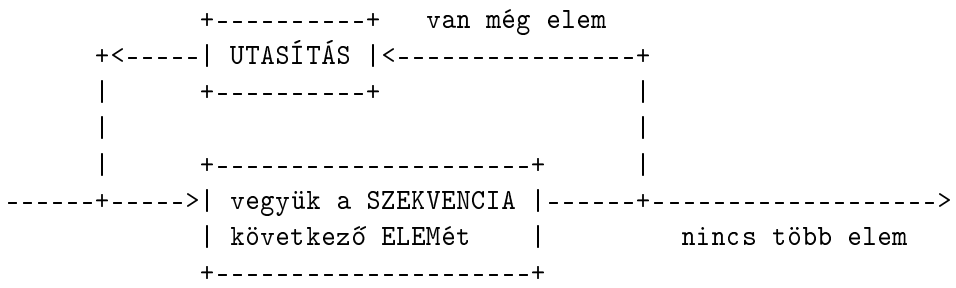
```
Hány pontot értél el? 12
Tanulj még!
Hány pontot értél el? 14
Tanulj még!
Hány pontot értél el? 20
Gratulálok, átmentél!
```

7.3. for utasítás

Szintaxis:

```
for ELEM in SZEKVENCIA:
    UTASÍTÁS
```

Működés:



Megjegyzések:

- A szekvencia lehet egész számok folytonos sorozata, de lehet más is (pl. sztring, tuple, lista, halmaz, szótár, megnyitott fájl).
- Akkor érdemes for ciklust alkalmazni, ha A) a szekvencia már rendelkezésre áll vagy B) a ciklus kezdetekor tudjuk az iterációk számát.

```
In [126]: # Értéktartomány (range) létrehozása.
          print(range(10))       # range objektum
          print(list(range(10))) # ugyanez, listává alakítva
```

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [127]: # Példa: Első n négyzetszám kiírása.
          n = int(input('n: '))
          for i in range(1, n + 1):
              print(i**2)
```

```
n: 10
1
4
```

```
9
16
25
36
49
64
81
100
```

In [128]:
```
# Példa n-szer n-es háromszög * karakterekből.
# *
# * *
# * * *
# * * * *

n = int(input('n: '))
for i in range(n): # végigmegyünk a sorokon
    # i + 1 db * kiírása
    for j in range(i + 1):
        print(' *', end='')
    # sortörés
    print()
```

```
n: 5
 *
 * *
 * * *
 * * * *
 * * * * *
```

In [129]:
```
# Ugyanez tömörebben...
n = int(input('n: '))
for i in range(n): # végigmegyünk a sorokon
    print(' *' * (i + 1))
```

```
n: 5
 *
 * *
 * * *
 * * * *
 * * * * *
```

In [130]:
```
# Példa: Magánhangzók megszámolása (angol kisbetűs szövegben).
s = input('Kérek egy szöveget: ')
```

```
vowels = set('aeiou')
nvowels = 0
for ch in s:
    if ch in vowels: # ha az adott karakter magánhangzó
        nvowels += 1
print(nvowels)
```

```
Kérek egy szöveget: apple tree
4
```

```
In [131]: # Példa: Caesar-kódolás (latin kisbetűs szövegre, szóköz nélkül).
          s = input('Kérek egy szöveget: ')
          offset = 3
          for ch in s:
              idx = ord(ch) - ord('a')
              ch_encoded = chr((idx + offset) % 26 + ord('a'))
              print(ch_encoded, end='')
```

```
Kérek egy szöveget: venividivici
yhqlylglylfl
```

Gyakorlás: Egyszerű számkitalálós játék

Készítsünk programot, amely sorsol egy egész számot 1-től 100-ig, majd tippeket kér a játékostól, amíg a játékos el nem találja a számot. A program minden tipp után írja ki, hogy a megadott tipp túl kicsi, túl nagy vagy helyes volt-e!

```
In [132]: # Az (ál)véletlenszám-generáló modul importálása.
          import random

In [133]: # Véletlen egész szám kisorsolása 1 és 100 között.
          x = random.randint(1, 100)

In [134]: y = 0
          while y != x:
              # tipp bekérése
              y = int(input('Tipp: '))

              # elágazás (3 ágú)
              if y > x:
                  print('Túl nagy.')
              elif y < x:
                  print('Túl kicsi.')
              else:
                  print('Eltaláltad!')
```

```
Tipp: 50
Túl kicsi.
Tipp: 75
Túl kicsi.
Tipp: 87
Túl nagy.
Tipp: 80
Túl kicsi.
Tipp: 82
Túl kicsi.
Tipp: 83
Túl kicsi.
Tipp: 84
Eltaláltad!
```

8. Comprehension-ök

A comprehension egy olyan nyelvi elem a Pythonban, amely szekvenciák tömör megadását teszi lehetővé. A comprehension némileg hasonlít a matematikában alkalmazott, tulajdonság alapján történő halmazmegadásra (példa: a páratlan számok halmaza megadható $\{2k+1 \mid k \in \mathbb{Z}\}$ módon).

8.1. Feltétel nélküli comprehension

```
In [135]: # Állítsuk elő az első 10 négyzetszám listáját gyűjtőváltozó használatával!
          l = []
          for i in range(1, 11):
              l.append(i**2)
          print(l)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
In [136]: # Ugyanez tömörebben, lista comprehension-nel:
          l = [i**2 for i in range(1, 11)]
          print(l)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
In [137]: # Állítsuk elő az első 10 négyzetszám halmazát gyűjtőváltozó használatával!
          s = set()
          for i in range(1, 11):
              s.add(i**2)
          print(s)
```

```
{64, 1, 4, 36, 100, 9, 16, 49, 81, 25}
```

```
In [138]: # Ugyanez tömörebben, halmaz comprehension-nel:
          s = {i**2 for i in range(1, 11)}
          print(s)
```

```
{64, 1, 4, 36, 100, 9, 16, 49, 81, 25}
```

```
In [139]: # Állítsunk elő egy szótárat, amely az angol kisbetűs magánhangzókhoz
          # hozzárendeli az ASCII-kódjukat! Használjunk gyűjtőváltozót!
          d = {}
          for ch in 'aeiou':
              d[ch] = ord(ch)
          print(d)
```

```
{'o': 111, 'e': 101, 'u': 117, 'i': 105, 'a': 97}
```

```
In [140]: # Ugyanez tömörebben, szótár comprehension-nel:
          d = {ch: ord(ch) for ch in 'aeiou'}
          print(d)
```

```
{'o': 111, 'e': 101, 'u': 117, 'i': 105, 'a': 97}
```

```
In [141]: # Feladat: Párok tagjainak megcserélése egy listában.
          pairs = [('alma', 10), ('körte', 20), ('barack', 30)]
          [(p[1], p[0]) for p in pairs]
```

```
Out[141]: [(10, 'alma'), (20, 'körte'), (30, 'barack')]
```

8.2. Feltételes comprehension

```
In [142]: # Feltételes lista comprehension.
          [i**2 for i in range(1, 11) if i % 2 == 0]
```

```
Out[142]: [4, 16, 36, 64, 100]
```

```
In [143]: # Feltételes halmaz comprehension.
          {i**2 for i in range(1, 11) if i % 2 == 0}
```

```
Out[143]: {4, 16, 36, 64, 100}
```

```
In [144]: # Feltételes szótár comprehension.
          {ch: ord(ch) for ch in 'aeiou' if ch != 'u'}
```

```
Out[144]: {'a': 97, 'e': 101, 'i': 105, 'o': 111}
```

9. Rendezés

```
In [145]: # Lista rendezése helyben.
          l = [10, 2, 11, 3]
          l.sort()

In [146]: l

Out[146]: [2, 3, 10, 11]

In [147]: # Rendezés csökkenő sorrendbe.
          l.sort(reverse=True)
          l

Out[147]: [11, 10, 3, 2]

In [148]: # Fontos, hogy az elemek összehasonlíthatók legyenek!
          q = [1, 2, 'alma']
          q.sort()
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-16-feabf817ddaa> in <module>()
      1 # Fontos, hogy az elemek összehasonlíthatók legyenek!
      2 q = [1, 2, 'alma']
----> 3 q.sort()


TypeError: unorderable types: str() < int()
```

```
In [149]: # Ha csak sztringeket tartalmaz a lista, akkor lehet rendezni.
          q = ['alma', 'szilva', 'körte']
          q.sort()
          q

Out[149]: ['alma', 'körte', 'szilva']

In [150]: # Kollekció rendezése listába.
          l1 = [10, 4, 20, 5]
          l2 = sorted(l1)

In [151]: l1

Out[151]: [10, 4, 20, 5]

In [152]: l2
```

```
Out[152]: [4, 5, 10, 20]

In [153]: # Tuple elemeit is rendezhetjük egy új listába.
          sorted(('c', 'a', 'b'))

Out[153]: ['a', 'b', 'c']

In [154]: # ...és halmaz elemeit is.
          sorted({'c', 'a', 'b'})

Out[154]: ['a', 'b', 'c']

In [155]: # Szótár esetén a sorted a kulcsokat rendezi.
          sorted({'b': 10, 'a': 20})

Out[155]: ['a', 'b']

In [156]: # Párok listájának rendezése (lexikografikusan).
          sorted([('sör', 10), ('bor', 20), ('pálinka', 30), ('bor', 50)])

Out[156]: [('bor', 20), ('bor', 50), ('pálinka', 30), ('sör', 10)]
```

Megjegyzés: A Python stabil rendezőalgoritmust alkalmaz.

10. Fájlkezelés

A fájl valamilyen adathordozón tárolt, logikailag összefüggő adatok összessége. Egy fájl életciklusa a következő lépésekből áll:

1. megnyitás
2. olvasás, írás, pozícionálás, ...
3. bezárás.

```
In [157]: # Fájl megnyitása.
          f = open('probafajl.txt')
```

A `probafajl.txt`-t a munkakönyvtárban a New / Text File menüponttal hozhatjuk létre.

```
In [158]: # Fájl bezárása.
          f.close()

In [159]: # Fájl tartalmának beolvasása sztringbe.
          f = open('probafajl.txt')
          s = f.read()
          f.close()
          s

Out[159]: '# mérési adatok\nalma,10\nkörte,20\nszilva,30\n'
```

```
In [160]: # ...ugyenez rövidebben:
          s = open('probafajl.txt').read()
          s

Out[160]: '# mérési adatok\nalma,10\nkörte,20\nszilva,30\n'

In [161]: # Egyetlen sor beolvasása.
          f = open('probafajl.txt')
          print(f.readline())
          print(f.readline())
          f.close()
```

```
# mérési adatok

alma,10
```

```
In [162]: # Megjegyzés: A readline a sortörést is beteszi az eredménybe.
          open('probafajl.txt').readline()

Out[162]: '# mérési adatok\n'

In [163]: # Ha szeretnénk levágni a sortörést:
          open('probafajl.txt').readline().strip()

Out[163]: '# mérési adatok'

In [164]: # Fájl sorainak beolvasása sztringlistába.
          open('probafajl.txt').readlines()

Out[164]: ['# mérési adatok\n', 'alma,10\n', 'körte,20\n', 'szilva,30\n']

In [165]: # Kitérő: Sztring darabolása egy határoló jelsorozat mentén (tokenizálás).
          'alma,körte,barack'.split(',')

Out[165]: ['alma', 'körte', 'barack']

In [166]: # Megjegyzés: alapértelmezés szerint a split fehér karakterek mentén darabol.
          'alma  körte\t barack'.split()

Out[166]: ['alma', 'körte', 'barack']

In [167]: # Iterálás egy szövegfájl sorain.
          for line in open('probafajl.txt'):
              print(line)
```

```
# mérési adatok

alma,10

körte,20

szilva,30
```

```
In [168]: # Fájl első sorának átugrása, a további sorok tokenizálása.
          data = []
          f = open('probafajl.txt')
          f.readline()
          for line in f:
              tok = line.strip().split(',')
              data.append((tok[0], int(tok[1])))
          f.close()
          data

Out[168]: [('alma', 10), ('körte', 20), ('szilva', 30)]

In [169]: # Sztring fájlba írása.
          open('probafajl2.txt', 'w').write('valami')

Out[169]: 6

In [170]: # Celsius-Fahrenheit táblázatot tartalmazó fájl elkészítése.
          file = open('celsius_fahrenheit.txt', 'w')
          for c in range(-20, 41, 5):
              f = c * 9 / 5 + 32
              file.write('{:4}\t{:4}\n'.format(c, f))
          file.close()

In [171]: # Határozzuk meg az igazi.txt szövegfájlban található szavak halmazát!
          {line.strip() for line in open('igazi.txt')}

Out[171]: {'a',
           'az',
           'csinálja',
           'egyáltalán',
           'fortranban',
           'gépidőelszámolást',
           'ha',
           'igazi',
           'intelligencia',
           'manipulációt',
           'megcsinálja',
```

```
 'mesterséges',
 'már',
 'programokat',
 'programozó',
 'szimbólum',
 'szövegkezelést'}
```

```
In [172]: # Olvassuk be a matrix.txt szövegfájl tartalmát egész számok listájának listájába!
          matrix = []
          for line in open('matrix.txt'):
              matrix.append([int(t) for t in line.split()])
          matrix
```

```
Out[172]: [[0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
           [0, 0, 1, 0, 1, 1, 0, 1, 0, 1],
           [0, 0, 1, 0, 0, 0, 1, 1, 0, 0],
           [0, 1, 0, 0, 1, 0, 1, 1, 0, 0],
           [1, 0, 1, 1, 0, 0, 1, 0, 1, 1],
           [1, 0, 1, 0, 0, 1, 1, 0, 1, 0],
           [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
           [0, 0, 0, 0, 0, 1, 0, 1, 0, 1],
           [1, 1, 0, 1, 0, 1, 1, 1, 0, 0],
           [1, 0, 1, 0, 1, 0, 0, 1, 0, 1]]
```

```
In [173]: # Ugyanez dupla comprehension-nel:
          [[int(t) for t in l.split()] for l in open('matrix.txt')]
```

```
Out[173]: [[0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
           [0, 0, 1, 0, 1, 1, 0, 1, 0, 1],
           [0, 0, 1, 0, 0, 0, 1, 1, 0, 0],
           [0, 1, 0, 0, 1, 0, 1, 1, 0, 0],
           [1, 0, 1, 1, 0, 0, 1, 0, 1, 1],
           [1, 0, 1, 0, 0, 1, 1, 0, 1, 0],
           [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
           [0, 0, 0, 0, 0, 1, 0, 1, 0, 1],
           [1, 1, 0, 1, 0, 1, 1, 1, 0, 0],
           [1, 0, 1, 0, 1, 0, 0, 1, 0, 1]]
```

Gyakorlás: Szóstatisztika

A `hamlet.txt` fájl a Hamlet angol nyelvű szövegkönyvét tartalmazza. Készíts programot, amely kiszámítja majd kiírja a szövegkönyvben szereplő 30 leggyakoribb szót! A szó definíciója a következő legyen:

- A szavakat a fehér karakterek (szóköz, tabulátor, soremelés) választják el egymástól.
- A kis- és nagybetűk ne számítsanak különbözőnek!
- A szó elején és végén található központozás karakterek ne számítsanak bele a szóba!

```
In [174]: # beolvasás kisbetűs szavak listájába
          words = open('hamlet.txt').read().lower().split()

In [176]: # központozás eltávolítása
          import string
          words2 = [w.strip(string.punctuation) for w in words]

In [177]: # szógyakoriságok kiszámítása
          freq = {}
          for w in words2:
              if w in freq: freq[w] += 1
              else: freq[w] = 1

In [178]: # rendezés gyakoriság szerint
          freq2 = sorted([(x[1], x[0]) for x in freq.items()], reverse=True)

          # leggyakoribb 50 szó kiírása
          for i in range(50):
              print(freq2[i])
```

```
(1145, 'the')
(973, 'and')
(736, 'to')
(674, 'of')
(565, 'i')
(539, 'you')
(534, 'a')
(513, 'my')
(431, 'in')
(409, 'it')
(381, 'that')
(358, 'ham')
(339, 'is')
(310, 'not')
(297, 'this')
(297, 'his')
(268, 'with')
(258, 'but')
(248, 'for')
(241, 'your')
(231, 'me')
(223, 'lord')
(219, 'as')
(216, 'be')
(213, 'he')
(200, 'what')
(195, 'king')
(195, 'him')
```

```
(194, 'so')
(180, 'have')
```

11. Kicsomagolás (unpacking)

```
In [179]: # Általános eset.
          x, (y, z) = [10, (20, [30, 40])]

In [180]: x

Out[180]: 10

In [181]: y

Out[181]: 20

In [182]: z

Out[182]: [30, 40]

In [183]: # Többszörös értékadás.
          a, b = 20, 30
          print(a)
          print(b)
```

```
20
30
```

```
In [184]: # Csere.
          a, b = b, a
          print(a)
          print(b)
```

```
30
20
```

```
In [185]: # Kicsomagolás alkalmazása for ciklusnál.
          data = [('a', 1), ('b', 2), ('c', 3)]
          for x, y in data:
              print(x)
              print(y)
```

```
a
1
b
2
c
3
```

12. Haladó indexelés (slicing)

A slice jelölésmód szintaxisa: [alsó határ: felső határ: lépésköz]. A kiválasztás intervalluma felülről nyitott, azaz a felső határ adja meg az első olyan indexet, amelyet már éppen nem választunk ki.

```
In [186]: # Példa haladó indexelésre.
          x = ['alma', 'körte', 10, 20]
          x[1:4:2]

Out[186]: ['körte', 20]

In [187]: # Az alsó határ, a felső határ és a lépésköz is elhagyható.
          x[1:]

Out[187]: ['körte', 10, 20]

In [188]: x[:2]

Out[188]: ['alma', 'körte']

In [189]: x[::-1]

Out[189]: [20, 10, 'körte', 'alma']

In [190]: x[3:0:-1]

Out[190]: [20, 10, 'körte']

In [191]: x[::2] # páros indexű elemek

Out[191]: ['alma', 10]

In [192]: x[1::2] # páratlan indexű elemek

Out[192]: ['körte', 20]

In [193]: # Használhatunk negatív indexeket is, a mínusz 1-edik jelenti az utolsó elemet.
          x[-1] # utolsó elem

Out[193]: 20
```

```
In [194]: x[-2] # utolsó előtti elem

Out[194]: 10

In [195]: x[:-1] # az összes elem, kivéve az utolsó

Out[195]: ['alma', 'körte', 10]

In [196]: x[-3:] # utolsó 3 elem

Out[196]: ['körte', 10, 20]
```

13. Haladó iterálási technikák

```
In [197]: # Iterálás az elemeken és indexeken egyszerre, hagyományos megoldás.
          x = ['alma', 'körte', 10, 20]
          for i in range(len(x)):
              print(x[i], i)
```

```
alma 0
körte 1
10 2
20 3
```

```
In [198]: # Ugyanez elegánsabban, enumerate-tel:
          for i, xi in enumerate(x):
              print(i, xi)
```

```
0 alma
1 körte
2 10
3 20
```

```
In [199]: list(enumerate(x))

Out[199]: [(0, 'alma'), (1, 'körte'), (2, 10), (3, 20)]

In [200]: for y in enumerate(x):
              print(y, y[0], y[1])
```

```
(0, 'alma') 0 alma
(1, 'körte') 1 körte
(2, 10) 2 10
(3, 20) 3 20
```

```
In [201]: # Szövegfájl feldolgozás esetén is használható az enumerate.
          for i, line in enumerate(open('igazi.txt')):
              print(i, line.strip())
```

```
0 az
1 igazi
2 programozó
3 a
4 szimbólum
5 manipulációt
6 fortranban
7 csinálja
8 az
9 igazi
10 programozó
11 a
12 szövegkezelést
13 fortranban
14 csinálja
15 az
16 igazi
17 programozó
18 a
19 gépidőelszámolást
20 már
21 ha
22 megcsinálja
23 egyáltalán
24 fortranban
25 csinálja
26 az
27 igazi
28 programozó
29 a
30 mesterséges
31 intelligencia
32 programokat
33 fortranban
34 csinálja
```

```
In [202]: # Iterálás több szekvencián egyszerre, hagyományos megoldás.
          x = ['sör', 'bor', 'pálinka']
          y = [10, 20, 30]
          for i in range(len(x)):
              print(x[i], y[i])
```

```
sör 10
```

```
bor 20
pálinka 30
```

```
In [203]: # Ugyanez elegánsabban, zip-pel:
          for xi, yi in zip(x, y):
              print(xi, yi)
```

```
sör 10
bor 20
pálinka 30
```

```
In [204]: list(zip(x, y))

Out[204]: [('sör', 10), ('bor', 20), ('pálinka', 30)]

In [205]: # Ha nem egyforma hosszú a 2 szekvencia,
          # akkor az eredmény a rövidebb hosszát veszi fel.
          x = ['sör', 'bor', 'pálinka', 'rum']
          y = [10, 20, 30]
          for xi, yi in zip(x, y):
              print(xi, yi)
```

```
sör 10
bor 20
pálinka 30
```

14. Függvények

A függvény névvel ellátott alprogram, amely a program más részeiből meghívható. Függvények használatával a számítási feladatok kisebb egységekre oszthatók. A gyakran használt függvények kódját könyvtárakba rendezhetjük. A matematikában egy függvénynek nincsenek mellékhatásai. Egy Python nyelvű függvénynek lehetnek! Pythonban a függvények „teljes jogú állampolgárok”:

- Egy változónak értékül adhatunk egy függvényt.
- Függvényeket lehet egymásba ágyazni.
- Egy függvény kaphat paraméterként függvényt ill. adhat eredményül függvényt.

```
In [206]: # Példa: n-edik gyök függvény.
          def root(x, n=2):
              '''Returns the n-th root of x.'''
              return x**(1 / n)

In [207]: # Dokumentációs sztring (docstring).
          root.__doc__ # 'dunder' doc
```

```
Out[207]: 'Returns the n-th root of x.'

In [208]: root.__doc__ *= 2
          print(root.__doc__)
```

```
Returns the n-th root of x.Returns the n-th root of x.
```

```
In [209]: # Gyök 2 kiszámítása.
          root(2)

Out[209]: 1.4142135623730951

In [210]: # Köbgyök 2 kiszámítása.
          root(2, n=3)

Out[210]: 1.2599210498948732

In [211]: # A második paramétert nem kell nevesíteni.
          root(2, 3)

Out[211]: 1.2599210498948732

In [212]: # Változónak értékül adhatunk függvényt.
          z = root
          z(2)

Out[212]: 1.4142135623730951

In [213]: # Példa egymásba ágyazásra ill. függvényt visszaadó függvényre.
          def f(x):
              def g(y):
                  return x + y
              return g

          h = f(42)

In [214]: type(h)

Out[214]: function

In [215]: h(100)

Out[215]: 142

In [216]: # Feladat: Készítsünk függvényt,
          # amely eldönti egy természetes számról, hogy prím-e!
          def is_prime(n):
              for i in range(2, int(n**0.5) + 1):
                  if n % i == 0:
                      return False
              return n != 1
```

```
In [217]: is_prime(1)
```

```
Out[217]: False
```

```
In [218]: is_prime(2)
```

```
Out[218]: True
```

```
In [219]: is_prime(13)
```

```
Out[219]: True
```

```
In [220]: is_prime(14)
```

```
Out[220]: False
```

```
In [221]: # Feladat: Készítsünk függvényt
          # két természetes szám legnagyobb közös osztójának a meghatározására!
          def gcd(a, b):
          for i in range(min(a, b), 0, -1):
              if a % i == 0 and b % i == 0:
                  return i
```

```
In [222]: gcd(12, 16)
```

```
Out[222]: 4
```

```
In [223]: gcd(1, 100)
```

```
Out[223]: 1
```

```
In [224]: gcd(17, 34)
```

```
Out[224]: 17
```

```
In [225]: # Feladat: Készítsünk másodfokú egyenlet megoldó függvényt!
          def solve_quadratic(a, b, c):
          d = b**2 - 4 * a * c
          if d > 0:
              x1 = (-b + d**0.5) / (2 * a)
              x2 = (-b - d**0.5) / (2 * a)
              return [x1, x2]
          elif d == 0:
              return [-b / (2 * a)]
          else:
              return []
```

```
In [226]: solve_quadratic(1, 3, 2)
```

```
Out[226]: [-1.0, -2.0]
```

```
In [227]: solve_quadratic(1, 2, 1)
```

```
Out[227]: [-1.0]
```

```
In [228]: solve_quadratic(1, 1, 10)
```

```
Out[228]: []
```

15. A funkcionális programozás néhány eleme

- lambda kifejezés: egysoros, névtelen függvény.
- map: függvény alkalmazása egy szekvencia elemeire.
- filter: szűrőfeltétel alkalmazása egy szekvenciára.

```
In [229]: # Példa lambda kifejezésre.
          f = lambda x: x**2

In [230]: type(f)

Out[230]: function

In [231]: f(3)

Out[231]: 9

In [232]: # Egynél több bemenet is megengedett.
          g = lambda x, y: x + y
          g(2, 3)

Out[232]: 5

In [233]: # ...de akár nulla bemenet is lehet.
          h = lambda: 42
          h()

Out[233]: 42

In [234]: # Lambda kifejezés alkalmazása rendezésnél:
          # Rendezzük párok listáját a második elem szerint!

          l = [(20, 'körte'), (30, 'szilva'), (40, 'alma')]
          sorted(l, key=lambda x: x[1])

Out[234]: [(40, 'alma'), (20, 'körte'), (30, 'szilva')]

In [235]: # Az előző feladat megoldása lambda kifejezés nélkül.

          def key_function(x):
              return x[1]

          sorted(l, key=key_function)

Out[235]: [(40, 'alma'), (20, 'körte'), (30, 'szilva')]

In [236]: # Példa a map függvény alkalmazására:
          # Hány elem van összesen az alábbi összetett adatszerkezetben?
          data = [{1, 2}, {3}, {4, 5, 6}]
```

```
In [237]: # Megoldás map-pel:
          sum(map(len, data))

Out[237]: 6

In [238]: # Ugyanez lista comprehension-nel:
          sum([len(x) for x in data])

Out[238]: 6

In [239]: # Olvassunk be 2 db, szóközzel határolt, egész számot!
          i, j = map(int, input("Kérek két egész számot: ").split())
          print(i)
          print(j)
```

```
Kérek két egész számot: 11 14
11
14
```

```
In [240]: # Példa a filter függvény alkalmazására:
          # Válasszuk ki a páros számokat az alábbi listából,
          # és adjuk vissza az eredményt tuple-ként!
          data = [2, 3, 5, 6, 7, 8]

In [241]: # Megoldás filter-rel:
          tuple(filter(lambda x: x % 2 == 0, data))

Out[241]: (2, 6, 8)

In [242]: # Ugyanez lista comprehension-nel:
          tuple([x for x in data if x % 2 == 0])

Out[242]: (2, 6, 8)
```

Gyakorlás: Premier League tabella

A `pl.txt` szövegfájl a Premier League 2011-12-es szezonjának eredményeit tartalmazza. Készíts programot, amely kiírja, hogy

- a mérkőzések hány százalékán esett gól, és melyik mérkőzésen esett a legtöbb gól,
- bekéri a felhasználótól n értékét, majd kiírja a bajnokság állását az n. forduló után (rendezési szempontok: pontszám, gólkülönbség, több rúgott gól)!

```
In [243]: # Adatok beolvasása szótárak listájába.
          f = open('pl.txt')
          for i in range(6): f.readline()
          games = []
          for line in f:
```

```
            t = line.strip().split('\t')
            games.append({
                'round': int(t[0]),
                'hteam': t[1],
                'ateam': t[2],
                'hgoals': int(t[3]),
                'agoals': int(t[4]),
            })
        f.close()
```

```
In [244]: # A mérkőzések hány százalékán esett gól?
          sum([g['hgoals'] + g['agoals'] > 0 for g in games]) / len(games) * 100
```

```
Out[244]: 92.89473684210526
```

```
In [245]: # Melyik mérkőzésen esett a legtöbb gól?
          max(games, key=lambda x: x['hgoals'] + x['agoals'])
```

```
Out[245]: {'agoals': 2,
          'ateam': 'Arsenal FC',
          'hgoals': 8,
          'hteam': 'Manchester United',
          'round': 3}
```

```
In [246]: # Tabella az n. forduló után.

          def update_stats(stats, hteam, hgoals, agoals):
              gdiff = hgoals - agoals
              if gdiff > 0: stats[hteam][0] += 3
              elif gdiff == 0: stats[hteam][0] += 1
              stats[hteam][1] += gdiff
              stats[hteam][2] += hgoals

          # n bekérése
          n = int(input('n: '))

          # statisztikák (pontszám, gólkülönbség, rúgott gólok) inicializálása
          stats = {g['hteam']: [0, 0, 0] for g in games}

          # statisztikák kiszámítása
          for g in games:
              if g['round'] <= n:
                  update_stats(stats, g['hteam'], g['hgoals'], g['agoals'])
                  update_stats(stats, g['ateam'], g['agoals'], g['hgoals'])

          # rendezés
          standings = sorted(stats.items(), key=lambda x: tuple(x[1]), reverse=True)

          # kiírás
```

```
        for team, s in standings:
            print('{:25}{:5}{:5}{:5}'.format(team, s[1], s[2], s[0]))
```

```
n: 10
Manchester City             28   36   28
Manchester United           15   27   23
Newcastle United             8   15   22
Tottenham Hotspur            6   20   22
Chelsea FC                   8   23   19
Liverpool FC                 4   14   18
Arsenal FC                  -1   20   16
Norwich City                -1   14   13
Aston Villa                  0   13   12
Swansea City                -3   12   12
Stoke City                  -6    8   12
Queens Park Rangers         -9    8   12
West Bromwich Albion        -4    9   11
Sunderland AFC               2   14   10
Fulham FC                    1   13   10
Everton FC                  -5   10   10
Wolverhampton Wanderers     -8    9    8
Blackburn Rovers           -10   13    6
Bolton Wanderers           -14   13    6
Wigan Athletic             -11    6    5
```

16. Modulok és csomagok

- **Modul**: Python nyelvű fájl. Definíciókat és utasításokat tartalmaz. Ha a modulhoz az `xyz.py` fájl tartozik, akkor a modulra `xyz` néven lehet hivatkozni. A modulok más Python programokból importálhatók.
- **Csomag**: Modulok gyűjteménye. Egy csomag alcsomagokat/modulokat is tartalmazhat. A hierarchiát a csomagon belüli könyvtárszerkezet határozza meg. A standard csomagok és modulok a standard könyvtárban találhatók, és nem igényelnek telepítést. A külső csomagok gyűjtőhelye a PyPI.

```
In [247]: # Modul/csomag importálása.
          import random

In [248]: random.randint(1, 10)

Out[248]: 6

In [249]: # Függvény importálása a egy modulból/csomagból.
          from random import randint

In [250]: randint(1, 10)
```

```
Out[250]: 3

In [251]: # Modul/csomag teljes tartalmának importálása.
          # (Megjegyzés: Ez a megoldás általában kerülendő.)
          from random import *

In [252]: # Függvény importálása almodulból/alcsomagból.
          from os.path import dirname

In [253]: dirname('/tmp/moricka/a.txt')

Out[253]: '/tmp/moricka'

In [254]: # Modul/csomag importálása rövidített néven.
          import numpy as np
          # Megjegyzés: A numpy egy külső csomag.

In [255]: np.cos(0)

Out[255]: 1.0
```

17. Fejezetek a standard könyvtárból

A Python standard könyvtára több mint 200 csomagot ill. modult tartalmaz. Szabványos megoldást biztosít a programozás mindennapjaiban felmerülő számos feladatra. A kurzuson csak a standard könyvtár egy kis részének az áttekintésére vállalkozunk. A jó programozó nem találja fel újra a spanyolviaszt. Ha lehetséges, akkor a standard könyvtár eszközeivel oldja meg a feladatot.

17.1. datetime

- Dátum- és időkezelésre biztosít eszközöket.
- Támogatja a dátumaritmetikát, kezeli az időzónákat, óraátállítást, szökőéveket stb.
- Időzónamentes és időzónával rendelkező dátumokat is megenged.

```
In [256]: import datetime

In [257]: # Mikroszekundum pontosságú időpont megadása.
          t = datetime.datetime(2019, 4, 3, 10, 20, 30, 567)

In [258]: t

Out[258]: datetime.datetime(2019, 4, 3, 10, 20, 30, 567)

In [259]: print(t)
```

```
2019-04-03 10:20:30.000567
```

```
In [260]: # Nap pontosságú dátum megadása.
          d = datetime.date(2019, 4, 3)
```

```
In [261]: d

Out[261]: datetime.date(2019, 4, 3)

In [262]: print(d)
```

```
2019-04-03
```

```
In [263]: # Időpont aritmetika.
          t1 = datetime.datetime(2019, 4, 3, 10, 20, 30, 567)
          t2 = datetime.datetime(2019, 4, 5, 16, 30, 40, 100000)
          dt = t2 - t1

In [264]: dt

Out[264]: datetime.timedelta(2, 22210, 99433)

In [265]: dt.days

Out[265]: 2

In [266]: dt.seconds

Out[266]: 22210

In [267]: dt.microseconds

Out[267]: 99433

In [268]: dt.total_seconds()

Out[268]: 195010.099433

In [269]: # Aritmetika nap pontosságú dátumokkal.
          datetime.date(2018, 12, 30) + datetime.timedelta(65)

Out[269]: datetime.date(2019, 3, 5)

In [270]: # Aktuális idő lekérdezése.
          datetime.datetime.now()

Out[270]: datetime.datetime(2019, 4, 4, 12, 52, 38, 664130)

In [271]: # A datetime objektum mezőinek elérése.
          print(t.year, t.month, t.day, t.hour, t.minute, t.second, t.microsecond)
```

```
2019 4 3 10 20 30 567
```

```
In [272]: # Feladat: Készítsünk programot, amely bekér egy dátumot,
          # majd kiírja, hogy a dátum hányadik nap az adott évben!
          import datetime

          # dátum bekérése
          tok = input('Kérek egy dátumot (éééé-hh-nn): ').split('-')
          y, m, d = int(tok[0]), int(tok[1]), int(tok[2])

          # az év hányadik napja?
          dt1 = datetime.datetime(y, m, d)
          dt2 = datetime.datetime(y, 1, 1)
          print('Az év {}. napja.'.format((dt1 - dt2).days + 1))
```

```
Kérek egy dátumot (éééé-hh-nn): 2013-03-01
Az év 60. napja.
```

```
In [273]: # Feladat: Készítsünk programot, amely életkor szerint növekvő
          # sorrendben írja ki az alábbi listában szereplő neveket!
          people = [
              # név            születési dátum
              ('Gipsz Jakab',  datetime.date(1957, 11, 21)),
              ('Wincs Eszter', datetime.date(1980, 5, 7)),
              ('Békés Farkas', datetime.date(2014, 7, 30)),
              ('Har Mónika',   datetime.date(1995, 2, 27)),
              ('Trab Antal',   datetime.date(1961, 4, 1)),
              ('Git Áron',     datetime.date(1995, 2, 28)),
              ('Bank Aranka',  datetime.date(1980, 9, 1))
          ]

In [274]: # Megoldás:
          sorted(people, key=lambda x: x[1], reverse=True)

Out[274]: [('Békés Farkas', datetime.date(2014, 7, 30)),
          ('Git Áron', datetime.date(1995, 2, 28)),
          ('Har Mónika', datetime.date(1995, 2, 27)),
          ('Bank Aranka', datetime.date(1980, 9, 1)),
          ('Wincs Eszter', datetime.date(1980, 5, 7)),
          ('Trab Antal', datetime.date(1961, 4, 1)),
          ('Gipsz Jakab', datetime.date(1957, 11, 21))]
```

17.2. time

Alacsony szintű időkezelésre ad eszközöket, ide tartozik pl. az időtartam mérés és a várakozás.

```
In [275]: import time

In [276]: # Aktuális idő lekérdezése (UNIX időbélyegként).
          time.time()
```

```
Out[276]: 1554376018.1734283

In [277]: # Időtartam mérés.
          t0 = time.time()
          s = 0
          x = 2
          for i in range(1000000):
              s += x**0.5
          t1 = time.time()
          t1 - t0

Out[277]: 0.4853982925415039

In [278]: # Várakozás 2 másodpercig.
          time.sleep(2)
```

17.3. math

Alapvető matematikai függvényeket tartalmaz.

```
In [279]: import math

In [280]: # Exponenciális függvény.
          math.exp(2)

Out[280]: 7.38905609893065

In [281]: # Természetes alapú logaritmus.
          math.log(8)

Out[281]: 2.0794415416798357

In [282]: # q alapú logaritmus.
          math.log(8, 2)

Out[282]: 3.0

In [283]: # Trigonometrikus függvények és inverzeik.
          math.sin(0)

Out[283]: 0.0

In [284]: math.cos(0)

Out[284]: 1.0

In [285]: math.tan(0)

Out[285]: 0.0

In [286]: math.asin(0)
```

```
Out[286]: 0.0

In [287]: math.acos(0)

Out[287]: 1.5707963267948966

In [288]: math.atan(1)

Out[288]: 0.7853981633974483

In [289]: # pi, e

In [290]: math.pi

Out[290]: 3.141592653589793

In [291]: math.e

Out[291]: 2.718281828459045

In [292]: # Feladat: Készítsünk programot, amely bekér egy síkbeli vektort,
          # majd kiírja a vektor x tengely pozitív felével bezárt szögét!

          # vektor bekérése, 1. megoldás
          vx = float(input('vx: '))
          vy = float(input('vy: '))
```

```
vx: 1
vy: 1
```

```
In [293]: # vektor bekérése, 2. megoldás
          vx, vy = map(float, input('v: ').split())
```

```
v: 1 1
```

```
In [294]: # vektor bekérése, 3. megoldás
          vx, vy = [float(x) for x in input('v: ').split()]
```

```
v: 1 1
```

```
In [295]: # szög kiszámítása, 1. megoldás
          import math

          wx, wy = 1, 0
          p = vx * wx + vy * wy
          vnorm = (vx**2 + vy**2)**0.5
          wnorm = (wx**2 + wy**2)**0.5
          alpha = math.degrees(math.acos(p / (vnorm * wnorm)))
          print(alpha)
```

```
45.00000000000001
```

```
In [296]: # szög kiszámítása, 2. megoldás
          vnorm = (vx**2 + vy**2)**0.5
          alpha = math.degrees(math.acos(vx / vnorm))
          print(alpha)
```

```
45.00000000000001
```

17.4. random

Álvéletlenszám-generálásra biztosít eszközöket.

```
In [297]: import random
```

```
In [298]: # Egész szám sorsolása egy intervallumból.
          random.randint(2, 10)
```

```
Out[298]: 6
```

```
In [299]: # Valós szám sorsolása egy intervallumból.
          random.uniform(2, 10)
```

```
Out[299]: 4.839874764251826
```

```
In [300]: # Sorsolás standard normális eloszlásból.
          random.normalvariate(0, 1)
```

```
Out[300]: 1.2105945539414045
```

```
In [301]: # Véletlenszám generátor állapotának beállítása.
          random.seed(42)
          print(random.uniform(2, 10))
          random.seed(42)
          print(random.uniform(2, 10))
```

```
7.11541438766307
7.11541438766307
```

```
In [302]: # Véletlenszám generátor objektum létrehozása.
          rnd1 = random.Random(43)
          rnd2 = random.Random(43)
          for i in range(5):
              print(rnd1.uniform(0, 1))
              print(rnd2.uniform(0, 1))
```

```
0.038551839337380045
0.038551839337380045
0.6962243226370528
0.6962243226370528
0.14393322139536102
0.14393322139536102
0.46253225482908755
0.46253225482908755
0.671646764117767
0.671646764117767
```

```
In [303]: # Elem kisorsolása egy szekvenciából.
          random.choice(['alma', 'körte', 'szilva'])
```

```
Out[303]: 'alma'
```

```
In [304]: # Visszatevés nélküli mintavétel.
          random.sample(range(1, 91), 5)
```

```
Out[304]: [12, 28, 30, 65, 78]
```

```
In [305]: # Feladat: Készítsünk programot, amely szimulál egy n hosszú
          # pénzfeldobás sorozatot, majd kiírja a fejek és írások darabszámát!
          import random
          n = 15
          seq = [random.choice('FI') for i in range(n)]
          print(seq)
```

```
['F', 'I', 'F', 'I', 'I', 'I', 'I', 'I', 'F', 'I', 'F', 'I', 'F', 'F', 'F']
```

```
In [306]: print('Fejek száma: {}'.format(seq.count('F')))
          print('Írások száma: {}'.format(seq.count('I')))
```

```
Fejek száma: 7
Írások száma: 8
```

```
In [307]: # Készítsünk programot, amely szimulál egy n hosszú pénzfeldobás sorozatot,
          # majd kiírja a leghosszabb fej ill. írás sorozat hosszát!
          import random

          n = 15
          data = [random.choice('FI') for i in range(n)]
          print(data)
```

```
            def longest_sequence(data, sign):
                maxlen = 0
                actlen = 0
                for x in data:
                    if x == sign: actlen += 1
                    else: actlen = 0

                    if actlen > maxlen:
                        maxlen = actlen
                return maxlen

            print('Leghosszabb fej sorozat hossza: {}.'.format(longest_sequence(data, 'F')))
            print('Leghosszabb írás sorozat hossza: {}.'.format(longest_sequence(data, 'I')))
```

```
['I', 'F', 'I', 'F', 'I', 'F', 'F', 'F', 'I', 'F', 'I', 'I', 'I', 'I', 'F']
Leghosszabb fej sorozat hossza: 3.
Leghosszabb írás sorozat hossza: 4.
```

17.5. collections

Specializált konténer adattípusokat tartalmaz.

```
In [308]: import collections
```

```
In [309]: # Gyakoriságszámító szótár (Counter).
          s = 'abrakadabra'
          freq = collections.Counter(s)
          print(freq)
          print(freq['a'])
```

```
Counter({'a': 5, 'b': 2, 'r': 2, 'k': 1, 'd': 1})
5
```

```
In [310]: # A szavak gyakorisága a Hamletben, Counterrel kiszámolva:
          import string
          words = open('hamlet.txt').read().strip().split()
          words = [word.lower().strip(string.punctuation) for word in words]
          freq = collections.Counter(words)
          freq.most_common(30)

Out[310]: [('the', 1145),
           ('and', 973),
           ('to', 736),
           ('of', 674),
           ('i', 565),
```

```
 ('you', 539),
 ('a', 534),
 ('my', 513),
 ('in', 431),
 ('it', 409),
 ('that', 381),
 ('ham', 358),
 ('is', 339),
 ('not', 310),
 ('his', 297),
 ('this', 297),
 ('with', 268),
 ('but', 258),
 ('for', 248),
 ('your', 241),
 ('me', 231),
 ('lord', 223),
 ('as', 219),
 ('be', 216),
 ('he', 213),
 ('what', 200),
 ('king', 195),
 ('him', 195),
 ('so', 194),
 ('have', 180)]
```

```
In [311]: # Szótár, alapértelmezett értékkel (defaultdict).
          d = collections.defaultdict(list)

In [312]: # Új kulcs-érték pár hozzáadása.
          d['alma'] = [1, 2, 3]

In [313]: # Hivatkozás nem létező kulcsra.
          # Nem fog hibát adni, hanem a list() értéket rendeli a kulcshoz.
          d['szilva']
          d

Out[313]: defaultdict(list, {'alma': [1, 2, 3], 'szilva': []})

In [314]: # Hivatkozás nem létező kulcsra, majd egy elem hozzáfűzése.
          d['körte'].append(10)
          d

Out[314]: defaultdict(list, {'alma': [1, 2, 3], 'szilva': [], 'körte': [10]})

In [315]: # defaultdict konvertálása hagyományos szótárrá.
          d2 = dict(d)

In [316]: # Garantáltan sorrendtartó szótár.
          od = collections.OrderedDict()
```

```
od['aa'] = 10
od['bb'] = 20
od['cc'] = 30
for key in od:
    print(key)
```

```
aa
bb
cc
```

In [317]:
```
# Tuple, nevesített elemekkel (namedtuple).
Game = collections.namedtuple('Game', ['hteam', 'ateam', 'hgoals', 'agoals'])
g = Game('Arsenal', 'Chelsea', 2, 1)
```

In [318]:
```
# Tuple stílusú használat.
print(g[0], g[1], g[2], g[3])
```

```
Arsenal Chelsea 2 1
```

In [319]:
```
# Struktúra stílusú használat.
print(g.hteam, g.ateam, g.hgoals, g.agoals)
```

```
Arsenal Chelsea 2 1
```

In [320]:
```
# Feladat: Készítsünk programot, amely n db kockadobást szimulál 2 kockával,
# majd kiírja hogy a dobások összege hányszor volt 2, 3, ..., ill. 12!

from collections import Counter
from random import randint

n = 10000
data = [randint(1, 6) + randint(1, 6) for i in range(n)]
Counter(data)
```

Out[320]:
```
Counter({3: 597,
         11: 543,
         9: 1089,
         6: 1357,
         10: 886,
         8: 1334,
         5: 1087,
         12: 291,
         4: 809,
         7: 1728,
         2: 279})
```

17.6. copy

Sekély (shallow) és mély (deep) másoló függvényt tartalmaz.

```
In [321]: import copy
```

```
In [322]: # Pythonban az értékadás NEM végez másolást, csak hivatkozást hoz létre.
          a = [1, 2, 3]
          b = a
          a[0] = 10
          print(b)
```

```
[10, 2, 3]
```

```
In [323]: # Sekély másolat készítése egy listák listája objektumról.
          a = [1, 2, 3]
          b = copy.copy(a)
          a[0] = 20
          print(b)
```

```
[1, 2, 3]
```

```
In [324]: # A sekély másolat csak az adatszerkezet legfelső szintjén végez másolást.
          x = [10, 20]
          y = [30, 40]
          a = [x, y]
          b = copy.copy(a)
          a[0][0] = 100
          print(b)
```

```
[[100, 20], [30, 40]]
```

```
In [325]: # Mély másolat készítése egy listák listája objektumról.
          x = [10, 20]
          y = [30, 40]
          a = [x, y]
          b = copy.deepcopy(a)
          a[0][0] = 100
          print(b)
```

```
[[10, 20], [30, 40]]
```

17.7. glob

Tartalmaz egy függvényt adott mintára illeszkedő fájlnevek összegyűjtésére.

```
In [326]: import glob
```

```
In [327]: # .ipynb kiterjesztésű fájlok az aktuális könyvtárban.
          glob.glob('*.ipynb')
```

```
Out[327]: ['09.ipynb',
           '10.ipynb',
           '01.ipynb',
           '06.ipynb',
           '08.ipynb',
           '05.ipynb',
           '07.ipynb',
           '02.ipynb',
           '04.ipynb',
           '03.ipynb']
```

```
In [328]: # A fájlneveket a sorted függvénnyel tudjuk rendezni.
          sorted(glob.glob('*.ipynb'))
```

```
Out[328]: ['01.ipynb',
           '02.ipynb',
           '03.ipynb',
           '04.ipynb',
           '05.ipynb',
           '06.ipynb',
           '07.ipynb',
           '08.ipynb',
           '09.ipynb',
           '10.ipynb']
```

17.8. gzip

GZIP formátumú tömörített fájlok olvasására és írására biztosít eszközöket. Megjegyzés: Egyéb tömörített formátumokat is támogat a standard könyvtár (pl. BZ2, LZMA, ZIP, TAR).

```
In [329]: import gzip
```

```
In [330]: # GZIP formátumú fájl elkészítése.
          f = gzip.open('moricka.txt.gz', 'wb')
          f.write('Móricka kedvence a Python.\n'.encode('utf-8'))
          f.close()
```

```
In [331]: # GZIP formátumú fájl beolvasása.
          gzip.open('moricka.txt.gz', 'rb').read().decode('utf-8')
```

```
Out[331]: 'Móricka kedvence a Python.\n'
```

17.9. os

Az operációs rendszer bizonyos szolgáltatásaihoz nyújt elérést.

```
In [332]: import os

In [333]: # Parancs futtatása.
          os.system('mkdir -p a/b')

Out[333]: 0

In [334]: # Milyen típusú fájl tartozik egy elérési útvonalhoz?

          os.path.isfile('09.ipynb') # (reguláris) fájl?

Out[334]: True

In [335]: os.path.isdir('09.ipynb') # könyvtár?

Out[335]: False

In [336]: os.path.exists('09.ipynb') # bármilyen (reguláris vagy speciális) fájl?

Out[336]: True

In [337]: # Előfordulhat, hogy, hogy az exists() és az isfile() eltérő eredményt ad:
          print(os.path.exists('/dev/null'))
          print(os.path.isfile('/dev/null'))
```

```
True
False
```

```
In [338]: # Könyvtárnév kinyerése egy elérési útvonalból.
          os.path.dirname('/tmp/pistike/titkos.txt')

Out[338]: '/tmp/pistike'

In [339]: # Környezeti változók elérése.
          os.environ['LANG']

Out[339]: 'hu_HU.UTF-8'
```

17.10. pickle

Python adatszerkezetek szerializálására (azaz bájtsorozattá alakítására) és deszerializálására nyújt egy megoldást. A „pickle” szó jelentése főnévként „ecetes lé”, „pác”, igeként „savanyítás” :-).

```
In [340]: import pickle
```

```
In [341]: # Összetett objektum szerializálása fájlba.
          data = [{'a': 1, 'b': 2}, {'c': 3, 'd': 4}]
          pickle.dump(data, open('data.pkl', 'wb'))

In [342]: # Deszerializálás.
          data2 = pickle.load(open('data.pkl', 'rb'))
          print(data2)
```

```
[{'a': 1, 'b': 2}, {'c': 3, 'd': 4}]
```

```
In [343]: # Két hasznos segédfüggvény.

          def topickle(obj, fname, protocol=4):
              pickle.dump(obj, open(fname, 'wb'), protocol)

          def frompickle(fname):
              return pickle.load(open(fname, 'rb'))

In [344]: topickle(data, 'data.pkl')
          frompickle('data.pkl')

Out[344]: [{'a': 1, 'b': 2}, {'c': 3, 'd': 4}]
```

17.11. subprocess

Alfolyamatok indítására és vezérlésére biztosít eszközöket.

```
In [345]: import subprocess

In [346]: # Folyamat indítása és a standard kimenet kinyerése sztringként.
          s = subprocess.getoutput('ls')
          print(s.split('\n')[:5])
```

```
['01.ipynb', '02.ipynb', '03.ipynb', '04.ipynb', '05.ipynb']
```

```
In [347]: # Interaktív parancssoros program vezérlése.
          p = subprocess.Popen(['python', '-i'], stdin=subprocess.PIPE, stdout=subprocess.PIPE)
          p.stdin.write('1 + 1\n'.encode('utf-8'))
          p.stdin.flush()
          p.stdout.readline().decode('utf-8')

Out[347]: '2\n'
```

17.12. urllib

Webcímek (URL-ek) megnyitására, beolvasására, kezelésére szolgáló csomag.

```
In [348]: from urllib.request import urlopen

In [349]: # Győr időjárási adatainak letöltése a
          # https://weather.com/weather/today/l/47.69,17.65 webcímről.
          url = 'https://weather.com/weather/today/l/47.69,17.65'
          data = urlopen(url).read().decode('utf-8')

In [350]: # Hőmérséklet kinyerése az adatokból.
          pattern = '<div class="today_nowcard-temp"><span class="">'
          f = int(data[data.index(pattern) + len(pattern):].split('<')[0])
          c = (f - 32) * 5 / 9
          print(c)
```

```
12.222222222222221
```

```
In [351]: # Csomagoljuk az egészet függvénybe!
          def get_gyor_temp():
              url = 'https://weather.com/weather/today/l/47.69,17.65'
              data = urlopen(url).read().decode('utf-8')
              pattern = '<div class="today_nowcard-temp"><span class="">'
              f = int(data[data.index(pattern) + len(pattern):].split('<')[0])
              c = (f - 32) * 5 / 9 # Fahrenheit => Celsius átalakítás
              return c

          get_gyor_temp()

Out[351]: 12.222222222222221

In [352]: # Speciális karakterek levédése
          # (pl. https://hu.wikipedia.org/wiki/Mesters%C3%A9ges_intelligencia).
          from urllib.request import quote, unquote
          print(quote('Mesterséges_intelligencia'))
          print(unquote('Mesters%C3%A9ges_intelligencia'))
```

```
Mesters%C3%A9ges_intelligencia
Mesterséges_intelligencia
```

Gyakorlás: Conway-féle életjáték

Készítsünk Python programot, amely a Conway-féle életjátékot valósítja meg. Készítsünk el a program procedurális és objektumorientált változatát is!

In [353]:
```
# Adjuk meg a kezdőállapotot egy sztringben!
# (Lehetne fájlból is beolvasni, de az egyszerűség kedvéért
# használjunk most sztringet!)

worldstr = '''
....................
....................
....................
....................
....................
....................
.........oo.........
........oo..........
.........o..........
....................
....................
....................
....................
....................
....................
....................
'''.strip()
```

In [354]:
```
import copy

# világ inicializálása
world = [list(l) for l in worldstr.split()]
nrows = len(world)
ncols = len(world[0])

while True:
    # világ kiírása
    for l in world: print(''.join(l))

    # világ frissítése
    new_world = copy.deepcopy(world)
    for i in range(2, nrows - 1): # végigmegyünk a sorokon
        for j in range(2, ncols - 1): # végigmegyünk az oszlopokon
            # élő szomszédok megszámolása
            c = 0
            for di in [-1, 0, 1]:
                for dj in [-1, 0, 1]:
                    if di == dj == 0: continue
                    if world[i + di][j + dj] == 'o': c += 1

            if c == 3: new_world[i][j] = 'o'
            elif c < 2 or c > 3: new_world[i][j] = '.'
```

```
            world = new_world

            input()
```

```
....................
....................
....................
....................
....................
....................
.........oo.........
........oo..........
.........o..........
....................
....................
....................
....................
....................
....................
....................

....................
....................
....................
....................
....................
....................
........ooo.........
........o...........
........oo..........
....................
....................
....................
....................
....................
....................
....................

        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        ~/.local/lib/python3.6/site-packages/ipykernel/kernelbase.py in
        _input_request(self, prompt, ident, parent, password)
        877             try:
    --> 878                 ident, reply = self.session.recv(self.stdin_socket, 0)
        879             except Exception:
...
```

```
In [355]: # (egy lehetséges) objektumorientált megoldás

          class GameOfLife:
              def __init__(self, worldstr):
```

```
            self.world = [list(l) for l in worldstr.split()]
            self.nrows = len(self.world)
            self.ncols = len(self.world[0])
            self.time = 0

        def show(self):
            for l in self.world: print(''.join(l))

        def count_neighbors(self, i, j):
            c = 0
            for di in [-1, 0, 1]:
                for dj in [-1, 0, 1]:
                    if di == dj == 0: continue
                    if self.world[i + di][j + dj] == 'o': c += 1
            return c

        def update(self):
            world = copy.deepcopy(self.world)
            for i in range(2, self.nrows - 1): # végigmegyünk a sorokon
                for j in range(2, self.ncols - 1): # végigmegyünk az oszlopokon
                    c = self.count_neighbors(i, j)
                    if c == 3: world[i][j] = 'o'
                    elif c < 2 or c > 3: world[i][j] = '.'
            self.world = world

        def run(self):
            while True:
                self.show()
                self.update()
                input()

    GameOfLife(worldstr).run()
```

```
..................
..................
..................
..................
..................
..................
.........oo.......
........oo........
.........o........
..................
..................
..................
..................
..................
..................
..................
```

```
.....................
.....................
.....................
.....................
.....................
.....................
........ooo..........
........o............
........oo...........
.....................
.....................
.....................
.....................
.....................
.....................
.....................
```

```
        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        ~/.local/lib/python3.6/site-packages/ipykernel/kernelbase.py in
        _input_request(self, prompt, ident, parent, password)
        877             try:
    --> 878                 ident, reply = self.session.recv(self.stdin_socket, 0)
        879             except Exception:
...
```

18. Kivételkezelés

A kivételkezelés egy modern megközelítés a hibakezelésre. Lehetővé teszi, hogy a legalkalmasabb helyen végezzük el a hibakezelést. A korábban uralkodó hibakód alapú módszer nehézkesebb. Tegyük fel, hogy a függvényhívási stack sokadik szintjén lép fel egy hiba. A hibát a kód több pontján kell kezelni (a hívó függvényben, a hívót hívó függvényben stb.), ami kódduplikáláshoz vagy GOTO utasítások alkalmazásához vezet. Pythonban a kivételeket a raise utasítás segítségével lehet létrehozni, és a try utasítás segítségével lehet elkapni. A beépített kivétel típusok hierarchiája itt tekinthető át.

```
In [356]: # Kivétel létrehozása.
          x = 11
          if x % 2 != 0:
              raise ValueError('x is odd')
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-54-64346145ed43> in <module>()
          2 x = 11
```

```
          3 if x % 2 != 0:
    ----> 4     raise ValueError('x is odd')


        ValueError: x is odd
```

```
In [357]: # Kivétel elkapása.
          try:
              x = 1 / 0
          except ZeroDivisionError:
              print('Division by zero!')
          finally:
              print('Executing finally clause.')
```

```
Division by zero!
Executing finally clause.
```

```
In [358]: try:
              math.acos(1.5)
          except ValueError:
              print('hiba')
```

```
hiba
```

```
In [359]: math.acos(1.5)
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-58-153a6d85bf10> in <module>()
    ----> 1 math.acos(1.5)


        ValueError: math domain error
```

19. Hibakeresés

```
In [360]: # Első lépés: A hibaüzenetet MINDIG olvassuk el! :-)
          ['a', 'b', 'c'][3]
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-35-4c1ec092f372> in <module>()
  1 # Első lépés: A hibaüzenetet MINDIG olvassuk el! :-)
----> 2 ['a', 'b', 'c'][3]


IndexError: list index out of range
```

```
In [361]: # Példa egy hibás függvényre.
          def calc_average(list_of_lists):
              joined = []
              for l in list_of_lists:
                  joined.append(l)
              return sum(joined) / len(joined)
```

```
In [362]: calc_average([[1, 2, 3], [4, 5], [6, 7]])
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-37-f6afa26f1001> in <module>()
----> 1 calc_average([[1, 2, 3], [4, 5], [6, 7]])


<ipython-input-36-507d7e387fff> in calc_average(list_of_lists)
  4     for l in list_of_lists:
  5         joined.append(l)
----> 6     return sum(joined) / len(joined)


TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

```
In [363]: # Keressük meg a hibát a %debug parancs segítségével!
          %debug
```

```
> <ipython-input-36-507d7e387fff>(6)calc_average()
      2 def calc_average(list_of_lists):
      3     joined = []
```

```
      4     for l in list_of_lists:
      5         joined.append(l)
----> 6     return sum(joined) / len(joined)

ipdb> print(joined)
[[1, 2, 3], [4, 5], [6, 7]]
ipdb> q
```

```
In [364]: # append helyett extend-et kellett volna használni.
          # A függvény javított változata:
          def calc_average(list_of_lists):
              joined = []
              for l in list_of_lists:
                  joined.extend(l)
              return sum(joined) / len(joined)

In [365]: calc_average([[1, 2, 3], [4, 5], [6, 7]])

Out[365]: 4.0
```

20. Objektumorientált programozás, nulladik közelítésben

Az objektumorientált programozás (OOP) olyan programozási módszertan, ahol az egymással kapcsolatban álló programegységek hierarchiájának megtervezése áll a középpontban. A korábban uralkodó procedurális megközelítés a műveletek megalkotására fókuszált. OOP esetén az adatokat és az őket kezelő függvényeket egységbe zárjuk (encapsulation). Az OOP másik fontos sajátossága az öröklődés (inheritance).

```
In [366]: # Példa: téglalap osztály.
          class Rectangle:
              def __init__(self, a, b): # konstruktor
                  self.a = a
                  self.b = b

              def area(self): # területszámító metódus
                  return self.a * self.b

              def perimeter(self): # kerületszámító metódus
                  return (self.a + self.b) * 2

          # 2 téglalap objektum létrehozása
          r1 = Rectangle(10, 20)
          r2 = Rectangle(15, 15)
          print(r1)
          print(r2)
```

```
# területek kiírása
print(r1.area())
print(r2.area())

# ugyanez hosszabban
print(Rectangle.area(r1))
print(Rectangle.area(r2))
```

```
<__main__.Rectangle object at 0x7f5b11cc9ac8>
<__main__.Rectangle object at 0x7f5b11cc9a20>
200
225
200
225
```

In [367]:
```
import math

# Példa: kör osztály.
class Circle:
    def __init__(self, r):
        self.r = r

    def area(self):
        return self.r**2 * math.pi

    def perimeter(self):
        return 2 * self.r * math.pi

# 2 kör objektum létrehozása
c1 = Circle(5)
c2 = Circle(10)

# területek kiírása
print(c1.area())
print(c2.area())
```

```
78.53981633974483
314.1592653589793
```

In [368]:
```
# A kerület-terület arány kiszámítása ugyanúgy történik a 2 esetben.
# Hozzunk létre egy egy síkidom ősosztályt, származtassuk ebből a kört és a
# téglalapot, és helyezzük át a kerület-terület arány számítást az ősosztályba!

class Shape:
    def area(self):
```

```
            raise NotImplementedError()

        def perimeter(self):
            raise NotImplementedError()

        def pa_ratio(self):
            return self.perimeter() / self.area()

    class Rectangle(Shape): # <= A Rectangle a Shape leszármazottja.
        def __init__(self, a, b):
            self.a = a
            self.b = b

        def area(self):
            return self.a * self.b

        def perimeter(self):
            return (self.a + self.b) * 2

    class Circle(Shape): # <= A Circle is a Shape leszármazottja.
        def __init__(self, r):
            self.r = r

        def area(self):
            return self.r**2 * math.pi

        def perimeter(self):
            return 2 * self.r * math.pi

    r1 = Rectangle(10, 20)
    r2 = Rectangle(15, 25)
    c1 = Circle(5)
    c2 = Circle(10)

    print(r1.pa_ratio())
    print(c2.pa_ratio())
```

```
0.3
0.19999999999999998
```

In [369]:
```
# Az alakzat ősosztály esetén a területszámítás nem értelmezhető.
s1 = Shape()
s1.area()
```

```
---------------------------------------------------------------------------
```

```
NotImplementedError                       Traceback (most recent call last)

<ipython-input-30-108142201253> in <module>
  1 s1 = Shape()
----> 2 s1.area()


<ipython-input-29-76b1d4a3833a> in area(self)
  5 class Shape:
  6     def area(self):
----> 7         raise NotImplementedError()
  8
  9     def perimeter(self):


NotImplementedError:
```

In [370]:
```
# Feladat: Készítsünk másodfokú egyenlet megoldó osztályt!
class QuadraticEquation:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def _calcd(self):
        return self.b**2 - 4 * self.a * self.c

   def nsolutions(self):
        d = self._calcd()
        if d > 0: return 2
        elif d == 0: return 1
        else: return 0

    def solve(self):
        d = self._calcd()
        if d > 0:
            x1 = (-self.b + d**0.5) / (2 * self.a)
            x2 = (-self.b - d**0.5) / (2 * self.a)
            return [x1, x2]
        elif d == 0:
            return [-self.b / (2 * self.a)]
        else:
            return []

eq = QuadraticEquation(1, 3, 2)
print(eq.nsolutions())
print(eq.solve())
```

```
print(QuadraticEquation(1, 2, 1).solve())
```

```
2
[-1.0, -2.0]
[-1.0]
```

In [371]:
```
# Feladat: 'Éhes kutyák'.

class Dog:
    def __init__(self, name, is_hungry=False):
        self.name = name
        self.is_hungry = is_hungry

    def eat(self):
        self.is_hungry = False

dogs = [
    Dog('Borzas', True),
    Dog('Vadász', False),
    Dog('Nokedli', False),
    Dog('Cézár', True),
    Dog('Csibész', True)
]
```

In [372]:
```
# Nézzük meg, hogy kik éhesek!
def who_are_hungry():
    for dog in dogs:
        if dog.is_hungry:
            print('{} éhes.'.format(dog.name))

who_are_hungry()
```

```
Borzas éhes.
Cézár éhes.
Csibész éhes.
```

In [373]:
```
# Etessük meg az összes éhes kutyát!
for dog in dogs:
    if dog.is_hungry:
        dog.eat()
```

In [374]:
```
# Éhezzenek meg a kutyák!
for dog in dogs:
    dog.is_hungry = True
```

```
In [375]: # Etessük meg az összes kutyát!
          for dog in dogs:
              dog.eat()

In [376]: # Újra nézzük meg, hogy kik éhesek!
          who_are_hungry()

In [377]: # Oldjuk meg az 'éhes kutyák' feladatot osztályok használata nélkül!
          dogs = [
              {'name': 'Borzas',  'is_hungry': True},
              {'name': 'Vadász',  'is_hungry': False},
              {'name': 'Nokedli', 'is_hungry': False},
              {'name': 'Cézár',   'is_hungry': True},
              {'name': 'Csibész', 'is_hungry': True}
          ]

          def who_are_hungry():
              for dog in dogs:
                  if dog['is_hungry']:
                      print('{} éhes.'.format(dog['name']))

          who_are_hungry()
          # stb.
```

```
Borzas éhes.
Cézár éhes.
Csibész éhes.
```

20.1. Speciális („dunder”) attribútumok és metódusok

- `__doc__`, `__class__`, `__init__()`, `__hash__()`, `__code__`, ...
- attribútumtárolásra: `__dict__`, `__dir__()`
- kiírásra: `__repr__()`, `__str__()`
- műveletvégzésre: `__add__()`, `__mul__()`, ...
- indexelésre: `__getitem__()`, `__setitem__()`, `__len__()`
- iterálásra: `__iter__()`, `__next__()`
- kontextuskezelésre: `__enter__()`, `__exit__()`
- ...

```
In [378]: # Példa: __repr__ metódussal rendelkező osztály.
          class Student:
              def __init__(self, name, neptun):
                  self.name = name
                  self.neptun = neptun

              def __repr__(self):
                  return 'Student(name={}, neptun={})'.format(self.name, self.neptun)
```

```
s = Student('Gipsz Jakab', 'ABC123')
print(s)
```

```
Student(name=Gipsz Jakab, neptun=ABC123)
```

Gyakorlás: Egyszerű vektor osztály

Készítsünk vektor osztályt, amely támogatja a vektorok közötti elemenkénti alapműveleteket (+, -, *, /), a vektor elemszámának lekérdezését, a haladó indexelést valamint a vektor sztringgé alakítását! Elvárt működés:

```
v1 = Vector([1.0, 2.0, 3.0])
v2 = Vector([4.0, 5.0, 6.0])
print(len(v1), v1[0], v1[:2]) # => 3 1.0 [1.0, 2.0]
print(v1 + v2)                # => Vector([5.0, 7.0, 9.0])
print(v1 * v2)                # => Vector([4.0, 10.0, 18.0]
```

```
In [379]: class Vector:
              def __init__(self, data):
                  self.data = data

              def __repr__(self): # sztringgé alakítás
                  return 'Vector({})'.format(self.data)

              def __add__(self, other): # összeadás
                  return Vector([x + y for x, y in zip(self.data, other.data)])

              def __sub__(self, other): # kivonás
                  return Vector([x - y for x, y in zip(self.data, other.data)])

              def __mul__(self, other): # szorzás
                  return Vector([x * y for x, y in zip(self.data, other.data)])

              def __truediv__(self, other): # osztás
                  return Vector([x / y for x, y in zip(self.data, other.data)])

              def __len__(self): # hossz lekérdezése
                  return len(self.data)

              def __getitem__(self, idx): # adott indexű elem kiolvasása
                  return self.data[idx]

              def __setitem__(self, idx, val): # adott indexű elem módosítása
                  self.data[idx] = val
```

```
          v1 = Vector([1.0, 2.0, 3.0])
          v2 = Vector([4.0, 5.0, 6.0])

In [380]: # 2 vektor összeadása
          v1 + v2

Out[380]: Vector([5.0, 7.0, 9.0])

In [381]: # ugyanez metódushívásként
          v1.__add__(v2)

Out[381]: Vector([5.0, 7.0, 9.0])

In [382]: # ugyanez függvényhívásként
          Vector.__add__(v1, v2)

Out[382]: Vector([5.0, 7.0, 9.0])

In [383]: # Megjegyzés: Beépített adattípusok esetén is
          # ugyanilyen módszerrel van megvalósítva a + operátor.
          print(1 + 1)
          print(int.__add__(1, 1))
          print('sör' + 'bor')
          print(str.__add__('sör', 'bor'))
```

```
2
2
sörbor
sörbor
```

```
In [384]: # 2 vektor kivonása
          v1 - v2

Out[384]: Vector([-3.0, -3.0, -3.0])

In [385]: # 2 vektor szorzása
          v1 * v2

Out[385]: Vector([4.0, 10.0, 18.0])

In [386]: # 2 vektor osztása
          v1 / v2

Out[386]: Vector([0.25, 0.4, 0.5])

In [387]: # elemszám lekérdezése
          len(v1)

Out[387]: 3
```

```
In [388]: # indexelés
          v1[0]

Out[388]: 1.0

In [389]: # haladó indexelés
          v1[1:]

Out[389]: [2.0, 3.0]

In [390]: # a v2 vektor utolsó koordinátájának átállítása
          v2[-1] = 100
          print(v2)
```

```
Vector([4.0, 5.0, 100])
```

21. NumPy

A NumPy egy alacsony szintű matematikai csomag, numerikus számításokhoz.

- Alapvető adatszerkezete az n-dimenziós tömb.
- C nyelven íródott. A szokásos tömbműveletek hatékonyan vannak benne megvalósítva.
- Tartalmaz lineáris algebrai és véletlenszám generáló almodult.
- Számos magasabb szintű csomag épül rá (pl. scipy, matplotlib, pandas, scikit-learn).

A NumPy külső csomag, a telepítésére többféle lehetőség van, például:

- `pip install numpy --user`
- `sudo apt-get install python3-numpy`
- `conda install numpy`

```
In [391]: # A numpy modul importálása np néven.
          import numpy as np

In [392]: # Verzió lekérdezése.
          np.__version__

Out[392]: '1.16.2'
```

21.1. Tömbök létrehozása

```
In [393]: # 1 dimenziós, egész számokból álló tömb létrehozása.
          a = np.array([2, 3, 4])

In [394]: # A tömb objektum típusa.
          type(a)

Out[394]: numpy.ndarray
```

```
In [395]: # Hány dimenziós a tömb?
          a.ndim

Out[395]: 1

In [396]: # A dimenziók mérete.
          a.shape

Out[396]: (3,)

In [397]: # Az elemek típusának lekérdezése.
          a.dtype

Out[397]: dtype('int64')

In [398]: # 2 dimenziós, lebegőpontos tömb létrehozása.
          b = np.array([[3.0, 4, 5], [6, 7, 8]])
          b

Out[398]: array([[3., 4., 5.],
                 [6., 7., 8.]])

In [399]: # Dimenziók száma, mérete, az elemek típusa.
          print(b.ndim)
          print(b.shape)
          print(b.dtype)
```

```
2
(2, 3)
float64
```

```
In [400]: # Az elemek adattípusának beállítása, 1. példa.
          np.array([2, 3, 4, 5], dtype='uint8')

Out[400]: array([2, 3, 4, 5], dtype=uint8)

In [401]: # Az elemek adattípusának beállítása, 2. példa.
          np.array([2, 3, 4, 5], dtype='float32')

Out[401]: array([2., 3., 4., 5.], dtype=float32)

In [402]: # Tömb betöltése szövegfájlból.
          np.genfromtxt('matrix.txt')

Out[402]: array([[0., 1., 1., 0., 1., 0., 1., 1., 0., 1.],
                 [0., 0., 1., 0., 1., 1., 0., 1., 0., 1.],
                 [0., 0., 1., 0., 0., 0., 1., 1., 0., 0.],
                 [0., 1., 0., 0., 1., 0., 1., 1., 0., 0.],
                 [1., 0., 1., 1., 0., 0., 1., 0., 1., 1.],
```

```
       [1., 0., 1., 0., 0., 1., 1., 0., 1., 0.],
       [1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],
       [0., 0., 0., 0., 0., 1., 0., 1., 0., 1.],
       [1., 1., 0., 1., 0., 1., 1., 1., 0., 0.],
       [1., 0., 1., 0., 1., 0., 0., 1., 0., 1.]])
```

```
In [403]: # Nullákból álló tömb létrehozása, 1. példa.
          np.zeros((2, 5))
```

```
Out[403]: array([[0., 0., 0., 0., 0.],
                 [0., 0., 0., 0., 0.]])
```

```
In [404]: # Nullákból álló tömb létrehozása, 2. példa.
          np.zeros((2, 3), dtype='int16')
```

```
Out[404]: array([[0, 0, 0],
                 [0, 0, 0]], dtype=int16)
```

```
In [405]: # Egyesekből álló tömb létrehozása, 1. példa.
          np.ones(4)
```

```
Out[405]: array([1., 1., 1., 1.])
```

```
In [406]: # Egyesekből álló tömb létrehozása, 2. példa.
          np.ones((2, 2, 2))
```

```
Out[406]: array([[[1., 1.],
                  [1., 1.]],

                 [[1., 1.],
                  [1., 1.]]])
```

```
In [407]: # Egységmátrix létrehozása.
          np.eye(4)
```

```
Out[407]: array([[1., 0., 0., 0.],
                 [0., 1., 0., 0.],
                 [0., 0., 1., 0.],
                 [0., 0., 0., 1.]])
```

```
In [408]: # Értéktartomány létrehozása a lépésköz megadásával.
          np.arange(-1, 1, 0.2)
```

```
Out[408]: array([-1.00000000e+00, -8.00000000e-01, -6.00000000e-01, -4.00000000e-01,
                 -2.00000000e-01, -2.22044605e-16,  2.00000000e-01,  4.00000000e-01,
                  6.00000000e-01,  8.00000000e-01])
```

```
In [409]: # Értéktartomány létrehozása az elemszám megadásával.
          np.linspace(-2, 2, 12)
```

```
Out[409]: array([-2.        , -1.63636364, -1.27272727, -0.90909091, -0.54545455,
                 -0.18181818,  0.18181818,  0.54545455,  0.90909091,  1.27272727,
                  1.63636364,  2.        ])

In [410]: # Vektorok összefűzése.
          a = np.array([2, 3, 4])
          b = np.array([6, 7])
          np.concatenate([a, b])

Out[410]: array([2, 3, 4, 6, 7])

In [411]: # Mátrixok összefűzése vízszintesen.
          a = np.array([[3, 4, 5], [6, 7 ,8]])
          np.hstack([a, a, a])

Out[411]: array([[3, 4, 5, 3, 4, 5, 3, 4, 5],
                 [6, 7, 8, 6, 7, 8, 6, 7, 8]])

In [412]: # Mátrixok összefűzése függőlegesen.
          np.vstack([a, a])

Out[412]: array([[3, 4, 5],
                 [6, 7, 8],
                 [3, 4, 5],
                 [6, 7, 8]])
```

21.2. Elemek és résztömbök

```
In [413]: # Hozzunk létre egy példamátrixot!
          a = np.array([[3, 4, 5], [6, 7 ,8]])
          a

Out[413]: array([[3, 4, 5],
                 [6, 7, 8]])

In [414]: # Elem kiválasztása (az indexelés 0-tól indul).
          a[0, 1] # 0 a sorindex, 1 az oszlopindex

Out[414]: 4

In [415]: # Ugyanez a __getitem__ függvény segítségével.
          np.ndarray.__getitem__(a, (0, 1))

Out[415]: 4

In [416]: # Teljes sor kiválasztása.
          a[1, :]

Out[416]: array([6, 7, 8])
```

```
In [417]: # Így is lehetne:
          a[1]

Out[417]: array([6, 7, 8])

In [418]: # Oszlop kiválasztása.
          a[:, -1]

Out[418]: array([5, 8])

In [419]: # Résztömb kiválasztása.
          a[:, :2]

Out[419]: array([[3, 4],
                 [6, 7]])

In [420]: # Adott indexű oszlopok kiválasztása.
          a[:, [0, 2]]

Out[420]: array([[3, 5],
                 [6, 8]])

In [421]: # Elemek kiválasztása logikai feltétel alapján.
          b = np.array([3, 6, 2, 10, 11, 1])
          b[b > 5]

Out[421]: array([ 6, 10, 11])

In [422]: # A tömb elemei módosíthatók.
          a[0, 0] = 100
          a

Out[422]: array([[100,   4,   5],
                 [  6,   7,   8]])

In [423]: # Oszlop módosítása.
          a[:, 1] = [40, 70]
          a

Out[423]: array([[ 3, 40,  5],
                 [ 6, 70,  8]])
```

21.3. Tömbműveletek

```
In [424]: # Hozzunk létre 2 példatömböt!
          a = np.array([[2, 3, 4], [5, 6, 7]])
          b = np.array([[1, 1, 1], [2, 2, 2]])
          print(a)
          print(b)
```

```
[[2 3 4]
 [5 6 7]]
[[1 1 1]
 [2 2 2]]
```

```
In [425]: # Elemenkénti összeadás.
          a + b

Out[425]: array([[3, 4, 5],
                 [7, 8, 9]])

In [426]: # Elemenkénti kivonás.
          a - b

Out[426]: array([[1, 2, 3],
                 [3, 4, 5]])

In [427]: # Elemenkénti szorzás.
          a * b

Out[427]: array([[ 2,  3,  4],
                 [10, 12, 14]])

In [428]: # Elemenkénti osztás.
          a / b

Out[428]: array([[2. , 3. , 4. ],
                 [2.5, 3. , 3.5]])

In [429]: # Elemenkénti egészosztás.
          a // b

Out[429]: array([[2, 3, 4],
                 [2, 3, 3]])

In [430]: # Elemenkénti hatványozás.
          a**b

Out[430]: array([[ 2,  3,  4],
                 [25, 36, 49]])

In [431]: # A művelet nem feltétlenül végezhető el.
          c = np.array([2, 3, 4]) # 3 hosszú tömb
          d = np.array([10, 20])  # 2 hosszú tömb
          c + d
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-40-52107b4937c9> in <module>
  2 c = np.array([2, 3, 4]) # 3 hosszú tömb
  3 d = np.array([10, 20])  # 2 hosszú tömb
----> 4 c + d


ValueError: operands could not be broadcast together with shapes (3,) (2,)
```

```
In [432]: # Jelenítsük meg újra az "a" tömböt!
          a

Out[432]: array([[2, 3, 4],
                 [5, 6, 7]])

In [433]: # Elemenkénti függvények (exp, log, sin, cos, ...).

          np.exp(a) # exponenciális függvény

Out[433]: array([[   7.3890561 ,   20.08553692,   54.59815003],
                 [ 148.4131591 ,  403.42879349, 1096.63315843]])

In [434]: np.cos(a) # koszinusz

Out[434]: array([[-0.41614684, -0.9899925 , -0.65364362],
                 [ 0.28366219,  0.96017029,  0.75390225]])

In [435]: # Statisztikai műveletek (min, max, sum, mean, std).

          a.min() # minimum

Out[435]: 2

In [436]: a.max() # maximum

Out[436]: 7

In [437]: a.sum() # összeg

Out[437]: 27

In [438]: a.mean() # átlag

Out[438]: 4.5
```

```
In [439]: a.std() # szórás

Out[439]: 1.707825127659933

In [440]: # Oszloponkénti statisztikák.
          # A 0. dimenzió azaz a sorok mentén aggregálunk,
          # ezért ez a dimenzió fog 'eltűnni', és az 1. dimenzió marad meg.
          a.sum(0)

Out[440]: array([ 7,  9, 11])

In [441]: # Soronkénti statisztikák.
          # A 1. dimenzió azaz az oszlopok mentén aggregálunk,
          # ezért ez a dimenzió fog eltűnni, és a 0. dimenzió marad meg.
          a.mean(1)

Out[441]: array([3., 6.])

In [442]: # Feladat: Hozzunk létre egy 3×3-as,
          # csupa True logikai értéket tartalmazó, NumPy tömböt!

          # 1. megoldás:
          np.array([[True] * 3] * 3)

Out[442]: array([[ True,  True,  True],
                 [ True,  True,  True],
                 [ True,  True,  True]])

In [443]: # 2. megoldás:
          np.ones((3, 3), dtype='bool')

Out[443]: array([[ True,  True,  True],
                 [ True,  True,  True],
                 [ True,  True,  True]])

In [444]: # Feladat: Írjuk ki az átlag feletti elemeket az alábbi NumPy tömbben!
          a = np.array([2, 10, 3, 5, 9, 7])

In [445]: # Megoldás:
          a[a > a.mean()]

Out[445]: array([10,  9,  7])

In [446]: # Feladat: Írjuk ki a páros elemeket az alábbi 2 dimenziós NumPy tömbben!
          b = np.array([[2, 3, 4], [5, 6, 7]])

In [447]: # Megoldás:
          b[b % 2 == 0]

Out[447]: array([2, 4, 6])
```

```
In [448]: # Típuskonverzió.
          a.astype('float32')

Out[448]: array([ 2., 10.,  3.,  5.,  9.,  7.], dtype=float32)

In [449]: # Transzponálás.
          b.T

Out[449]: array([[2, 5],
                 [3, 6],
                 [4, 7]])

In [450]: # A transzponálás nem végez másolást,
          # csak egy új nézetet hoz létre az eredeti adattartalomra.
          b.T[0, 1] = 100
          b

Out[450]: array([[  2,   3,   4],
                 [100,   6,   7]])

In [451]: # Hozzunk létre egy 12 elemű példatömböt!
          c = np.arange(12)
          c

Out[451]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

In [452]: # Átalakítás 2×6-ossá.
          c.reshape((2, 6))

Out[452]: array([[ 0,  1,  2,  3,  4,  5],
                 [ 6,  7,  8,  9, 10, 11]])

In [453]: # Elég csak az egyik dimenzió méretét megadni, a másik helyére írhatunk (-1)-et.
          c.reshape((2, -1))

Out[453]: array([[ 0,  1,  2,  3,  4,  5],
                 [ 6,  7,  8,  9, 10, 11]])

In [454]: # Átalakítás 4×3-assá.
          c.reshape((-1, 3))

Out[454]: array([[ 0,  1,  2],
                 [ 3,  4,  5],
                 [ 6,  7,  8],
                 [ 9, 10, 11]])

In [455]: # Átalakítás 2×2×3-assá.
          c.reshape((2, 2, -1))
```

```
Out[455]: array([[[ 0,  1,  2],
                  [ 3,  4,  5]],

                 [[ 6,  7,  8],
                  [ 9, 10, 11]]])
```

```
In [456]: # Ha olyan méretet adunk meg, ahol az összelemszám nem jöhet ki 12-re,
          # akkor hibaüzenetet kapunk.
          c.reshape((7, -1))
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-67-22fc76211542> in <module>
          1 # Ha olyan méretet adunk meg, ahol az össz-elemszám nem lehet 12,
          2 # akkor hibaüzenetet kapunk.
        ----> 3 c.reshape((7, -1))


        ValueError: cannot reshape array of size 12 into shape (7,newaxis)
```

```
In [457]: # Az értékadás NumPy esetén sem végez másolást.
          a = np.array([2, 3, 4])
          b = a
          b[0] = 200
          print(a)
```

```
[200   3   4]
```

```
In [458]: # Másolni a copy metódussal lehet.
          a = np.array([2, 3, 4])
          b = a.copy()
          b[0] = 200
          print(a)
```

```
[2 3 4]
```

```
In [459]: # Keresés.
          # Példa: Mely indexeknél találhatók az 5-nél kisebb elemek?
          a = np.array([10, 4, 2, 11, 6])
          np.where(a < 5)[0]
```

```
Out[459]: array([1, 2])
```

```
In [460]: # Rendezés helyben.
          a.sort()
          a

Out[460]: array([ 2,  4,  6, 10, 11])

In [461]: # Rendezés új tömbbe.
          a = np.array([5, 8, 10, 3, 9])
          b = np.sort(a)
          print(a)
          print(b)
```

```
[ 5  8 10  3  9]
[ 3  5  8  9 10]
```

```
In [462]: # A rendezett tömb elemei mely indexeknél találhatók az eredeti tömbben?
          a = np.array([5, 8, 10, 3, 9])
          idxs = a.argsort()
          idxs

Out[462]: array([3, 0, 1, 4, 2])

In [463]: # A tömb rendezése az indextömb segítségével.
          a[idxs]

Out[463]: array([ 3,  5,  8,  9, 10])

In [464]: # Van min, max, argmin és argmax függvény is.
          print(a)
          print(a.min())
          print(a.max())
          print(a.argmin())
          print(a.argmax())
```

```
[ 5  8 10  3  9]
3
10
3
2
```

```
In [465]: # A rendező műveleteket soronként ill. oszloponként is használhatjuk.
          a = np.array([[3, 8, 6], [10, 1, 5]])
          print(a)

          print(a.max(1)) # soronkénti maximum

          print(a.min(0)) # oszloponkénti minimum

          print(a.argmin(0)) # oszloponkénti argmin
```

```
[[ 3  8  6]
 [10  1  5]]
[ 8 10]
[3 1 5]
[0 1 1]
```

```
In [466]: # Két vektor skaláris szorzata.
          np.array([3, 4, 5]) @ np.array([2, 2, 2])
```

```
Out[466]: 24
```

```
In [467]: # Mátrixszorzás.
          A = np.array([[2, 3, 4], [5, 6, 7]])
          print(A @ A.T)
          print(A.T @ A)
```

```
[[ 29  56]
 [ 56 110]]
[[29 36 43]
 [36 45 54]
 [43 54 65]]
```

21.4. Broadcastolás

- A broadcastolás a különböző alakú operandusok kezelésére szolgál.
- Példa:

```
A (4d tömb):        8 × 1 × 6 × 5
B (3d tömb):            7 × 1 × 5
Eredmény (4d tömb): 8 × 7 × 6 × 5
```

```
In [468]: # Vektor szorzása skalárral.
          np.array([2, 3, 4]) * 10
```

```
Out[468]: array([20, 30, 40])
```

```
In [469]: # Példa nem broadcastolható tömbökre.
          np.array([4, 5, 6]) + np.array([8, 9])
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-12-41f1c3bea554> in <module>
  1 # Példa nem broadcastolható tömbökre.
```

```
    ----> 2 np.array([4, 5, 6]) + np.array([8, 9])


        ValueError: operands could not be broadcast together with shapes (3,) (2,)
```

```
In [470]: # Mátrix szorzása vektorral.
          A = np.array([[2, 3, 4], [5, 6, 7]])
          b = np.array([1, 2, 3])
          print(A)
          print(b)
          print(A.shape)
          print(b.shape)

          A * b # Oszloponkénti szorzást végez.
```

```
[[2 3 4]
 [5 6 7]]
[1 2 3]
(2, 3)
(3,)
```

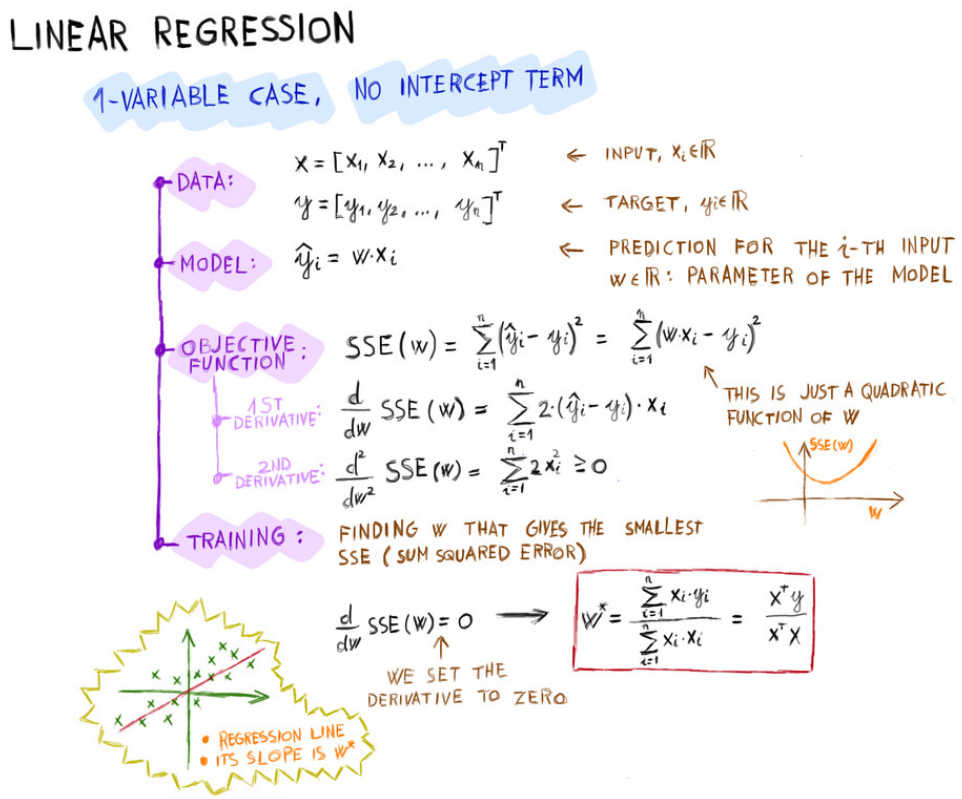
```
Out[470]: array([[ 2,  6, 12],
                 [ 5, 12, 21]])

In [471]: # Soronkénti szorzás.
          (A.T * np.array([10, 20])).T

Out[471]: array([[ 20,  30,  40],
                 [100, 120, 140]])
```

Gyakorlás: Egyváltozós lineáris regresszió

A baseball.txt szövegfájl professzionális amerikai baseball játékosok testmagasságáról és testsúlyáról tartalmaz adatokat. Készítsünk programot, amely lineáris modellt ad a testsúly testmagasságból történő előrejelzésére! Részfeladatok:

- Határozzuk meg a minimális RMSE (root mean squared error) hibát adó modellparamétert!
- Írjuk ki a modell RMSE illetve MAE (mean absolute error) hibáját a tanító adathalmazon!

```
In [472]: # Adatok betöltése.
          data = np.genfromtxt('baseball.txt', delimiter=',')
          x = data[:, 0]
          y = data[:, 1]
```




```
        # Átlag kivonása.
        xmean = x.mean()
        ymean = y.mean()
        x -= xmean
        y -= ymean
```

In [473]:
```
        # Optimális modellparaméter meghatározása.
        w = (x @ y) / (x @ x)
        print(w)
```

```
0.8561919786085516
```

In [474]:
```
        # RMSE hiba kiszámítása.
        yhat = x * w
        rmse = ((yhat - y)**2).mean()**0.5
        print(rmse)
```

```
8.071205900903676
```

In [475]:
```
        # MAE hiba kiszámítása.
        mae = (np.abs(yhat - y)).mean()
        print(mae)
```

```
6.398400208620017
```

```
In [476]: # Milyen testsúlyt becsül a modell, ha a testmagasság 182cm?
          (182 - xmean) * w + ymean
```

```
Out[476]: 87.16109411817018
```

22. pandas

A pandas egy NumPy-ra épülő adatfeldolgozó és elemző eszköz. Alapötleteit az R nyelvből vette. A pandas alapvető adattípusa a DataFrame (tábla) és a Series (oszlop). Segítségükkel memóriabeli, oszlopalapú adatbázis kezelés valósítható meg.

```
In [477]: # A pandas modul importálása pd néven.
          import pandas as pd

In [478]: # A pandas verziószáma.
          pd.__version__

Out[478]: '0.24.1'

In [479]: # DataFrame létrehozása oszlopokból.
          # A bemenet egy szótár, ahol a kulcsok az oszlopnevek, az értékek az oszlopok.
          df1 = pd.DataFrame({
              'aa': [2, 3, 4],
              'bb': [1.5, 10, 20]
          })
          df1

Out[479]:    aa    bb
          0   2   1.5
          1   3  10.0
          2   4  20.0

In [480]: # A df1 objektum típusa.
          type(df1)

Out[480]: pandas.core.frame.DataFrame

In [481]: # Oszlopnevek.
          df1.columns

Out[481]: Index(['aa', 'bb'], dtype='object')

In [482]: # Végigiterálás az oszlopneveken.
          for c in df1:
              print(c)
```

```
aa
bb
```

```
In [483]: # Sorok száma.
          len(df1)
```

```
Out[483]: 3
```

```
In [484]: # A DataFrame alakja.
          df1.shape
```

```
Out[484]: (3, 2)
```

```
In [485]: # Összesítő információ.
          df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
aa    3 non-null int64
bb    3 non-null float64
dtypes: float64(1), int64(1)
memory usage: 128.0 bytes
```

```
In [486]: # Alapvető oszlopstatisztikák.
          df1.describe()
```

```
Out[486]:          aa         bb
          count  3.0    3.00000
          mean   3.0   10.50000
          std    1.0    9.26013
          min    2.0    1.50000
          25%    2.5    5.75000
          50%    3.0   10.00000
          75%    3.5   15.00000
          max    4.0   20.00000
```

```
In [487]: # DataFrame létrehozása sorokból.
          # A bemenet szótárak listája, ahol minden szótár egy sort reprezentál.
          df2 = pd.DataFrame([
              {'cc': 30, 'dd': 20},
              {'cc': 40},
              {'cc': 2.5, 'dd': 10}
          ])
          df2
```

```
Out[487]:      cc    dd
          0  30.0  20.0
          1  40.0   NaN
          2   2.5  10.0

In [488]: # Minden DataFrame-hez (és Series-hez) tartozik index.
          # Alapértelmezés szerint az index 0-tól induló, 1-esével növekedő sorszám.
          df2.index

Out[488]: RangeIndex(start=0, stop=3, step=1)

In [489]: # Ez természetesen felülbírálható, másfajta indexet is használhatunk.
          df3 = pd.DataFrame({
              'aa': [2, 3, 4],
              'bb': [1.5, 10, 20],
              'cc': [42, 1, 1]
          }, index=['alma', 'körte', 'szilva'])
          df3

Out[489]:         aa    bb  cc
          alma     2   1.5  42
          körte    3  10.0   1
          szilva   4  20.0   1

In [490]: # Series létrehozása index megadása nélkül.
          se1 = pd.Series([4, 5, 6, 10])
          se1

Out[490]: 0     4
          1     5
          2     6
          3    10
          dtype: int64

In [491]: # Az se1 objektum típusa.
          type(se1)

Out[491]: pandas.core.series.Series

In [492]: # Series létrehozása index megadásával.
          se2 = pd.Series([4, 5, 6, 10], index=['sör', 'bor', 'pálinka', 'rum'])
          se2

Out[492]: sör         4
          bor         5
          pálinka     6
          rum        10
          dtype: int64

In [493]: # DataFrame-ből [] operátorral lehet kiválasztani oszlopot.
          df3['aa']
```

```
Out[493]: alma      2
          körte     3
          szilva    4
          Name: aa, dtype: int64

In [494]: # ...illetve ha az oszlop neve érvényes azonosítónév, akkor . operátorral is.
          df3.aa

Out[494]: alma      2
          körte     3
          szilva    4
          Name: aa, dtype: int64

In [495]: # Több oszlop kiválasztása.
          df3[['aa', 'bb']]

Out[495]:         aa    bb
          alma     2   1.5
          körte    3  10.0
          szilva   4  20.0

In [496]: # Sor(ok)kiválasztása DataFrame-ből.
          df3.loc['alma']

Out[496]: aa     2.0
          bb     1.5
          cc    42.0
          Name: alma, dtype: float64

In [497]: # ...pozíció alapján is lehet sort kiválasztani
          df3.iloc[1]

Out[497]: aa     3.0
          bb    10.0
          cc     1.0
          Name: körte, dtype: float64

In [498]: # Elem kiválasztása Series-ből.
          print(df3['aa'])
          df3['aa']['körte']
```

```
alma      2
körte     3
szilva    4
Name: aa, dtype: int64
```

```
Out[498]: 3

In [499]: # Nyers adattartalom elérése.
          df3['aa'].values

Out[499]: array([2, 3, 4])
```

22.1. Egyszerű lekérdezések (a `pl.txt` adathalmazon bemutatva)

```
In [500]: # Töltsük be a pl.txt fájl adatait DataFrame-be!
          names = ['round', 'hteam', 'ateam', 'hgoals', 'agoals']
          df = pd.read_csv('pl.txt', sep='\t', skiprows=6, names=names)
          df.head(12) # az első 12 sor megjelenítése
```

Out[500]:

	round	hteam	ateam	hgoals	agoals
0	1	Blackburn Rovers	Wolverhampton Wanderers	1	2
1	1	Fulham FC	Aston Villa	0	0
2	1	Liverpool FC	Sunderland AFC	1	1
3	1	Queens Park Rangers	Bolton Wanderers	0	4
4	1	Wigan Athletic	Norwich City	1	1
5	1	Newcastle United	Arsenal FC	0	0
6	1	Stoke City	Chelsea FC	0	0
7	1	West Bromwich Albion	Manchester United	1	2
8	1	Manchester City	Swansea City	4	0
9	1	Tottenham Hotspur	Everton FC	2	0
10	2	Sunderland AFC	Newcastle United	0	1
11	2	Arsenal FC	Liverpool FC	0	2

```
In [501]: # új oszlop felvétele
          df['goals'] = df['hgoals'] + df['agoals']

In [502]: # A mérkőzések hány százalékán esett gól?
          (df['goals'] > 0).mean() * 100

Out[502]: 92.89473684210526

In [503]: # Melyik mérkőzésen esett a legtöbb gól?
          df.loc[df['goals'].idxmax()]

Out[503]: round                    3
          hteam     Manchester United
          ateam             Arsenal FC
          hgoals                    8
          agoals                    2
          goals                    10
          Name: 29, dtype: object

In [504]: # Írjuk ki, hogy a 10., 20. és 30. fordulóban hány gól esett összesen!
          gb = df.groupby('round')
          gb['goals'].sum()[[10, 20, 30]]

Out[504]: round
          10    39
          20    29
          30    25
          Name: goals, dtype: int64
```

In [505]:
```
# Hány gólt rúgott összesen a Manchester United?

# 1. megoldás: groupby alkalmazásával
g1 = df.groupby('hteam')['hgoals'].sum()
g2 = df.groupby('ateam')['agoals'].sum()
print((g1 + g2)['Manchester United'])
```

```
89
```

In [506]:
```
# 2. megoldás: résztábla kiválasztásával
x1 = df[df['hteam'] == 'Manchester United']['hgoals'].sum()
x2 = df[df['ateam'] == 'Manchester United']['agoals'].sum()
print(x1 + x2)
```

```
89
```

In [507]:
```
# Kitől kapta a legtöbb gólt a Chelsea FC?
df1 = df[df['hteam'] == 'Chelsea FC']
df2 = df[df['ateam'] == 'Chelsea FC']
df3 = pd.DataFrame({
    'team': list(df1['ateam']) + list(df2['hteam']),
    'goals': list(df1['agoals']) + list(df2['hgoals']),
})
df3.sort_values('goals').iloc[-1]
```

Out[507]:
```
team     Arsenal FC
goals             5
Name: 4, dtype: object
```