Bevezetés a Python programozásba - jegyzet

May 29, 2024

1 Adattípusok és alapműveletek, Változók megadása

Ismerkedés a jupyter notebook felülettel. Első python programunk. Adattípusok és alapműveletek: egész, valós, (komplex), sztingek, logikai, semmi (None); és műveleteik. Változók megadása, használata.

1.1 Bevezetés

1.1.1 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 (kódolás).

1.1.2 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, nyílt forráskódú + platformfüggetlen + értelmezett nyelv, típusai dinamikusak + többparadigmás nyelv − bizonyos feladatokhoz lassú lehet − többszálú lehetőségei korlátozottak

1.1.3 Történelem

- **1994**: A Python 1.0 megjelenése.
- **2000**: A Python 2.0 megjelenése.
- **2001**: A Python Software Foundation megalakulása.
- **2003**: Az első PyCon konferencia.
- **2008**: A Python 3.0 megjelenése. Nem volt kompatibilis a 2-es verzióval. Az áttérés lassan ment, de végül megtörtént.
- **2018**: Guido van Rossum lemond a BDFL címről. Egy ötfős bizottság lesz a legfőbb döntéshozó szerv a nyelvvel kapcsolatban (lásd: PEP 8016).

1.2 A Jupyter Notebook környezet (Colab)

- 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. (`a` ugyan ez elé)
    - `m (ctrl+MM)` : Az aktuális cella típusának átállítása szövegesre.
    - `dd (ctrl+MD)`: Az aktuális cella törlése.
    - `Enter`: Átlépés szerkesztőmódba (az aktuális cella tartalmának szerkesztése).
  - Szerkesztőmódban tudjuk szerkeszteni a cellák tartalmát. Néhány billentyűparancs:
    - `Shift+Enter`: Az aktuális cella futtatása.
    - `Esc`: Visszalépés parancsmódba.
- A billentyűparancsokról a Help / Keyboard Shortcuts menü ad részletesebb leírást. (Eszközök / Használható parancsok vagy `ctrl +MH)`)

1.3 Technikai részletek

1.3.1 Implementációk

- CPython (http://python.org/)
- PyPy (http://pypy.org/)
- IronPython (http://ironpython.net/)
- Jython (http://www.jython.org/)
- MicroPython (https://micropython.org/)

1.3.2 Telepítés

Windows

A legcélszerűbb egy Python disztribúciót telepíteni: - Anaconda (https://www.anaconda.com/products/distribution) - Miniconda (http://conda.pydata.org/miniconda.html)

Linux

Több életképes alternatíva is van: - A rendszer csomagkezelőjének használata. - Az értelmező telepítése csomagkezelővel (vagy akár forráskódból), a külső csomagok telepítése pip-pel. - Python disztibúció használata.

1.3.3 Fejlesztőkörnyezetek

nehézsúlyú - PyCharm (http://www.jetbrains.com/pycharm/) - Visual Studio Code (https://code.visualstudio.com/) - PyScripter (https://sourceforge.net/projects/pyscripter/) - Spyder (https://www.spyder-ide.org/) ...

könnyűsúlyú - Emacs / Vim / Geany/ … - IDLE (az alap Python csomag része) - Jupyter Notebook …

1.4 Egyszerű adattípusok

1.4.1 Egész szám (int)

- A számok között a szokásos módon végezhetünk műveleteket. (`+ - * /`)
- A python dokumentációban a számokra vonatkozó rész
- Eredmény kiíratása `print()` függvény segítségével

[1]:
```
# Próbálja ki a 4 alapműveletet!
print(2 + 4)
print(2 - 4)
print(2 * 5)
print(2 / 5)
```

```
6
-2
10
0.4
```

Megjegyzések: - A szóközök nem számítanak, a fenti írásmód a PEP 8 kódolási stílust követi. - **A sorkezdő behúzásnak viszont jelentése van a Pythonban!** - A Jupyter a futtatás után megjeleníti a cella utolsó kifejezését. - Vesszővel elválasztva az egész sor egy gyűjtemény => Mindet kiírja

Ügyeljünk a precedenciára! Azaz a műveletek szokásos sorrendjére!

A sorrend zárójelezéssel `()` felülbírálható.

[2]:
```
# Ha szeretnénk felülbírálni a precedenciát (műveletek szokásos sorrendjét),␣
↪akkor használjunk zárójelezést!
1/(3+4*2)
```

[2]:
```
0.09090909090909091
```

Egész értékű változó

[3]:
```
# Hozzunk létre egy i nevű változót, és tegyük bele a 11 értéket!
i = 11
```

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

[4]:
```
# irassuk ki a képernyőre i értékét -- print()
print(i)
```

```
11
```

[5]:
```
# A változóra a továbbiakban is lehet hivatkozni.
print(i + 3)
print(3 * i)
```

```
14
33
```

[6]:
```
# A változó értéke természetesen változtatható.
i = 12
```

[7]:
```
# Az értékadást lehet kombinálni a többi művelettel.
# += *= -= használata
i = 10
print(i)
i *= 2     # i = i * 2
print(i)
```

```
10
20
```

[8]:
```
i = 10
i += 3    # i = i + 3
print(i)
```

```
13
```

Megjegyzések a műveletekről Sok hibalehetőséget megelőz, hogy külön műveleti jele van a lebegőpontos (`/`) és az egészosztásnak (`//`).

[9]:
```
# Lebegőpontos osztás.
3/7
```

[9]: `0.42857142857142855`

[10]:
```
# Egészosztás (levágja a törtrészt).
3//7
```

[10]: `0`

További hasznos műveletek

[11]:
```
# Maradékképzés.
3 % 7
```

[11]: `3`

[12]:
```
# Van hatványozás is, ** a műveleti jele.
2**10
```

[12]: 1024

[13]:
```
# Az abszolútértéket az abs() függvény kiszámítja
abs(-4)
```

[13]: 4

Egész számok tárolódása a memóriában **Feladat:** Mekkora a 8 bájton = 64 biten ábrázolható legnagyobb egész szám?

Pythonban egyébként nincs ilyen határ. A Python képes tetszőleges hosszúságú egész számokkal dolgozni. Mutassuk is ezt meg egy példán.

[14]:
```
# 8 bájton = 64 biten ábrázolható legnagyobb egész szám
max_int = 2**64 - 1
print(max_int)
```

```
18446744073709551615
```

[15]:
```
# nincsen túlcsordulási hiba
max_int+1
```

[15]: 18446744073709551616

[16]:
```
# A Python képes tetszőleges hosszúságú egész számokkal dolgozni
# nincsen túlcsordulási hiba.
11111111111111111111111111111111111111111111111 + 1
```

[16]: 111111111111111111111111111111111111111111111112

1.4.2 Lebegőpontos szám (float)

- 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.
- A python dokumentációban a számokra vonatkozó rész

[17]:
```
# Lebegőpontos állandókat a tizedespont használatával tudunk megadni.
1.25
```

[17]: 1.25

[18]:
```
# Gyök kettő (közelítő) kiszámítása.
2**0.5 , 2**(1/2)
```

[18]: (1.4142135623730951, 1.4142135623730951)

[19]:
```
# Hozzunk létre egy f nevű, lebegőpontos típusú változót!
f = 1.25
f
```

[19]:
```
1.25
```

[20]:
```
# A type() függvénnyel tudjuk lekérdezni f típusát.
type(f)
```

[20]:
```
float
```

[21]:
```
# ...vagy bármely más érték típusát.
type(i)
```

[21]:
```
int
```

[22]:
```
# Tegyünk most f-be egy int típusú értéket!
# Pythonban ez minden probléma nélkül megtehető.
f = 100
```

[23]:
```
type(f)
```

[23]:
```
int
```

1.4.3 Komplex szám (complex)

- A Python támogatja a komplex számokkal való számolást, külső könyvtárak használata nélkül.
- A j-jelöli a képzetes egységet ha szám kerül elé
- A python dokumentációban a számokra vonatkozó rész

[24]:
```
# Osztás algebrai alakban.
1/(2 + 3j)
```

[24]:
```
(0.15384615384615385-0.23076923076923078j)
```

[25]:
```
# A képzetes egység hatványozása.
1j**2
```

[25]:
```
(-1+0j)
```

1.4.4 Logikai érték (bool)

- A logikai igaz értéket a `True`, a hamisat a `False` jelöli.
- A nagy kezdőbetű fontos, a Python különbözőnek tekinti a kis- és nagybetűket.
- A python dokumentációban a logikai értékre vonatkozó rész

[26]:
```
# Hozzunk létre logikai típusú változót!
# Nézzük meg a típusát is!
b = False
type(b)
```

[26]:
```
bool
```

Logikai műveletek: ÉS (`and`), VAGY (`or`), TAGADÁS (`not` / `!`)

[27]:
```
# Logikai ÉS művelet. -- and
print(True and True)
print(True and False)
print(False and False)
```

```
True
False
False
```

[28]:
```
# Logikai VAGY művelet. -- or
print(True or False)
print(True or True)
print(False or False)
```

```
True
True
False
```

[29]:
```
# Logikai tagadás. -- not
not True
```

[29]:
```
False
```

Az összehasonlító műveletek (`>`,`>=`, `==`) eredménye logikai érték.

[30]:
```
# Nagyobb-e -3 mint 2?
-3 > 2
```

[30]:
```
False
```

[31]:
```
# nagyobb egyenlő-e?
3 >= 3
```

[31]:
```
True
```

[32]:
```
# kisebb egyenlő-e?
3 <= 3
```

[32]:
```
True
```

[33]:
```
# Pythonban az egyenlőségvizsgálat műveleti jele ==.
3 == 3
```

[33]:
```
True
```

[34]:
```
# != (nem egyenlő) operátor.
3 != 3
```

[34]:
```
False
```

Feladat: Definiáljunk egy n egész változót és vizsgáljuk meg, hogy páros 50-nél nagyobb számot adtunk-e meg?

[35]:
```
n = 48
(n > 50) and (n % 2 == 0)
```

[35]:
```
False
```

1.4.5 None / Semmi, üres (NoneType)

- 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.
- None a python dokumentációban

[36]:
```
# A None érték típusa.
type(None)
```

[36]:
```
NoneType
```

[37]:
```
# Ha a cella utolsó kifejezése None értékű, akkor nincs kimenet.
2 + 3
None
```

1.4.6 Sztring (str)

- A sztring adattípus szöveges értékek tárolására szolgál.
- Pythonban a sztring nem más mint Unicode szimbólumok (másnéven Unicode karakterek) nem módosítható sorozata.
- (https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str)

[38]:
```
# A sztringállandót ' jelekkel határoljuk.
s = 'alma és körte'
s
```

[38]:
```
'alma és körte'
```

- Síma idézőjel: `'allows embedded "double" quotes'`
- Dupla idézőjel: `"allows embedded 'single' quotes" "it's"`

- Tripla idézőjel (több soros szöveg): `'''Three single quotes'''`, `"""Three double quotes"""`

A határoló idézőjel ' nem a sztring része, csak az adattípust jelzi

[39]:
```
# ...de lehet használni " jeleket is.
s2 = "ez egy másik szöveg"
s2
```

[39]:
```
'ez egy másik szöveg'
```

[40]:
```
# Írjuk ki a sztring tartalmát, határoló jelek nélkül! print()
s3 = ' ide jön egy idézet "ez egy idézet"'
print(s3)
```

```
 ide jön egy idézet "ez egy idézet"
```

[2]:
```
# több soros sztring tripla idézőjellel
'''ez így egy szöveg
és itt folytatódik'''
```

[2]:
```
'ez így egy szöveg\nés itt folytatódik'
```

[41]:
```
# A type() függvény most is működik.
type(s2)
```

[41]:
```
str
```

[42]:
```
# A sztringben természetesen használhatunk Unicode szimbólumokat.
'I   '
```

[42]:
```
'I   '
```

[3]:
```
# Az újsornak és a tabulátornak is van karaktere: \n és \t
s = "alma és\n körte:\tbarack"
s
```

[3]:
```
'alma és\n körte:\tbarack'
```

[4]:
```
print(s)
```

```
alma és
 körte: barack
```

Milyen hosszú egy sztring? Azaz hány karakterből áll!

- Használjuk a `len()` függvényt.
- A szóköz is karakternek számít!

[43]:
```
s = "alma"
len(s)
```

[43]:
```
4
```

Sztring karaktereinek “kinyerése” (indexelés)

- Az indexelés 0-tól indul!!!
- “[ ]” között: s[0]
- Negatív index is értelmes (végéről).

[44]:
```
# s karaktereinek kinyerése. Az indexelés 0-tól indul!
s[0],s[1]
```

[44]:
```
('a', 'l')
```

[45]:
```
# A kinyert karaktert egy 1 hosszú sztring formájában kapjuk vissza.
len(s) , len(s[0])
```

[45]:
```
(4, 1)
```

[46]:
```
# Túlindexelés esetén hibaüzenetet kapunk.
# próbáljuk ki!
s[8]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_142610/3792544162.py in <module>
      1 # Túlindexelés esetén hibaüzenetet kapunk.
      2 # próbáljuk ki!
----> 3 s[8]

IndexError: string index out of range
```

[47]:
```
# negatív index is értelmes
s[-1]
```

[47]:
```
'a'
```

[48]:
```
# A sztring karaktereit nem lehet módosítani!
s[1]="b"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_142610/555297530.py in <module>
      1 # A sztring karaktereit nem lehet módosítani!
----> 2 s[1]="b"
```

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

[49]:
```
# Természetesen s-nek adhatunk új értéket.
s = "körte   "
```

[51]:
```
# Írjuk ki s tartalmát!
print(s)
```

```
körte
```

Néhány hasznos sztring eljárás `strip()`, `lower()`, `upper()`

[52]:
```
# Sztringek összefűzése.
"Helló!: " + s
```

[52]:
```
'Helló!: körte   '
```

[53]:
```
# Sztring ismétlése a megadott számú alkalommal.
"alma " * 10
```

[53]:
```
'alma alma alma alma alma alma alma alma alma alma '
```

[54]:
```
# Tartalmazásvizsgálat:
"a" in "alma"
```

[54]:
```
True
```

[55]:
```
"k" in "alma"
```

[55]:
```
False
```

[56]:
```
# Üres sztring létrehozása. Nulla karakterből áll.
s0 = ""
len(s0) , type(s0)
```

[56]:
```
(0, str)
```

Sztring kódolása Hogyan kerül az Unicode karaktersorozat egy fájlba?

[63]:
```
# Sztringből a kódolás műveletével képezhetünk bájtsorozatot.
s = "körte   "
s_cod = s.encode("utf-8")
print(s)
print(s_cod)
```

```
körte
b'k\xc3\xb6rte \xe2\x99\xa5 \xe2\x99\xac'
```

[64]:
```
# Az eredmény típusa?
type(s_cod)
```

[64]:
```
bytes
```

[65]:
```
# A bájtok száma nagyobb lehet, mint a Unicode szimbólumok száma!
len(s_cod), len(s)
```

[65]:
```
(14, 9)
```

[66]:
```
# Bájtsorozatból a dekódolás műveletével képezhetünk sztringet.
s_cod.decode("utf-8")
```

[66]:
```
'körte   '
```

Feladat:

- Hány bájton tárolódnak a magyar ábécé ékezetes kisbetűi UTF-8 kódolás esetén?
- Hány bájton tárolódik a és a szimbólum?

[67]:
```
len("é".encode("utf-8"))
```

[67]:
```
2
```

2 Gyűjtemények; Konverziós lehetőségek; Adat beolvasás és kiíratás

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

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

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

2.1 Gyűjtemények (Kollekciók)

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

2.1.1 Néhány álltalános megfontolás

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

2.1.2 Tuple

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

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

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

[2]:
```
# Ellenőrizzük t típusát!
type(t)
```

[2]:
```
tuple
```

[3]:
```
# Az elemek számát a len függvénnyel kérdezhetjük le.
len(t)
```

[3]:
```
5
```

[4]:
```
# Tuple elemeinek elérése (az indexelés 0-tól indul).
t[1]
```

[4]:
```
2
```

[5]:
```
# Az elemeken nem lehet módosítani!
t[1] = 16
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_12800/1175813823.py in <module>
      1 # Az elemeken nem lehet módosítani!
----> 2 t[1] = 16

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

[6]:
```
# A t változó persze kaphat új értéket.
t = (1, 2, 3, (12, 24))
```

[8]:
```
# Tartalmazásvizsgálat.
print(12 in t)
print(1 in t)
```

```
False
True
```

[9]:
```
# Amennyiben nem okoz kétértelműséget, a ( és ) határoló elhagyható!
t2 = 1, 2, "alma"
print(t2)
```

```
(1, 2, 'alma')
```

[10]:
```
# Üres tuple létrehozása.
t0 = ()
len(t0), type(t0)
```

[10]:
```
(0, tuple)
```

[67]:
```
# Egy elemű tuple létrehozása. A zárójel el is hagyható.
t1 = ("s",)
type(t1) , len(t1)
```

[67]:
```
(tuple, 1)
```

2.1.3 Lista (list)

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

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

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

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

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

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

[24]:
```
1.23
```

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

```
l
```

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

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

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

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

[27]: True

[28]:
```
# Üres lista létrehozása.
l0 = []
type(l0), len(l0)
```

[28]: (list, 0)

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

[29]:
```
# Elem hozzáfűzése a lista végére: append()
l.append(134)
l
```

[29]: [1, 16, True, 'alma', 134]

[30]:
```
# Elem beszúrása a lista közepére: insert()
l.insert(1,"körte")
l
```

[30]: [1, 'körte', 16, True, 'alma', 134]

[31]:
```
# Elem indexének meghatározása (az első előfordulásé)   .index()
l.index("alma")
```

[31]: 4

[32]:
```
# Adott indexű elem törlése .pop()
l.pop(5)
l
```

[32]: [1, 'körte', 16, True, 'alma']

[33]:
```
# Utolsó elem törlése.
l.pop()
```

```
l
```

[33]:
```
[1, 'körte', 16, True]
```

[34]:
```
# Két lista összefűzése egy új listába.
[1,2,3] + ['alma','körte']
```

[34]:
```
[1, 2, 3, 'alma', 'körte']
```

[35]:
```
# Lista többszörözése.
['alma','körte']*3
```

[35]:
```
['alma', 'körte', 'alma', 'körte', 'alma', 'körte']
```

2.1.4 Halmaz (set)

- A halmaz adattípus a matematikai halmazfogalom számítógépes megfelelője:
  - Egy elem csak 1x szerepelhet!
- Halmazt indexelni nem lehet, a tartalmazásvizsgálat O(1) időben fut le.
- Halmaz a python dokumentációban

[40]:
```
# Hozzunk létre egy s nevű halmazváltozót!
# Az elemek típusa nem feltétlenül azonos.

s = {1, 1, 3, "alma", 1.25, 1,5}
s
```

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

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

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

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

[38]:
```
True
```

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

[39]:
```
False
```

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

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

Halmaz módosítása, halmazműveletek:

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

[75]:
```
s = {1, 1, 3, "alma", 1.25, 1, 5}
```

[76]:
```
# Elem hozzáadása a halmazhoz.
s.add(12)
print(s)
```

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

[77]:
```
# Elem eltávolítása.
s.remove(1)
s
```

[77]:
```
{1.25, 12, 3, 5, 'alma'}
```

[78]:
```
# unió
{1,2,3} | {1,5,2}
```

[78]:
```
{1, 2, 3, 5}
```

[79]:
```
# metszet
{1,2,3} & {1,5,2}
```

[79]:
```
{1, 2}
```

[80]:
```
# halmazkivonás
{1,2,3} - {1,5,2}
```

[80]:
```
{3}
```

2.1.5 Szótár (dict)

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

[65]:
```
# Hozzunk létre egy d nevű szótárváltozót!
d = { "alma":     120 ,
     "körte":     12.5 ,
      "barack":   4}
d
```

[65]:
```
{'alma': 120, 'körte': 12.5, 'barack': 4}
```

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

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

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

[51]:
```
120
```

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

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

KeyError: 'meggy'
```

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

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

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

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

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

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

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

[111]:
```
True
```

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

[52]:
```
dict
```

[69]:
```
# szótár kulcsok lekérdezése
d.keys()
```

[69]:
```
dict_keys(['körte', 'barack'])
```

2.2 Konverzió

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

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

[53]:
```
1
```

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

[57]:
```
1.0
```

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

[54]:
```
123
```

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

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

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

[56]:
```
1.234
```

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

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

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

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

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

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

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

[62]:
```
{'alma': 1, 'körte': 2}
```

[83]:
```
# dict => párok listája

d = {'alma': 1, 'körte': 2}
list(d.items())
```

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

2.3 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.

2.3.1 Standard bemenet – Billentyűzet

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

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

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

```
Ez egy szöveg
```

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

```
A megedott egész 12
```

2.3.2 Standard kimenet

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

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

```
200

200
```

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

```
200	200
```

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

2.3.3 Formázott kiírás

[4]:
```
# Formázott kiírás f-sztringgel.
x1 = 2
x2 = 2**0.5

print ("A megedott x1 egész", x1)
print ("A megedott x2 valós:", x2)
print (f"A megedott x2 valós: {x2}")
```

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

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

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

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

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

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

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

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

3 Vezérlési szerkezetek

- Pythonban a vezérlési szerkezetek belsejét behúzással kell jelölni.
- Ezt a behúzást a `Tab` billentyű lenyomásával érhetjük el
- Emiatt garantált, hogy a program megjelenése és logikai jelentése összhangban van.

3.1 for ciklus:

- for ciklus a Python dokumentációban
- Szintaxis:

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

- 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.

3.1.1 1.példa

Írjuk ki az alábbi könyvcímeket egymás alá a képernyőre:

Jane Eyre, Shirley, Agnes Grey, Üvöltő Szelek

[25]:
```
# ha csak az eddigi tudásunkat alkalmazzuk

konyv_lista = ['Jane Eyre', 'Shirley', 'Agnes Grey', 'Üvöltő szelek']

print(konyv_lista[0])
print(konyv_lista[1])
```

```
print(konyv_lista[2])
print(konyv_lista[3])
```

```
Jane Eyre
Shirley
Agnes Grey
Üvöltő szelek
```

[1]:
```
# Na de mi van, ha bővül a lista tovabbi konyvekkel?
konyv_lista = ['Jane Eyre', 'Shirley', 'Agnes Grey', 'Üvöltő szelek']
uj_konyvek = ['Wildfell asszonya', 'Villette', 'Henry Hastings kapitány']
```

[27]:
```
# új könyvek hozzáfűzése az eredeti listához
konyv_lista = konyv_lista + uj_konyvek
print(konyv_lista)
```

```
['Jane Eyre', 'Shirley', 'Agnes Grey', 'Üvöltő szelek', 'Wildfell asszonya',
'Villette', 'Henry Hastings kapitány']
```

[2]:
```
# de a hozzáfűzést is elvégezhetem for ciklussal:
konyv_lista = ['Jane Eyre', 'Shirley', 'Agnes Grey', 'Üvöltő szelek']
uj_konyvek = ['Wildfell asszonya', 'Villette', 'Henry Hastings kapitány']

for konyv in uj_konyvek:
    konyv_lista.append(konyv)

print(konyv_lista) # ez már a cikluson kívüli utasítás!
```

```
['Jane Eyre', 'Shirley', 'Agnes Grey', 'Üvöltő szelek', 'Wildfell asszonya',
'Villette', 'Henry Hastings kapitány']
```

[28]:
```
# írjuk ki a neveket for ciklussal
for konyv in konyv_lista:
    print(konyv)
```

```
Jane Eyre
Shirley
Agnes Grey
Üvöltő szelek
Wildfell asszonya
Villette
Henry Hastings kapitány
```

[29]:
```
# sorszámozzuk a kiírt könyvcímeket
sorszam = 1
for konyv in konyv_lista:
    print(str(sorszam)+". könyv:" , konyv)
#    sorszam = sorszam + 1
```

```
    sorszam += 1
```

```
1. könyv: Jane Eyre
2. könyv: Shirley
3. könyv: Agnes Grey
4. könyv: Üvöltő szelek
5. könyv: Wildfell asszonya
6. könyv: Villette
7. könyv: Henry Hastings kapitány
```

3.1.2 1.példa folytatása

Írjuk ki az alábbi könyvcímeket egymás alá a képernyőre sorszámozva. - a könyvcímek után tüntessük fel az egyes címek hosszát is zárójelben - mekkora a könyvcímek összhosszúsága? - mekkora a könyvcímek átlagos hossza?

[22]:
```
konyv_lista = ['Jane Eyre',
               'Shirley',
               'Agnes Grey',
               'Üvöltő szelek',
               'Wildfell asszonya',
               'Villette',
               'Henry Hastings kapitány'
              ]
```

[44]:
```
sorszam = 1
ossz_hossz = 0
for konyv in konyv_lista:
    hossz = len(konyv)
    print(str(sorszam)+". könyv:" , konyv, f"\t({hossz} karakter)")
    sorszam += 1
    ossz_hossz += hossz
atlag = ossz_hossz/len(konyv_lista)
print()
print("A könyvcímek összhossza: ", ossz_hossz, "karakter")
print()
print("A könyvcímek átlagos hossza: ", f'{atlag:.2f}' )
```

```
1. könyv: Jane Eyre     (9 karakter)
2. könyv: Shirley       (7 karakter)
3. könyv: Agnes Grey    (10 karakter)
4. könyv: Üvöltő szelek         (13 karakter)
5. könyv: Wildfell asszonya     (17 karakter)
6. könyv: Villette      (8 karakter)
7. könyv: Henry Hastings kapitány       (23 karakter)

A könyvcímek összhossza:  87 karakter
```

```
A könyvcímek átlagos hossza:  12.43
```

3.1.3 2.példa

a) Írjuk ki a képernyőre az első 10 négyzetszámot!

b) Gyűjtsük össze egy listába az első 10 négyzetszámot!

[47]:
```
# Értéktartomány (range) létrehozása.
list(range(10))
```

[47]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[49]:
```
# 3 paraméter: kezdő érték (1), minél kisebb (11), mekkora lépésközzel (2)
list(range(1, 10 + 1, 2))
```

[49]:
```
[1, 3, 5, 7, 9]
```

[52]:
```
n = 10
for elem in range(1, n + 1):
    print(elem**2)
```

```
1
4
9
16
25
36
49
64
81
100
```

[53]:
```
n = 10
negyzetszamok = []   # üres lista
for elem in range(1, n + 1):
    negyzetszamok.append(elem**2)
print(negyzetszamok)
```

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

3.1.4 3.példa

a) Írjuk ki a képernyőre n darab * karaktert!

b) Készítsünk n-szer n-es háromszöget * karakterekből!

```
n = 4
*
**
```

```
***
****
```

[3]:
```python
# Példa: n darab * karakter kiírása.
n = 3
# for ciklussal
for elem in range(n):
    print("*",end="")

print()
# beépített lehetőséggel egyszerűbb
print("*"*n)
```

```
***
***
```

[58]:
```python
# Példa: n-szer n-es háromszög * karakterekből.
n = 4
for elem in range(n):
    print("*" * (elem + 1))
```

```
*
**
***
****
```

3.1.5 4.példa:

a) Számoljuk meg a magánhangzókat egy angol kisbetűs szövegben!

b) Készítsünk statisztikát a magánhangzókról! Azaz külön, külön legyen meg minden magánhangzóra a darabszáma.

a) Stratégia: Végigmegyünk a szöveg karakterein és ha találtunk egy magánhangzót azt följegyezzük egy gyűjtő változóba.

[63]:
```python
# a) Magánhangzók megszámolása (angol kisbetűs szövegben).
text = 'This is a short text for testing the algorithm.'
vowels = {'a', 'e', 'i', 'o', 'u'}


v_num = 0
for ch in text:
    if ch in vowels:
        v_num = v_num + 1
print("A magángangzók száma: ",v_num )
```

```
A magángangzók száma:  12
```

b) Stratégia 1.: Végigmegyünk az egyes magánhangzókon külön külön.

Mindegyik magánhangzónál megvizsgáljuk, hogy hányszor szerepel a szövegben.

Az eredményt (magánhangzó –> darabszám) például szótárban eltároljuk.

[2]:
```
# b) Magánhangzók statisztikája szótárral
text = 'This is a short text for testing the algorithm.'
vowels = {'a', 'e', 'i', 'o', 'u'}

# teljesen üres szótár feltöltése
d = dict()
for v in vowels:
    for ch in text:
        if v == ch:
            if v in d:
                d[v] += 1
            else:
                d[v] = 1

print(d)
```

```
{'e': 3, 'o': 3, 'a': 2, 'i': 4}
```

b) Stratégia 2.: Készítünk egy szótárat aminek a magánhangzók a kulcsai és minden értéket kezdetben 0-ra állítunk.

A szövegen végighaladva minden betűnél megvizsgáljuk, hogy szerepel-e a szótárkulcsok között. Ha igen akkor eggyel növelem a megtalált magánhangzónál szereplő értéket.

[1]:
```
# b) Magánhangzók statisztikája szótárral (angol kisbetűs szövegben).
text = 'This is a short text for testing the algorithm.'
vowels = {'a', 'e', 'i', 'o', 'u'}

d = dict()

# üres szótár inicializálása kulcsokal
for ch in vowels:
    d[ch] = 0
print(d)

# adatok feltöltése
for ch in text:
    if ch in d:
        d[ch] += 1

print(d)
```

```
{'e': 0, 'u': 0, 'o': 0, 'a': 0, 'i': 0}
{'e': 3, 'u': 0, 'o': 3, 'a': 2, 'i': 4}
```

3.2 if (ha) utasítás

- if utasítás a Python dokumentációban
- Szintaxis:

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

- 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.

3.2.1 1.Példa: Kérsz sört?

[ ]:
```
kor = 15
if kor >= 18:
    print(kor,"éves vagy. Kaphatsz sört!")
else:
    print(kor,"éves vagy. Nem kaphatsz sört!")
```

```
15 éves vagy. Nem kaphatsz sört!
```

3.2.2 2.példa: Mennyi 2*4? Az eredmény függvényében adjunk visszajelzést: - "Ügyes vagy!" - "Gyakorold még a szorzótáblát!"

[ ]:
```
n = int (input ("Mennyi 2*4? "))

if n == 8:
    print("Ügyes vagy!")
else:
    print("Gyakorold még a szorzótáblát!")
```

```
Gyakorold még a szorzótáblát!
```

3.3 for ciklus if utasításokkal kombinálva

3.3.1 1.példa: Számlista elemeinek vizsgálata

Az alábbi számlistában vizsgáljuk meg az elemeket: - írjuk ki az adott elemet, majd utána - írjuk ki ha negatív - írjuk ki ha egyjegyű - írjuk ki hogy páros, vagy páratlan

[80]:
```
print("-"*42+"\nSzámlista elemeinek vizsgálata:\n"+"-"*42)
számlista=[0, 99, -7, 56, 23, 6, 1, -5, 9]

for szám in számlista:
```

```
    print("\nVizsgált szám:", szám, end=';\t')

    if szám<0:
        print(" negatív", end=',')
    # nem kell else-ág, mert csak a negatívakra akarunk figyelmeztetni

    if 0<=szám<10:
        print(" egyjegyű",end=',')

    # egyenlőség vizsgálata: '==' jel. Olvasása: "Egyenlő-e"
    if (szám % 2)==0:    # szám % 2 = szám-nak a 2-vel vett osztási maradéka
        print(" páros",end=',')
    else:
        print(" páratlan",end=',')
print()
```

```
-----------------------------------------
Számlista elemeinek vizsgálata:
-----------------------------------------

Vizsgált szám: 0;       egyjegyű, páros,
Vizsgált szám: 99;      páratlan,
Vizsgált szám: -7;      negatív, páratlan,
Vizsgált szám: 56;      páros,
Vizsgált szám: 23;      páratlan,
Vizsgált szám: 6;       egyjegyű, páros,
Vizsgált szám: 1;       egyjegyű, páratlan,
Vizsgált szám: -5;      negatív, páratlan,
Vizsgált szám: 9;       egyjegyű, páratlan,
```

3.3.2 2.példa: összetett feltételek

- Az alábbi számlistából válogassa ki a 2-vel és 3-mal is osztható számokat!
- Melyik számok oszthatók 7-tel úgy, hogy a négyzetük 3-mal osztva 1 maradékot ad?

[81]:
```
számlista=[0, 99, -7, 56, 23, 6, 1, -5, 9]
```

[83]:
```
print("\n2-vel és 3-mal is osztható számok listája.")
for szám in számlista:
    if (szám%2==0) and (szám%3==0): # a szám 2-vel és 3-mal is osztható
        print(szám, end=', ')

print("\nMelyik számok oszthatók 7-tel úgy, hogy a négyzetük 3-mal osztva 1␣
 ↪maradékot ad?")
for szám in range(100):  # nem fix lista, hanem 0-tól 99-ig megyünk
    if (szám % 7==0) and (szám**2 % 3 == 1): # beljebb tördelve a for miatt
        print(szám,end=', ')  # még beljebb tördelve az if miatt
```

```
2-vel és 3-mal is osztható számok listája.
0, 6,
Melyik számok oszthatók 7-tel úgy, hogy a négyzetük 3-mal osztva 1 maradékot ad?
7, 14, 28, 35, 49, 56, 70, 77, 91, 98,
```

3.4 while ciklus

- lehet, hogy nem tudjuk előre a lépések számát…
- while utasítása Python dokumentációban
- Szintaxis:

```
while FELTÉTEL:
    UTASÍTÁS
```

- Egy jól megírt program esetén **az utasítás a feltételt előbb-utóbb hamisra állítja**. (Ellenkező esetben, ha a feltétel igaz, akkor az is marad, így végtelen ciklus keletkezik.)

3.4.1 1.példa: Móricka a programozásvizsgán

Móricka addig megy a programozás vizsgára amíg az nem sikerül neki. A ponthatár 16 pont.

[1]:
```
# Példa: Móricka a programozásvizsgán.
while int(input('Hány pontot értél el? ')) < 16:
    print('Tanulj még!')

print('Gratulálok, átmentél.')
```

```
Tanulj még!
Gratulálok, átmentél.
```

3.4.2 2.példa: Testmagasság bekérése amíg valós adatot nem kapunk

- Kérjük be a felhasználótól a centiméterben megadott testmagasságot (h)
- Vegyük figyelembe, hogy a valaha létező
  - legalacsonyabb ember 54,6 cm,
  - a legmagasabb ember pedig 272 cm-es volt!

[2]:
```
h = float ( input("Adja meg a testmagasságát cm-ben: ") )

while (h < 54.6) or (h > 272):
    print("Ez nem lehet!")
    h = float ( input("Adja meg a testmagasságát cm-ben: ") )

print(f"Az ön testmagassága {h:0.1f} cm.")
```

```
Ez nem lehet!
Az ön testmagassága 166.0 cm.
```

3.4.3 3.példa

Ki tudnánk írni az első 10 negyzetszámot while segítségével a képernyőre?

[3]:
```
i = 1
while (i <= 10):
    print(i**2)
    i += 1
```

```
1
4
9
16
25
36
49
64
81
100
```

3.4.4 Hibalehetőségek:

- Végtelen cilus: a feltétel mindig teljesül
- El sem induló ciklus: a feltétel már az első lépésben elbukik

[4]:
```
# Végtelen ciklus ha mindig teljesül a feltétel.
# NE FUTASSA!

# i = 1
# while (i <= 10):
#     print(i**2)
```

[5]:
```
# El sem induló ciklus
i = 11
print(i)
while (i <= 10):
    print(i**2)
    i += 1
print('Vége')
```

```
11
Vége
```

3.5 Ciklus futásának befolyásolása

- `break` - azonnali kiugrás a ciklusból
- `continue` - az adott "ütem" továbbléptetése

3.5.1 1.példa: Móricka a programozásvizsgán (break verzió)

[6]:
```
# break használatával
while True:
    pont = int(input('Hány pontot értél el? '))
    if pont < 16:
        print('Tanulj még!')
    else:
        print('Gratulálok, átmentél.')
        break
```

```
Tanulj még!
Gratulálok, átmentél.
```

3.5.2 2.példa: Testmagasság bekérése (break verzió)

[7]:
```
# break használatával
while True:
    h = float ( input("Adja meg a testmagasságát cm-ben: ") )
    if (h < 54.6) or (h > 272):
        print("Ez nem lehet!")
    else:
        print(f"Az ön testmagassága {h:0.1f} cm.")
        break
```

```
Az ön testmagassága 123.0 cm.
```

3.5.3 3.Példa: Páros négyzetszámok kiíratása 100-ig

[8]:
```
# range testreszabásával:
N = 10
for elem in range(2, N+1 ,2):
    print(elem**2)
```

```
4
16
36
64
100
```

[9]:
```
# if segítségével párosságot ellenőrizve
N = 10
for elem in range(1, N+1):
    if elem%2 == 0:
        print(elem**2)
```

```
4
16
```

```
36
64
100
```

[10]:
```python
# if segítségével páratlanságot ellenőrizve és ütemet átlépve
N = 10
for elem in range(1, N+1):
    if elem%2 != 0:
        continue
    print(elem**2)
```

```
4
16
36
64
100
```

3.6 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!

[12]:
```python
# Az (ál)véletlenszám-generáló modul importálása.
import random

# Véletlen egész szám kisorsolása 1 és 100 között. -- randint() függvény a␣
 ↪random modulból
n = random.randint(1,100)

# Tippek kérése, amíg a játékos el nem találja a számot.

tipp =-1
while n != tipp:
    # ----   tipp bekerese --------
    tipp = int(input("Adja meg a tippet: "))
    #-------------------------------------

    if tipp > n:
        print("Ez túl nagy tipp.")
    elif tipp < n:
        print("Ez túl kicsi tipp.")
    else:
        print("Gratulálunk, eltalálta!")
```

```
Ez túl kicsi tipp.
Ez túl kicsi tipp.
Ez túl kicsi tipp.
```

```
Ez túl kicsi tipp.
Ez túl nagy tipp.
Ez túl nagy tipp.
Ez túl nagy tipp.
Ez túl nagy tipp.
Ez túl nagy tipp.
Ez túl nagy tipp.
Gratulálunk, eltalálta!
```

[ ]:
```
# randint függvény dokumentációs sztringje!
random.randint.__doc__
```

[ ]:
```
'Return random integer in range [a, b], including both end points.\n        '
```

4 Pythonos “ügyességek”

Rendezés; Haladó indexelési technikák; Comprehension; Kicsomagolás, értékcsere és haladó iterációk

4.1 Még néhány hasznos lista eljárás:

rendezés, minimum, maximum, összeg

4.1.1 Lista rendezése helyben

[61]:
```
# Lista rendezése helyben.
l = [10, 2, 11, 3]
l.sort()
print(l) # az eredeti lista megváltozott!
```

```
[2, 3, 10, 11]
```

[62]:
```
# Rendezés csökkenő sorrendbe - reverse paraméter
l = [10, 2, 11, 3]
l.sort(reverse = True)
print(l)
```

```
[11, 10, 3, 2]
```

[63]:
```
# Fontos, hogy az elemek összehasonlíthatók legyenek!
l = [2, 1, 'alma']
l.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_175381/1268567011.py in <module>
      1 # Fontos, hogy az elemek összehasonlíthatók legyenek!
```

```
      2 l = [2, 1, 'alma']
----> 3 l.sort()

TypeError: '<' not supported between instances of 'str' and 'int'
```

[64]:
```
# Ha csak sztringeket tartalmaz a lista, akkor lehet rendezni.
l = ['alma', 'szilva', 'körte']
l.sort()
print(l)
```

```
['alma', 'körte', 'szilva']
```

4.1.2 Gyűjtemény rendezése listába.

[65]:
```
# Kollekció rendezése listába.
l1 = [10, 4, 20, 5]
sorted(l1)
```

[65]: [4, 5, 10, 20]

[66]:
```
# Tuple elemeit is rendezhetjük új listába.
t1 = (10, 4, 20, 5)
sorted(t1)
```

[66]: [4, 5, 10, 20]

[67]:
```
# ...és halmaz elemeit is.
sorted({3,676,11,42})
```

[67]: [3, 11, 42, 676]

[68]:
```
# Szótár esetén a sorted a kulcsokat rendezi.
d1 = {"barack": 10, "alma": 3, "mandula" : 25 }
sorted(d1)
```

[68]: ['alma', 'barack', 'mandula']

[69]:
```
# Párok listájának rendezése (lexikografikusan).
l = [('sör', 10), ('bor', 20), ('pálinka', 30), ('bor', 5)]
sorted(l)
```

[69]: [('bor', 5), ('bor', 20), ('pálinka', 30), ('sör', 10)]

4.1.3 minimum, maximum, összeg

[70]:
```
l1 = [10, 4, 20, 5]
print("maximum", max(l1))
print("minimum", min(l1))
print("összeg", sum(l1))
```

```
maximum 20
minimum 4
összeg 39
```

4.2 Haladó indexelés (szeletelés/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.

[71]:
```
# Példák haladó indexelésre.
data = ['alma', 'banán', 10, 20, 30]
```

[72]:
```
# első 3 elem kiválasztása
print(data [0:3])
print(data [ :3])
```

```
['alma', 'banán', 10]
['alma', 'banán', 10]
```

[73]:
```
# Minden második elem elem kiválasztása:
data [::2]
```

[73]: ['alma', 10, 30]

[74]:
```
# Minden kivéve az utolsó elemet
# Használhatunk negatív indexeket is
data [:-1]
```

[74]: ['alma', 'banán', 10, 20]

[75]:
```
# utolsó 3 elem
data[-3 : ]
```

[75]: [10, 20, 30]

[76]:
```
# lista elemek fordított sorrendben
data[::-1]
```

[76]: [30, 20, 10, 'banán', 'alma']

Szövegre is működik a haladó indexelés!

[77]:
```
text = "abcde"
text[::-1]
```

[77]:
```
'edcba'
```

4.3 Comprehension

- A comprehension gyűjtemények tömör megadását teszi lehetővé.
- Comprehension a Python dokumentációban
- Hasonlít a matematikában alkalmazott, tulajdonság alapján történő halmazmegadásra (példa: a páratlan számok halmaza megadható $\{2k+1 \mid k \in \mathbb{Z}\}$ módon).

4.3.1 Feltétel nélküli comprehension

[78]:
```
# Állítsuk elő az első 10 négyzetszám listáját gyűjtőváltozó használatával!
N = 10
l = []
for elem in range(1, N+1):
    l.append(elem**2)
print(l)
```

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

[79]:
```
# Ugyanez tömörebben, lista comprehension-nel is megoldható
N = 10
l = [ elem**2 for elem in range(1, N + 1) ]
print(l)
```

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

[80]:
```
# Állítsuk elő az első 10 négyzetszám halmazát!
N = 10
h_nsz = { elem**2 for elem in range(1, N + 1) }
print(h_nsz)
```

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

De lehet könnyen szótárat is készíteni.

Példa: Párosítsuk össze a számokat a négyzetükkel szótárat használva

[81]:
```
# szótár készítése comprehensönnel
N = 10
{ elem: elem**2 for elem in range(1, N+1)   }
```

[81]:
```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Feladat: Állítsunk elő egy szótárat, amely az angol kisbetűs magánhangzókhoz hozzárendeli az ASCII-kódjukat!

Segítség: a kód előállításához használjuk az `ord()` függvényt!

[82]:
```
# Használjunk először gyűjtőváltozót!
vowels = "aeiou"
d = {}

for v in vowels:
    d[v] = ord(v)

print(d)
```

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

[83]:
```
# Ugyanez tömörebben, szótár comprehension-nel:
vowels = "aeiou"
{ v: ord(v)     for v in vowels }
```

[83]:
```
{'a': 97, 'e': 101, 'i': 105, 'o': 111, 'u': 117}
```

Feladat: Párok tagjainak megcserélése egy listában.

[84]:
```
# cseréljük meg a párok tagjait!
pairs = [('alma', 10), ('körte', 20), ('barack', 30)]

[ (p[1], p[0] ) for p in pairs]
```

[84]:
```
[(10, 'alma'), (20, 'körte'), (30, 'barack')]
```

4.3.2 Feltételes comprehension

[85]:
```
# Páros négyzetszámok gyűjtőváltozó használatával
N = 10
l = []
for elem in range(1, N + 1):
    if elem % 2 == 0:
        l.append(elem**2)
print(l)
```

```
[4, 16, 36, 64, 100]
```

[86]:
```
# Páros négyzetszámok comprehension:
N = 10
l = [ elem**2 for elem in range(1, N +1) if elem%2 == 0 ]
print(l)
```

```
[4, 16, 36, 64, 100]
```

[87]:
```
# Feltételes szótár comprehension.
# Az angol kisbetűs magánhangzókhoz hozzárendeli az ASCII-kódjukat "u"␣
 ↪kivételével
{ v: ord(v)     for v in vowels   if v != "u" }
```

[87]:
```
{'a': 97, 'e': 101, 'i': 105, 'o': 111}
```

Feladat: Magánhangzók megszámolása (angol kisbetűs szövegben Comprehensionnel

[88]:
```
text = 'This is a short text for testing the algorithm.'
vowels = 'aeiou'

sum(1 for elem in text if elem in vowels)
```

[88]:
```
12
```

[90]:
```
# másik megoldás count() eljárással
sum(text.count(mgh) for mgh in vowels)
```

[90]:
```
12
```

Feladat: Magánhangzók statisztikája (angol kisbetűs szövegben Comprehensionnel

[91]:
```
text = 'This is a short text for testing the algorithm.'
vowels = 'aeiou'

{ mgh: text.count(mgh) for mgh in vowels}
```

[91]:
```
{'a': 2, 'e': 3, 'i': 4, 'o': 3, 'u': 0}
```

4.4 Kicsomagolás (unpacking)

- többszörös értékadás
- értékcsere
- for ciklusban is alkalmazható

[93]:
```
# Általános eset: 'alma' ; 2  és [30,40] listához hozzáférés?
[x,(y,z)] =   ['alma', (2, [30, 40])]
print(x)
print(y)
print(z)
```

```
alma
2
[30, 40]
```

[94]:
```
# Ha a bal és jobb oldal nem illeszthető egymásra, hibát kapunk.
x, y, z = 1, 2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_175381/3325170505.py in <module>
      1 # Ha a bal és jobb oldal nem illeszthető egymásra, hibát kapunk.
----> 2 x, y, z = 1, 2

ValueError: not enough values to unpack (expected 3, got 2)
```

[95]:
```
# Többszörös értékadásra is jó
x, y = 1, 2
print(f"x = {x}  és y = {y}")
```

```
x = 1  és y = 2
```

Fontos alkalmazás: Értéket cserélni is lehet így változók közt!

[97]:
```
x, y = 1, 2       # értékadás
print(f"x = {x}  és y = {y}")  # szép kiíratás

x, y = y, x       # értékcsere
print(f"x = {x}  és y = {y}")
```

```
x = 1  és y = 2
x = 2  és y = 1
```

[ ]:
```
# Kicsomagolás alkalmazható for ciklusnál is
pairs = [('sör', 10), ('bor', 20), ('rum', 30)]

for x,y in pairs:
    print(x)
```

```
sör
bor
rum
```

[ ]:
```
# gyűjtsük listába az előző párok listájából a sztringeket
[ x  for x, y in pairs]
```

```
['sör', 'bor', 'rum']
```

4.5 Haladó iterálási technikák

4.5.1 enumerate

- enumerate a dokumentációban
- Alkalmazás: Sorindex nyilvántartása szövegfájl feldolgozásnál.

[ ]:
```
# Iterálás az elemeken és indexeken egyszerre, hagyományos megoldás.
x = ['alma', 'körte', 'szilva']

for elem in range(len(x)):
    print(elem , x[elem])
```

```
0 alma
1 körte
2 szilva
```

[ ]:
```
# Ugyanez elegánsabban, enumerate-tel:
x = ['alma', 'körte', 'szilva']
for i, xi in enumerate(x):
    print(i,xi)
```

```
0 alma
1 körte
2 szilva
```

[ ]:
```
# Az enumerate eredménye egy iterálható objektum
# átalakítható párok listájává
list(enumerate(x))
```

```
[(0, 'alma'), (1, 'körte'), (2, 'szilva')]
```

[ ]:
```
# Az enumerate kicsomagolás nélkül is használható.
for elem in enumerate(x):
    print(elem[0], elem[1])
```

```
0 alma
1 körte
2 szilva
```

4.5.2 zip

- zip a dokumentációban
- könnyű adatpárokat készíteni

[54]:
```
# 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
```

[60]:
```
# Ugyanez elegánsabban, zip-pel:
x = ['sör', 'bor', 'pálinka']
y = [10, 20, 30]

for elem in zip(x,y):
    print(elem[0], elem[1])
```

```
sör 10
bor 20
pálinka 30
```

[56]:
```
# A zip eredménye egy iterálható objektum.
# Ami átalakítható listává.
list(zip(x,y))
```

[56]:
```
[('sör', 10), ('bor', 20), ('pálinka', 30)]
```

[58]:
```
# Ha szekvenciák hossza nem azonos, akkor az eredmény a rövidebb hosszát veszi␣
 ↪fel.
x = [10, 20]
y = ['a', 'b', 'c']
list(zip(x,y))
```

[58]:
```
[(10, 'a'), (20, 'b')]
```

[59]:
```
# A zip kettőnél több szekvenciára is alkalmazható.
x = ['alma', 'körte', 'barack']
y = [10, 20, 30]
z = ['X', 'Y', 'Z']
list(zip(x,y,z))
```

[59]:
```
[('alma', 10, 'X'), ('körte', 20, 'Y'), ('barack', 30, 'Z')]
```

5 Fájlkezelés és ismerkedés a függvényekkel

5.1 Fájlkezelés

- Fájlkezelés a dokumentációban
- A fájl valamilyen adathordozón tárolt, logikailag összefüggő adatok összessége.
- Egy fájl életciklusa a következő lépésekből áll:
  1. megnyitás
  2. olvasás, írás, pozícionálás, ...
  3. bezárás

5.1.1 Fájl írása `.write()`

A megnyitás lehetséges módjai: - `"r"`: olvasásra (read) - Csak meglévő fált tud megnyitni - `"w"`: írásra (write) - Létrehozza az új üres fájt. (Ha létezett, ha nem) - `"a"`: hozzáfűzésre (append) - A

meglévő fájlt megnyitja és a fájl végére áll.

[18]:
```
#1 Fájl megnyitása (írásra) - létre is hozza a fájlt!
f = open('valami.txt',"w")
#2 Sztring fájlba írása
f.write("1 alma\n")
#3 Fájl bezárása.
f.close()
```

[19]:
```
# with használatával     - Vigyázat!! felül írtam az előzőt
with open('valami.txt', 'w') as f:
    f.write("2 körte\n")
```

[20]:
```
# röviden               - Most hozzáfűzünk az előzőhöz!
open('valami.txt', 'a').write("3 barack\n")
```

[20]: 9

Példa: készítsünk egy "gyumolcs.txt" nevű fájlt, amely az alábbi gyümölcsök neveit tartalmazza sorszámmal ellátva. Az első sorban szerepeljen egy fejléc a megadott sztringgel.

[57]:
```
nevek = ["alma", "körte", "szilva"]
fejlec = "index\tname\n"
```

[58]:
```
with open('gyumolcs.txt', 'w') as f:
    f.write(fejlec)
    for num, gy in enumerate(nevek):
        f.write(f"{num}\t{gy}\n")
```

5.1.2 Fájl olvasása .read()

[59]:
```
# Fájl tartalmának beolvasása sztringbe.
f = open('gyumolcs.txt',"r")
s = f.read()
f.close()
```

[60]:
```
print(s)
```

```
index   name 0  alma
1       körte
2       szilva
```

[25]:
```
# with használatával
with open('gyumolcs.txt',"r") as f:
    s = f.read()

print(s)
```

```
0       alma
1       körte
2       szilva
```

[27]:
```
#rövidebben
s = open('gyumolcs.txt',"r").read()

print(s)
```

```
0       alma
1       körte
2       szilva
```

Példa: Készítsünk egy fájlt, majd olvassuk be az első sorát! - Az `example_file.txt`-t a munkakönyvtárban a New / Text File menüponttal hozhatjuk létre. (Desktop alkalmazás) - A Colab felületén pedig a bal oldali menü könyvtár ikonja, majd jobb egérgomb felugró menü Új Fájl. *Ez csak a munkamenet alatt létezik!* - De persze bármilyen egyszerű szövegszerkesztővel is elkészíthetjük, majd a notebook mellé menthetjük

[28]:
```
# Első sor beolvasása
f = open('example_file.txt',"r")
s = f.readline()
f.close()
print(s)
```

```
2 körte
```

[29]:
```
# Megjegyzés: A readline a sortörést is beteszi az eredménybe.
s
```

[29]:
```
'2 körte\n'
```

[30]:
```
# A sortörést pl. a strip függvénnyel vághatjuk le:
s.strip()
```

[30]:
```
'2 körte'
```

[17]:
```
# Fájl sorainak beolvasása sztringlistába.
f = open('example_file.txt',"r")
l = f.readlines()
f.close()
print(l)
```

```
['2 körte\n']
```

Példa: Olvassuk be a korábban megírt `gyumolcs.txt` fájlt tartalmát adatpárok listájaként!

[33]:
```
# Sztring darabolása egy határoló jelsorozat mentén (tokenizálás).
line = 'aa,bb,ccc'
line.split(",")
```

[33]:
```
['aa', 'bb', 'ccc']
```

[34]:
```
# Alapértelmezés szerint a split fehér karakterek mentén darabol.
line = 'aa  bb\tccc\n'
line.split(",")
```

[34]:
```
['aa  bb\tccc\n']
```

[35]:
```
# Iterálás egy szövegfájl sorain.
with open("gyumolcs.txt") as f:
    for line in f:
        print(line)
```

```
0       alma

1       körte

2       szilva
```

[62]:
```
# Fájl első sorának átugrása, a további sorok tokenizálása.
data = []
f = open('gyumolcs.txt',"r")
f.readline() # 1. sor átugrása
for line in f:
    tok = line.strip().split()
    record = int(tok[0]), tok[1]
    data.append(record)
f.close()

print(data)
```

```
[(1, 'körte'), (2, 'szilva')]
```

5.2 Függvények: bevezetés, alapfogalmak

- Függvények a dokumentációban
- 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.
- Fontos különbséget tenni egy függvény *definíciója* és *meghívása* között:
  - A függvény definíciója megadja, hogy milyen bemenethez milyen kimenet rendelődjön (és milyen mellékhatások hajtódjanak végre). A függvény definíciója a programban általában csak egy helyen szerepel (ha több helyen szerepel, akkor az utolsó definíció lesz az érvényes.)
  - A függvény meghívása azt jelenti, hogy egy adott bemenethez kiszámítjuk a hozzárendelt értéket. Egy definiált függvényt a programban többször is meg lehet hívni.

[65]:
```
# Példa: n-edik gyök függvény definiálása.
def gyok(x:float , n:int = 2) -> float:
    """
    x -első változó
    n - második
    x n-edik gyökét számítja ki
    """
    return x**(1/n)
```

Ha a függvény első utasítása egy sztring, akkor ez lesz a függvény dokumentációs sztringje.

[68]:
```
# Dokumentációs sztring (docstring) lekérdezése.
# 'dunder' doc

print(gyok.__doc__)
```

```
x -első változó
n - második
x n-edik gyökét számítja ki
```

- Pythonban a függvényeknek *pozícionális* és *kulcsszó* paraméterei lehetnek.
  - Függvénydefiníciókor először a pozícionális majd a kulcsszó paramétereket kell felsorolni.
  - A pozícionális paramétereknek nincs alapértelmezett értékük, a kulcsszó paramétereknek van.
  - Mindegyik paramétertípusból lehet nulla darab is.
- Függvényhíváskor…
  - Az összes pozícionális paraméter értékét meg kell adni, olyan sorrendben, ahogy a definíciónál szerepeltek,
  - A kulcsszó paraméterek értékét nem kötelező megadni.

[70]:
```
# Gyök 2 kiszámítása.
gyok(2, 2)
```

[70]:
```
1.4142135623730951
```

[71]:
```
gyok(2)
```

[71]:
```
1.4142135623730951
```

[72]:
```
# Köbgyök 2 kiszámítása.
gyok(2, n = 3)
```

[72]:
```
1.2599210498948732
```

[73]:
```
# A második paramétert nem kell nevesíteni, ha az kulcsszó paraméter
gyok(2, 3)
```

[73]:
```
1.2599210498948732
```

[74]:
```
# Változónak értékül adhatunk függvényt.
y = gyok
```

[75]:
```
# nézzük meg az új függvény változó típusát, dokumentációs sztringjét!
print(type(y))
print(y.__doc__)
```

```
<class 'function'>

    x -első változó
    n - második
    x n-edik gyökét számítja ki
```

6 Függvények gyakorlás

6.1 Feladat: Celsius-Fahrenheit átváltás

Készítsünk adat fájlt, ami Celsius-Fahrenheit adatpárokat tartalmaz 0°C tól 100°C-ig 10 fokonként. Az átszámítás szabálya

$$[T_F] = [T_C] \cdot \frac{9}{5} + 32$$

Az átszámítást függvény segítségével végezzük el.

[76]:
```
# átszámító függvény:
def Fahrenheit(c):
    return c * 9 / 5 + 32
```

[77]:
```
# Celsius-Fahrenheit táblázatot tartalmazó fájl elkészítése.
file = open('celsius_fahrenheit.txt', 'w')
file.write('Celsius\tFahrenheit\n')
for c in range(-20, 41, 5):
```

```
    f = Fahrenheit(c)
    file.write(f'{c}\t{f}\n')
file.close()
```

6.2 Feladat: Prímtesztelés

Készítsünk függvényt, amely eldönti egy természetes számról, hogy prím-e!

[1]:
```
# 1. változat: függvény nélkül
# Ez a [2, n-1] intervallumon minden számmal ellenőrzi az oszthatóságot
n = 23
eredm = True
for k in range(2,n):
    if n%k == 0:
        eredm=False
        break         # kilép a ciklusból ha eljut ide
print(eredm)
```

```
True
```

[2]:
```
# 2. változat: függvénnyel
# javított verzió: Ez a [2, sqrt(n)] intervallumon ellenőriz csak

def is_prime(n):
    eredm = True
    n0 = int(n**0.5)+1
    for k in range(2,n0):
        if n%k == 0:
            eredm=False
            break
    return eredm
```

[3]:
```
# előző "igényesebben"
def is_prime(n):
    n0 = int(n**0.5)+1
    for k in range(2, n0):
        if n % k == 0:
            return False
    return True
```

[4]:
```
is_prime(123),is_prime(7),is_prime(1993)
```

[4]:
```
(False, True, True)
```

6.3 Szervezzük át a korábban már megírt programjainkat függvénnyé

6.3.1 Példa: n-szer n-es háromszög * karakterekből.

[5]:
```
# korábbi kódunk
n = 10
for hossz in range(1, n + 1):
    print("*" * hossz )
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

[6]:
```
# fügvénnyel amely stringet ad vissza
def haromszog(n):
    s = ""
    for elem in range(n):
        s +="*"*(elem+1)
        s += "\n"
    return s[0:-1]
```

[7]:
```
print(haromszog(10))
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

6.3.2 Példa: Magánhangzók statisztikája szótárral (angol kisbetűs szövegben).

[8]:
```
# korábbi kódunk

text = 'This is a short text for testing the algorithm.'
vowels = {'a', 'e', 'i', 'o', 'u'}
```

```
stat = dict()

for mgh in vowels:
    mgh_db = 0
    for betu in text.lower():
        if betu == mgh:
            mgh_db += 1
    print(mgh, ":", mgh_db )
    stat[mgh] = mgh_db
```

```
o : 3
e : 3
i : 4
a : 2
u : 0
```

[9]:
```
stat
```

[9]:
```
{'o': 3, 'e': 3, 'i': 4, 'a': 2, 'u': 0}
```

[10]:
```
# fügvénnyel

def mgh_stat(text):
    vowels = {'a', 'e', 'i', 'o', 'u'}

    stat = dict()

    for mgh in vowels:
        mgh_db = 0
        for betu in text.lower():
            if betu == mgh:
                mgh_db += 1
        stat[mgh] = mgh_db
    return stat
```

[11]:
```
mgh_stat("Ez egy proba szoveg")
```

[11]:
```
{'o': 2, 'e': 3, 'i': 0, 'a': 1, 'u': 0}
```

6.4 Lambda kifejezések

- A lambda kifejezés nem más, mint egysoros, névtelen függvény.
- Lambda kifejezések a dokumentációban

[12]:
```
# Példa lambda kifejezésre.
f = lambda x: x + 42
f(3)
```

[12]: 45

[13]:
```
# Egynél több bemenet is megengedett.
```

6.4.1 Lambda kifejezés alkalmazása rendezésnél

[14]:
```
# Párok listájának rendezése a második elem szerint.
pairs = [('alma', 22), ('körte', 11), ('barack', 33)]
sorted(pairs, key = lambda x: x[-1] )
```

[14]: [('körte', 11), ('alma', 22), ('barack', 33)]

[15]:
```
sorted(pairs)
```

[15]: [('alma', 22), ('barack', 33), ('körte', 11)]

[16]:
```
# Az előző feladat megoldása lambda kifejezés nélkül külön függvényt definiálva
pairs = [('alma', 22), ('körte', 11), ('barack', 33)]
```

[17]:
```
# Szótárkulcsok rendezése az értékek szerint.
words = {'king': 203, 'denmark': 24, 'queen': 192}
```

6.5 Teknős grafika függvényekkel

[18]:
```
#import turtle
```

[19]:
```
!pip3 install ColabTurtle
```

```
Requirement already satisfied: ColabTurtle in
/home/domotor/anaconda3/envs/pylatest/lib/python3.9/site-packages (2.1.0)
```

[20]:
```
import ColabTurtle.Turtle as turtle
```

[21]:
```
# lássuk, hogyan működik a teknőc
turtle.initializeTurtle(initial_speed = 3 )
turtle.bgcolor(1,1,1)
turtle.color("red")    # red = piros színű a toll
turtle.forward(200)    # forward = előre megyünk 200 pixelt
turtle.left(90)        # left = balra fordulunk 90 fokot
turtle.forward(200)    # forward = előre megyünk 200 pixelt

turtle.penup()         # most felvesszük a tollat
turtle.forward(100)    # forward = előre megyünk 100 pixelt
turtle.pendown()       # és letesszük a tollat
```

```
<IPython.core.display.HTML object>
```

[22]:
```
# írjunk függvényt ami megvalósít egy adott méretű továbbosonoást
def oson(tavolsag):
    turtle.penup()
    turtle.forward(tavolsag)
    turtle.pendown()
```

[23]:
```
# írjunk függvényt amely négyzetet rajzol adott színnel és mérettel
def negyzet(meret : int, szin: str) -> None:
    '''
    adott meretü és színű négyzetet rajzol teknősgrafikával.
    '''
    turtle.color(szin)
    for i in range(4):
        turtle.forward(meret)
        turtle.left(90)
```

[24]:
```
# játszunk el a függvényeinkkel!
#Rajzoljunk több elfofgatott négyzetet.
#Rajroljunk egymás mellé más-más színű négyzeteket!

turtle.initializeTurtle(initial_speed=13)

for i in range(10):
    meret = (i+1)*20
    turtle.left(i*1)
    negyzet(meret,"red")
```

```
<IPython.core.display.HTML object>
```

[25]:
```
# írjunk függvényt ami testzőleges szabályos sokszöget rajzol!
def Nszog(N, meret, szin):
    '''
    N oldalú szabályos sokszöget rajzol meret oldalhosszal
    '''
    turtle.color(szin)
    for i in range(N):
        turtle.forward(meret)
        turtle.left(360/N)
```

[26]:
```
turtle.initializeTurtle(initial_speed=13)

for i,szin in enumerate(["red","green","violet","orange","blue"]):
    meret = (i+1)*20
    turtle.left(i*5)
    oson(-i*5)
    Nszog(6,meret,szin)
```

```
<IPython.core.display.HTML object>
```

[27]:
```
# rajzoljunk kisházat a meglévő sokszögek segítségével
```

[28]:
```
def haz(meret):
    turtle.right(90)
    # házikó fő része: egy négyzet
    negyzet(meret, "red")
    # felmegyünk a tető rajzolásához:
    turtle.left(90)
    oson(meret)
    turtle.right(90)
    # most jön a tető
    Nszog(3, meret, "blue")
    # elmegyünk az ablak helyére:
    oson(meret*0.2)
    turtle.right(90)
    oson(meret*0.5)
    turtle.left(90)
    # most jön az ablak
    negyzet(meret*0.3, "green")
    # ... és vissza a kiindulópontra. Ha nem mennénk, összezavarodhatnánk!
    turtle.right(90)
    oson(meret*0.5)
    turtle.right(90)
    oson(meret*0.2)
    turtle.right(180)
```

[29]:
```
turtle.initializeTurtle(initial_speed=13)

haz(100)
```

```
<IPython.core.display.HTML object>
```

7 Modulok és csomagok. Ismerkedés a numpy és a matplotlib csomag lehetőségeivel

7.1 Modulok és Csomagok néhány példával

Modul: Python nyelvű fájl. Modulok a dokumentációban - 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. Csomagok a dokumentációban - Egy csomag alcsomagokat/almodulokat 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 (https://pypi.python.org/pypi).

7.1.1 Modul vagy csomag importálása, majd abból a szükséges függvény meghívása

[1]:
```
# Modul/csomag importálása.
import random
```

[2]:
```
# A random egészet sorsoló függvény meghívása
random.randint(1, 100)
```

[2]: 75

[8]:
```
# de egy listából is vehetünk ki véletlenszerűen elemet
l = ["alma", "körte","szilva","málna"]
random.choice(l)
```

[8]: 'málna'

7.1.2 Csak a szükséges függvények importálása egy modulból/csomagból.

[8]:
```
# Egyetlen függvény importálása egy modulból/csomagból.
from random import randint
```

[9]:
```
randint(10, 20)
```

[9]: 14

[9]:
```
# Függvény importálása almodulból/alcsomagból.
from os.path import dirname
```

[10]:
```
dirname('/tmp/pistike/a.txt')
```

[10]: '/tmp/pistike'

7.1.3 Modul/csomag teljes tartalmának importálása

Megjegyzés: Ez a megoldás általában kerülendő. Keveredhetnek a függvény nevek.

[ ]:
```
# from random import *
```

7.1.4 Modul vagy csomag importálása rövid néven

[10]:
```
# rövidítve
import random as rd
```

[11]:
```
rd.randint(0,10)
```

[11]: 1

7.2 Megjegyzés: listák másolásáról

- Másolás - copy csomag a dokumentációban
- Sekély (shallow) és mély (deep) másoló függvényt tartalmaz.

[12]:
```
import copy as cp
```

a = b értékadás számokkal

[13]:
```
a = 2
b = a
print("a = ", a, ";\tb = ", b)
a = 3
print("a = ", a, ";\tb = ", b)
```

```
a =  2 ;        b =  2
a =  3 ;        b =  2
```

a = b értékadás listákkal - NEM végez másolást - csak hivatkozást hoz létre

[14]:
```
# Pythonban az értékadás NEM végez másolást, csak hivatkozást hoz létre.
a = [1, 2, 3]
b = a
print("a = ", a, ";\tb = ", b)
b[0] = 2
print("a = ", a, ";\tb = ", b)
```

```
a =  [1, 2, 3] ;        b =  [1, 2, 3]
a =  [2, 2, 3] ;        b =  [2, 2, 3]
```

Ha független másolatot szeretnénk ahhoz kell a copy csomag

[15]:
```
# Sekély másolat készítése.
a = [1, 2, 3]
b = cp.copy(a)
print("a = ", a, ";\tb = ", b)
b[0] = 2
a[-1] =10
print("a = ", a, ";\tb = ", b)
```

```
a =  [1, 2, 3] ;        b =  [1, 2, 3]
a =  [1, 2, 10] ;       b =  [2, 2, 3]
```

[19]:
```
# Sekély másolat készítése egy listák listája objektumról.
a = [[1], [2], [2]]
b = cp.copy(a)
print("a = ", a, ";\tb = ", b)
```

```
a =  [[1], [2], [2]] ;  b =  [[1], [2], [2]]
```

[20]:
```
# A cp.copy() csak az adatszerkezet legfelső szintjén végez másolást.
b[0][0] = 10
print("a = ", a, ";\tb = ", b)
```

```
a =  [[10], [2], [2]] ; b =  [[10], [2], [2]]
```

[21]:
```
# Mély másolat készítése egy listák listája objektumról.
a = [[1], [2], [3]]
b = cp.deepcopy(a)
b[0][0] = 10
print("a = ", a, ";\tb = ", b)
```

```
a =  [[1], [2], [3]] ;  b =  [[10], [2], [3]]
```

7.3 NumPy csomag

A NumPy egy alacsony szintű matematikai csomag, numerikus számításokhoz.

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

A NumPy külső csomag. A Colab felületén ez is elérhető.

De a saját gépünkön szükség van telepítésére. Többféle lehetőség van, például: - `pip install numpy --user` - `sudo apt-get install python3-numpy` - `conda install numpy`

[22]:
```
# A NumPy modul importálása np néven.
import numpy as np
```

[23]:
```
# Verzió lekérdezése.
np.__version__
```

[23]:
```
'1.23.4'
```

7.3.1 Állandók és függvények

[24]:
```
# Meguan a pi és az e állandó sok tizedes jegyre:
np.pi, np.e
```

[24]:
```
(3.141592653589793, 2.718281828459045)
```

[25]:
```
# cos(x) sin(x) trigonometrikus függvények
np.cos(np.pi/4)
```

[25]:
```
0.7071067811865476
```

[26]:
```
# sqrt(x) gyökvonás
np.sqrt(3)
```

[26]:
```
1.7320508075688772
```

[27]:
```
# Van beépített függvény legnagyobb közös osztóra
# greatest common divisor --> gcd()

np.gcd(60,18)
```

[27]:
```
6
```

[36]:
```
print(np.gcd.__doc__)
```

```
gcd(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K',
dtype=None, subok=True[, signature, extobj])

Returns the greatest common divisor of ``|x1|`` and ``|x2|``

Parameters
----------
x1, x2 : array_like, int
    Arrays of values.
    If ``x1.shape != x2.shape``, they must be broadcastable to a common
    shape (which becomes the shape of the output).

Returns
-------
y : ndarray or scalar
    The greatest common divisor of the absolute value of the inputs
    This is a scalar if both `x1` and `x2` are scalars.

See Also
--------
lcm : The lowest common multiple

Examples
--------
>>> np.gcd(12, 20)
4
>>> np.gcd.reduce([15, 25, 35])
5
>>> np.gcd(np.arange(6), 20)
array([20,  1,  2,  1,  4,  5])
```

7.3.2 Tömbök létrehozása

Példa: egész számokból álló tömb létrehozása

[83]:
```
# egész számokból álló 1x3 tömb - SORVEKTOR
a = np.array([2, 3, 4])
```

[84]:
```
print(a)
```

```
[2 3 4]
```

[85]:
```
# A tömb objektum típusa.
type(a)
```

[85]:
```
numpy.ndarray
```

[86]:
```
# a tömb elemei módosíthatóak
a[2] = 5
print(a)
```

```
[2 3 5]
```

Tömb jellemzők lekérdezése

[56]:
```
# Hány dimenziós a tömb?
a.ndim
```

[56]:
```
1
```

[65]:
```
# A tömb alakja/mérete.
a.shape
```

[65]:
```
(3,)
```

[66]:
```
len(a)
```

[66]:
```
3
```

[59]:
```
# összes elem
a.size
```

[59]:
```
3
```

[60]:
```
# Az elemek típusának lekérdezése.
# A NumPy tömbök homogének, azaz egyféle adatok vannak benne
a.dtype
```

[60]:
```
dtype('int64')
```

Feladat: Hozzunk létre egy 2 x 3-as tömböt, valós számokkal

Kérdezzük le a dimenziók számát, a tömb méretét, az elemek számát, az elemek típusát!

[61]:
```
# Hozzunk létre egy 2 x 3-as tömböt, valós számokkal
b = np.array( [ [ 0.1, 0.2  , 4.5 ],
                [4.44, 5.12 , np.pi]  ]  )
```

[62]:
```
print(b)
```

```
[[0.1        0.2        4.5        ]
 [4.44       5.12       3.14159265]]
```

[64]:
```
# Dimenziók száma, mérete, az elemek típusa.
b.ndim, b.size, b.shape, len(b), b.dtype
```

[64]: (2, 6, (2, 3), 2, dtype('float64'))

További módszerek tömb létrehozásra `genfromtxt()`; `zeros()`; `ones()`; `arange()`; `concatenate()`

[50]:
```
# készítsünk  egy "matrix.txt" nevű fájlt 2 sorban 3 egész szám szóközzel␣
 ↪elválasztva
# majd olvassuk be a txt fájlt tömbként
np.genfromtxt('matrix.txt')
```

[50]:
```
array([[1., 0., 1.],
       [1., 1., 0.]])
```

[51]:
```
# Nullákból álló tömb létrehozása 1dimenziós
np.zeros(7)
```

[51]: array([0., 0., 0., 0., 0., 0., 0.])

[67]:
```
# Nullákból álló tömb létrehozása többdimenziós
np.zeros((2,3))
```

[67]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

[69]:
```
# Egyesekből álló tömb létrehozása
np.ones((5, 1))
```

[69]:
```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.]])
```

[54]:
```
# Értéktartomány létrehozása a lépésköz megadásával.
np.arange(-5, 5, 0.5)
```

[54]:
```
array([-5. , -4.5, -4. , -3.5, -3. , -2.5, -2. , -1.5, -1. , -0.5,  0. ,
        0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5])
```

[55]:
```
# Vektorok összefűzése.
a = np.array([2, 3])
b = np.array([44, 55, 66])
np.concatenate([a, b])
```

[55]:
```
array([ 2,  3, 44, 55, 66])
```

7.3.3 Másolás beépítve

[70]:
```
a = np.array([1,2,3])
b = a.copy()
print("a = ", a, ";\tb = ", b)
b [0] = 10
print("a = ", a, ";\tb = ", b)
```

```
a =  [1 2 3] ;  b =  [1 2 3]
a =  [1 2 3] ;  b =  [10  2  3]
```

7.3.4 Elemek és résztömbök

[88]:
```
# Hozzunk létre egy példamátrixot!
a = np.array([[3, 4, 5],
              [6, 7, 8]])
```

[89]:
```
# Elem kiválasztása (az indexelés 0-tól indul).
print("2. sor 3. eleme: ", a[1, 2])    # 1 a sorindex, 2 az oszlopindex
```

```
2. sor 3. eleme 8
```

[94]:
```
print("1. sor 2. eleme: ", a[0, 1])
```

```
1. sor 2. eleme:  4
```

[95]:
```
# Teljes sor kiválasztása.
a[1,  ]
```

[95]:
```
array([6, 7, 8])
```

[96]:
```
# Így is lehetne.
a[1]
```

[96]:
```
array([6, 7, 8])
```

[97]:
```
# Oszlop kiválasztása.
a[:, 1]
```

[97]:
```
array([4, 7])
```

[98]:
```
# Adott indexű oszlopok kiválasztása.
a[:, [0, 2]]
```

[98]:
```
array([[3, 5],
       [6, 8]])
```

[99]:
```
# Elemek kiválasztása logikai feltétel alapján.
a[a % 2 == 0]     # páros elemek kiválasztása
```

[99]:
```
array([4, 6, 8])
```

[100]:
```
a % 2 == 0
```

[100]:
```
array([[False,  True, False],
       [ True, False,  True]])
```

[101]:
```
a[a > 5]
```

[101]:
```
array([6, 7, 8])
```

[104]:
```
# A tömb elemei módosíthatóak.
a[0, -1] = 100
a
```

[104]:
```
array([[  3,   4, 100],
       [  6,   7,   8]])
```

[75]:
```
a[:, -1]
```

[75]:
```
array([100,   8])
```

[76]:
```
# Oszlop módosítása.
a[:, -1] = 40, 70
a
```

[76]:
```
array([[100,   4,  40],
       [  6,   7,  70]])
```

7.3.5 Vektor műveletek (Tömbműveletek) Koordinátánként

[77]:
```
# Hozzunk létre 2 példatömböt!
a = np.array([1, 7 ,3])
b = np.array([1, 1, 1])
```

[78]:
```
# Elemenkénti összeadás.
a + b
```

[78]: array([2, 8, 4])

[79]:
```
# Elemenkénti kivonás.
a - b
```

[79]: array([0, 6, 2])

[80]:
```
# számmal szorzás
2 * a
```

[80]: array([2, 14, 6])

[82]:
```
# Elemenkénti szorzás.
print(a,b)
a*b
```

```
[1 7 3] [1 1 1]
```

[82]: array([1, 7, 3])

[83]:
```
# Elemenkénti osztás.
a/b
```

[83]: array([1., 7., 3.])

[85]:
```
# Elemenkénti egészosztás.
b//a
```

[85]: array([1, 0, 0])

[86]:
```
# Elemenkénti hatványozás.
a**2
```

[86]: array([1, 49, 9])

[87]:
```
# 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-87-52107b4937c9> in <cell line: 4>()
      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,)
```

[ ]:
```
# Két vektor skaláris szorzata.
a = np.array([3, 4, 5])
b = np.array([2, 2, 2])
a @ b
```

7.3.6 Függvények és tömbök

[105]:
```
# Hozzunk létre egy példamátrixot!
a = np.array([[3, 4, 5],
              [6, 7, 8]])
```

[106]:
```
# Elemenkénti függvények (exp, log, sin, cos, ...).
np.cos(a) # koszinusz
```

[106]:
```
array([[-0.9899925 , -0.65364362,  0.28366219],
       [ 0.96017029,  0.75390225, -0.14550003]])
```

[107]:
```
np.log(a) # természetes alapú logaritmus
```

[107]:
```
array([[1.09861229, 1.38629436, 1.60943791],
       [1.79175947, 1.94591015, 2.07944154]])
```

[108]:
```
# álltalunk megadott függvény is alkalmazható array-re:
def f(x):
    return (x-2)**2
print(f(0.5))
print(f(a))
```

```
2.25
[[ 1  4  9]
 [16 25 36]]
```

7.3.7 Statisztikai műveletek (min, max, sum, mean, std).

[109]:
```
# Hozzunk létre egy példamátrixot!
a = np.array([[3, 4, 5],
              [6, 7, 8]])
```

[110]:
```
a.sum()
```

[110]: 33

[111]:
```
a.min(), a.max()
```

[111]: (3, 8)

[112]:
```
a.mean()
```

[112]: 5.5

[113]:
```
# minimum és maximum hely (argmin,argmax)
a.argmax()
```

[113]: 5

7.3.8 Keresés - np.where()

[101]:
```
# Példa: Mely indexeknél találhatók az 5-nél kisebb elemek?
a = np.array([3, 10, 11, 4, 7, 8])

print(a[a<5])
np.where(a<5)
```

```
[3 4]
```

[101]: (array([0, 3]),)

7.3.9 Rendezés

[118]:
```
# Rendezés helyben.
a = np.array([3, 10, 11, 4, 7, 8])
a.sort()
```

[119]:
```
print(a)
```

```
[ 3  4  7  8 10 11]
```

[121]:
```
# Rendezés új tömbbe.
a = np.array([3, 10, 11, 4, 7, 8])
```

```
np.sort(a)
```

[121]:
```
array([ 3,  4,  7,  8, 10, 11])
```

[122]:
```
# Rendezés csökkenő sorrendbe.
np.sort(a)[::-1]
```

[122]:
```
array([11, 10,  8,  7,  4,  3])
```

7.4 Matplotlib

A Matplotlib adatok ábrázolásához hasznos csomag.

- Nyomdai minőségben testreszabható grafikon rajzolás
- Sokféle képfájlként kimenthető eredmény
- De animációt is lehet vele készíteni

A Matplotlib is külső csomag. A Colab felületén ez is elérhető.

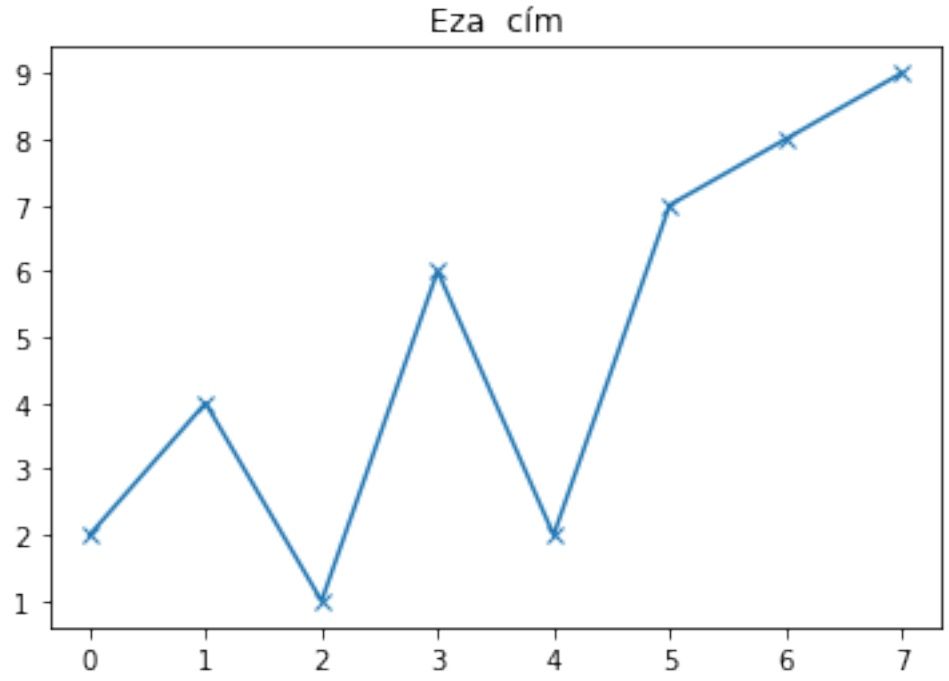
De a saját gépünkön szükség van telepítésére. Többféle lehetőség van, például: - `pip install matplotlib --user` - `conda install -c conda-forge matplotlib`

[124]:
```
import matplotlib.pyplot as plt

# Colab /Jupyter notebook speciális beállítás
%matplotlib inline
```

[125]:
```
# kreáljunk egy adatsort és plt.plot() függvénnyel rajzoljuk ki
data = np.array([2,4,1,6,2,7,8,9] )
# data.size

plt.plot(data , "x-")
plt.title("Eza  cím")
plt.show()
```



8 Alkalmazás: Gyorsulás adatok feldolgozása

Ez a notebook egy lift mozgása során a telefon gyorsulásmérőjével gyűjtött adatok - beolvasását, - ábrázolását, - majd a pillanatnyi sebesség és helyzet kiszámítását tartalmazza

8.0.1 Hogyan mozog, hely, sebesség és gyorsulás kapcsolata egyenes mentén

Hely adatok: - Minden időpontban tudom a helykoordinátát: $x(t)$ függvény - Sok időpontban tudom a helykoordinátát (MÉRÉS): $x_i = x(t_i)$ adatok

Hely adatokból sebesség: - Pillanatnyi sebesség $v(t) = \frac{\Delta x}{\Delta t} = \frac{x(t) - x(t - \Delta t)}{\Delta t}$, ahol Δt nagyon rövid. - Adatokkal $v_i = \frac{x_i - x_{i-1}}{\Delta t}$, ahol Δt kellően rövid

Sebesség adatokból gyorsulás: - Pillanatnyi gyorsulás $a(t) = \frac{\Delta v}{\Delta t} = \frac{v(t) - v(t - \Delta t)}{\Delta t}$, ahol Δt nagyon rövid. - Adatokkal $a_i = \frac{v_i - v_{i-1}}{\Delta t}$, ahol Δt kellően rövid

Gyorsulás adatokból sebesség: Az előző összefüggést átrendezve - Adatokkal $v_i = v_{i-1} + a_i \cdot \Delta t = v_{i-1} + a_i \cdot (t_i - t_{i-1})$

Sebesség adatokból hely adat: teljesen hasonló számítással - Adatokkal $x_i = x_{i-1} + v_i \cdot \Delta t = x_{i-1} + v_i \cdot (t_i - t_{i-1})$

8.0.2 A szükséges csomagok importálása

[2]:
```
import pandas as pd         # Pandas csomag - adatkezelés
import numpy as np          # Numpy  csomag - numerikus számítások
import matplotlib.pyplot as plt    # MatPlotLib - ábrázolás
%matplotlib inline
```

8.0.3 Fájl beolvasása

Előtte győződjünk meg róla, hogy az aktuálisan használt könyvtárunkba már bemásoltuk a fájlt - A fájl letölthető innen: Lift_gyorsulas_adatok_nyers.xlsx"

- pandas csomag `read_excel()` eljárása

[2]:
```
input_file="Lift_gyorsulas_adatok_nyers.xlsx"

adata = pd.read_excel(input_file)
```

[3]:
```
adata
```

[3]:

	Time (s)	Acceleration x (m/s^2)	Acceleration y (m/s^2)	\
0	0.015842	0.099	-0.024	
1	0.020933	-0.015	0.052	
2	0.026023	0.118	0.052	
3	0.031113	0.060	0.032	
4	0.036203	0.099	0.109	
…	…	…	…	
3298	16.800199	0.079	0.090	
3299	16.805289	0.079	0.052	
3300	16.810379	0.022	0.071	
3301	16.815469	0.099	0.013	
3302	16.820558	0.079	0.032	

	Acceleration z (m/s^2)	Absolute acceleration (m/s^2)
0	9.797	9.797530
1	9.969	9.969147
2	9.912	9.912838
3	9.988	9.988231
4	9.912	9.913093
…	…	…
3298	10.084	10.084711
3299	9.816	9.816456
3300	9.816	9.816281
3301	9.893	9.893504
3302	10.008	10.008363

[3303 rows x 5 columns]

A beolvasott adatok DataFrame-je

- `.head()` és `.tail()` segít az adatok egy részét megjeleníteni
- síma `[ : ]` is működik most, de jobb az `.iloc[ : ]`

[4]:
```
adata.iloc[0:10]
```

[4]:
```
   Time (s)  Acceleration x (m/s^2)  Acceleration y (m/s^2)  \
0  0.015842                   0.099                  -0.024
1  0.020933                  -0.015                   0.052
2  0.026023                   0.118                   0.052
3  0.031113                   0.060                   0.032
4  0.036203                   0.099                   0.109
5  0.041294                   0.003                   0.071
6  0.046384                   0.099                   0.128
7  0.051475                   0.118                   0.128
8  0.056565                   0.099                  -0.024
9  0.061654                   0.003                   0.109

   Acceleration z (m/s^2)  Absolute acceleration (m/s^2)
0                   9.797                       9.797530
1                   9.969                       9.969147
2                   9.912                       9.912838
3                   9.988                       9.988231
4                   9.912                       9.913093
5                   9.893                       9.893255
6                   9.873                       9.874326
7                  10.008                      10.009514
8                   9.835                       9.835528
9                   9.873                       9.873602
```

A beolvasott adatok néhány jellemzője:

- `.shape` – méretek
- `.columns` – oszlop nevek
- `.dtypes` – oszlopok adatípusai
- `.describe()` – statisztikai jellemzők

[5]:
```
# méretek
len(adata)
```

[5]:
```
3303
```

[6]:
```
# összefoglaló
adata.describe()
```

[6]:
```
          Time (s)  Acceleration x (m/s^2)  Acceleration y (m/s^2)  \
count  3303.000000             3303.000000             3303.000000
```

```
mean      8.418001                0.076068                0.078240
std       4.853175                0.047409                0.057631
min       0.015842               -0.150000               -0.196000
25%       4.217066                0.041000                0.032000
50%       8.417952                0.079000                0.071000
75%      12.618917                0.099000                0.109000
max      16.820558                0.252000                0.473000

       Acceleration z (m/s^2)  Absolute acceleration (m/s^2)
count             3303.000000                    3303.000000
mean                 9.896643                       9.897527
std                  0.251682                       0.251619
min                  7.843000                       7.848925
25%                  9.835000                       9.835584
50%                  9.893000                       9.894010
75%                  9.969000                       9.969316
max                 10.601000                      10.601933
```

[9]:
```
# oszlopnevek
adata.columns
```

[9]:
```
Index(['Time (s)', 'Acceleration x (m/s^2)', 'Acceleration y (m/s^2)',
       'Acceleration z (m/s^2)', 'Absolute acceleration (m/s^2)'],
      dtype='object')
```

[8]:
```
  # adatípusok
adata.dtypes
```

[8]:
```
Time (s)                        float64
Acceleration x (m/s^2)          float64
Acceleration y (m/s^2)          float64
Acceleration z (m/s^2)          float64
Absolute acceleration (m/s^2)   float64
dtype: object
```

Nevezzük át az oszlopkat használhatóbb (rövidebb, nem szóközös) nevekre szótárral érdemes magadni a kapcsolatot az új és a régi oszlop nevek közt - `.rename(columns = ...)` - `inplace = True` kulcs paraméter igazra állítva ezzel át is írja az adatokat

[11]:
```
# új oszlopnevek listája
uj_oszlop = ["t", "ax", "ay", "az", "aabs"]
```

[18]:
```
# érdemes a zippelést használni a szótár előállításához:
atnevez = dict(zip(adata.columns, uj_oszlop))
atnevez
```

[18]:
```
{'Time (s)': 't',
 'Acceleration x (m/s^2)': 'ax',
 'Acceleration y (m/s^2)': 'ay',
 'Acceleration z (m/s^2)': 'az',
 'Absolute acceleration (m/s^2)': 'aabs'}
```

[21]:
```
# átnevezés
adata.rename(columns = atnevez, inplace = True)
```

[22]:
```
adata
```

[22]:

	t	ax	ay	az	aabs
0	0.015842	0.099	-0.024	9.797	9.797530
1	0.020933	-0.015	0.052	9.969	9.969147
2	0.026023	0.118	0.052	9.912	9.912838
3	0.031113	0.060	0.032	9.988	9.988231
4	0.036203	0.099	0.109	9.912	9.913093
…	…	…	…	…	…
3298	16.800199	0.079	0.090	10.084	10.084711
3299	16.805289	0.079	0.052	9.816	9.816456
3300	16.810379	0.022	0.071	9.816	9.816281
3301	16.815469	0.099	0.013	9.893	9.893504
3302	16.820558	0.079	0.032	10.008	10.008363

[3303 rows x 5 columns]

Hívatkozás egy-egy oszlopra

- `adata["cimke"]`
- `adata.cimke` – ha nincs benne szóköz furcsa karakter…

[24]:
```
adata["aabs"]
```

[24]:
```
0        9.797530
1        9.969147
2        9.912838
3        9.988231
4        9.913093
          …
3298    10.084711
3299     9.816456
3300     9.816281
3301     9.893504
3302    10.008363
Name: aabs, Length: 3303, dtype: float64
```

[26]:
```
adata[["t", "az"]].head(2)
```

[26]:
```
          t     az
0  0.015842  9.797
1  0.020933  9.969
```

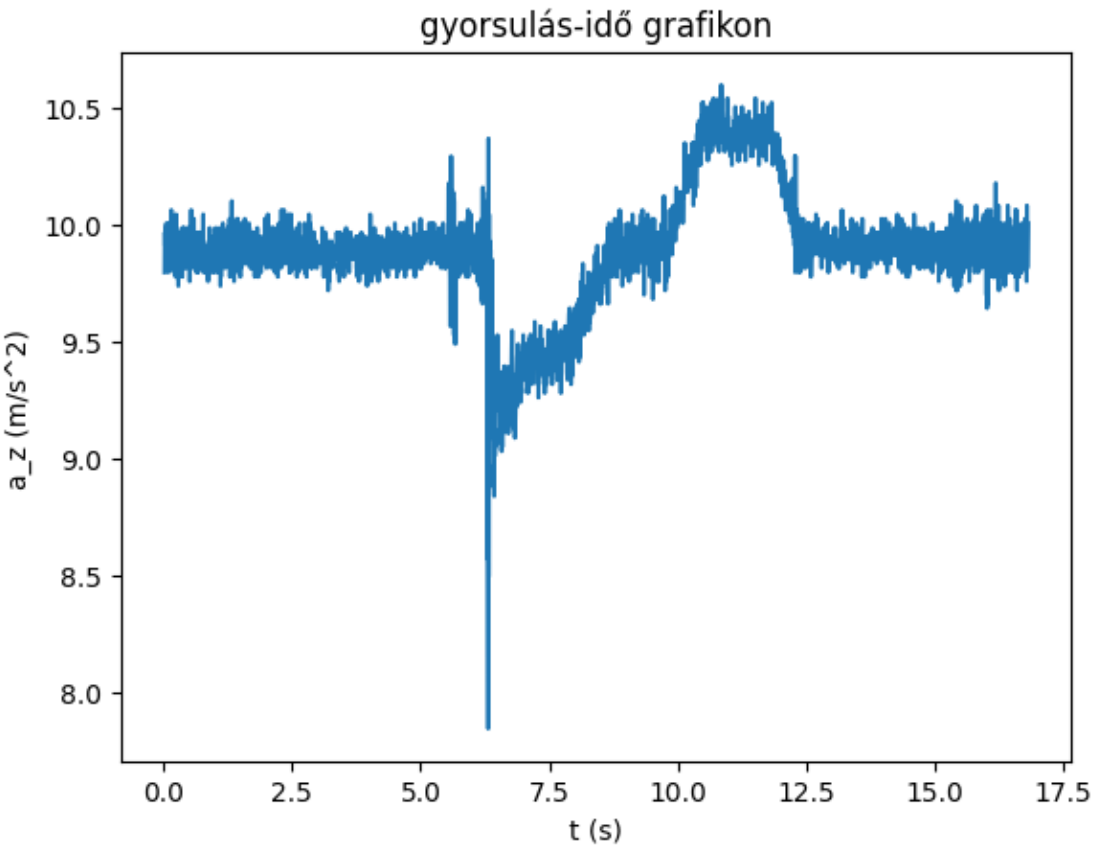
[27]:
```
adata.az
```

[27]:
```
0        9.797
1        9.969
2        9.912
3        9.988
4        9.912
         …
3298    10.084
3299     9.816
3300     9.816
3301     9.893
3302    10.008
Name: az, Length: 3303, dtype: float64
```

Ábrázoljuk a z irányú gyorsulás adatokat

- `plt.plot()` – matplotlib ábrázolás
- `plt.xlabel()` – tengely nevek
- `plt.title()` – grafikon cím

[43]:
```
plt.plot(adata.t, adata["az"])
plt.title("gyorsulás-idő grafikon")
plt.xlabel("t (s)")
plt.ylabel("a_z (m/s^2)")
plt.show()
```



8.0.4 Az adatok feldolgozása

fölösleges oszlopok törlése

- `del` vagy
- `.drop(columns= )` használata

[31]:
```
# fölösleges oszlopok törlése
del adata["ax"]
```

[34]:
```
adata.drop(columns=["ay","aabs"],inplace = True)
```

[35]:
```
adata
```

[35]:

	t	az
0	0.015842	9.797
1	0.020933	9.969
2	0.026023	9.912
3	0.031113	9.988
4	0.036203	9.912
…	…	…

```
3298  16.800199  10.084
3299  16.805289   9.816
3300  16.810379   9.816
3301  16.815469   9.893
3302  16.820558  10.008

[3303 rows x 2 columns]
```

a nehézségi gyorsulás állandó értékének levonása

[38]:
```
g = adata["az"][0:1000].mean()
```

[40]:
```
adata["az0"] = adata["az"]-g
```

[41]:
```
adata
```

[41]:
```
              t      az       az0
0      0.015842   9.797 -0.092017
1      0.020933   9.969  0.079983
2      0.026023   9.912  0.022983
3      0.031113   9.988  0.098983
4      0.036203   9.912  0.022983
…           …      …        …
3298  16.800199  10.084  0.194983
3299  16.805289   9.816 -0.073017
3300  16.810379   9.816 -0.073017
3301  16.815469   9.893  0.003983
3302  16.820558  10.008  0.118983

[3303 rows x 3 columns]
```

“Integrálás”

[46]:
```
def dint(time,a,v0=0.0):
    N=len(time)                    # a tömb első dimenziója
    intt=np.zeros(N)           # az integrál értékek tömbje
    # Ide jön a lényegi rész
    intt[0] = v0
    for i in range(1,N):
        intt[i] = a[i]*(time[i] - time[i-1])+intt[i-1]
    return intt
```

pillanatnyi sebesség v(t) kiszámítása és ábrázolása

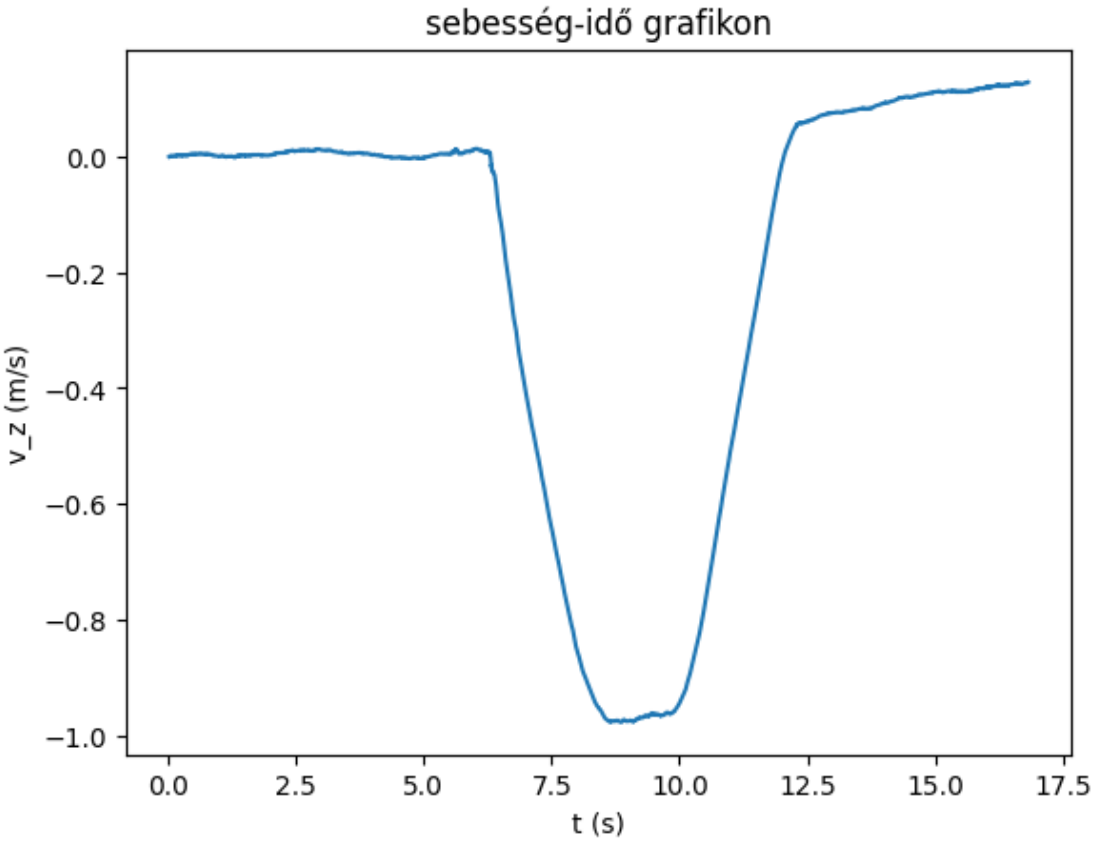
[48]:
```
adata["v"] =  dint(adata["t"], adata["az0"])
```

[49]:
```
adata.head(5)
```

[49]:

	t	az	az0	v
0	0.015842	9.797	-0.092017	0.000000
1	0.020933	9.969	0.079983	0.000407
2	0.026023	9.912	0.022983	0.000524
3	0.031113	9.988	0.098983	0.001028
4	0.036203	9.912	0.022983	0.001145

[50]:
```
plt.plot(adata.t, adata["v"])
plt.title("sebesség-idő grafikon")
plt.xlabel("t (s)")
plt.ylabel("v_z (m/s)")
plt.show()
```



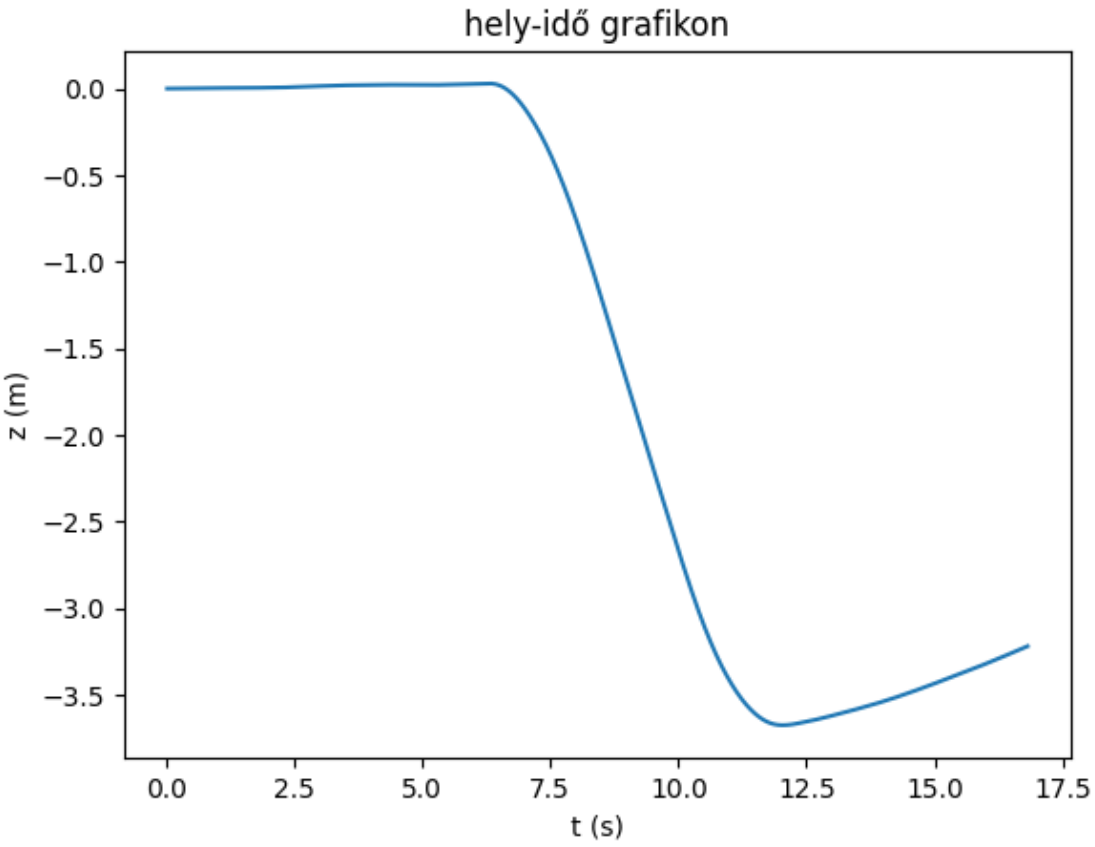
Megjegyzés: A grafikon nem tökéletes, hiszen a liftnek meg kellene állnia, azaz a mozgás végén is 0 m/s sebesség kellene, ami enyhén emelkedik.

A hiba fő oka a mérés hibájában keresendő. A telefonok gyorsulás mérője egy hoszzabb idejű mérés során elállítódik. A mérés végén, már nem ugyan az a g értéket méri

pillanatnyi hely x(t) kiszámítása és ábrázolása

[51]:
```
adata["z"] =  dint(adata["t"], adata["v"])
```

[52]:
```
plt.plot(adata.t, adata["z"])
plt.title("hely-idő grafikon")
plt.xlabel("t (s)")
plt.ylabel("z (m)")
plt.show()
```



[ ]:
```

```

9 Mozgásegyenlet megoldása 1 dimenzióban (egyenes menti mozgások)

Ez a notebook különböző egyenes menti erőhatások esetén numerikusan megodlja a mozgásegyenletet.

Az erők ismeretében már tudjuk a gyorsulás aktuális értékét. Innen pedig az előző alkalommal használt számítások logikáját követjük. Tehát kiválasztott Δt időlépésekben ki tudjuk számítani a - pillanatnyi sebesség és - hely adatokat - Illetve ezeket tudjuk ábrázolni az idő függvényében

9.1 Elmélet

9.1.1 Emlékeztetőül

Gyorsulás adatokból sebesség: $\boxed{v_i = v_{i-1} + a_i \cdot \Delta t}$

Sebesség adatokból helyzet: $\boxed{x_i = x_{i-1} + v_i \cdot \Delta t}$

Ehhez persze kell egy ismert v_0 kezdő sebesség és x_0 kezdő helyzet! (**kezdeti feltételek**)

9.1.2 Newton törvények alapján a mozgásegyenlet

Model: pontszerű test

$$\boxed{\sum \vec{F} = m \cdot \vec{a}}$$

- Az egyes kölcsönhatásokat egy-egy F erővel tudjuk leírni (erőtörvények), amelyek vektori módon összegződnek. (egynes mentén előjelesen!)
- Az így kapott eredő erő okozza a test gyorsulását.
- Ez a gyorsulás arányos az eredő erővel és
- az arányossági tényező a testre jellemző állandó, amit tömegnek hívunk.

9.1.3 Néhány hasznos erőtörvény

Ezeket kísérleteket elemezve kapjuk, a Newton törvények nem mondják meg.

- Nehézségi erő a Földfelszín közelében: $\vec{F} = m \cdot \vec{g}$
- Rugóban ébredő erő (egyenes mentén): $ F=-D x$
- Közegellenállás kis sebességek: $\vec{F} = -C \cdot \vec{v}$
- Közegellenállás nagy sebességek: $\vec{F} = -C \cdot \vec{v} \cdot |\vec{v}|$ pontosabban $\vec{F} = -\frac{1}{2} \cdot C \cdot \rho \cdot A \cdot \vec{v} \cdot |\vec{v}|$, ahol C az alaktényező, ρ a közeg sűrűsége, A a homlokfelület (mozgásirányra merőleges felület).
- Csúszási súrlódás: felületet összenyomó erővel arányos nagyságú mozgást akadályozó $F = \mu_0 \cdot N$

9.2 Egyszerű mozgásegyenlet-megoldó

9.2.1 A szükséges csomagok importálása

[2]:
```
import numpy as np                  # Numpy  csomag - numerikus számítások
import matplotlib.pyplot as plt     # MatPlotLib - ábrázolás
%matplotlib inline
```

9.2.2 A léptetés megvalósítása függvénnyel

Mik a releváns paraméterek?

- aktuális helyzet x
- aktuális sebesség v
- erők eredője: F függhet a tömegtől, sebességtől, helyzettől.
- tömeg, mert a gyorsulás $a = \dfrac{F}{m}$,
- időlépés: Δt

Mit ad meg a függvény? - az új, Δt idővel későbbi helyzetet és sebességet

[3]:
```
# Egyszerű, józan eszes léptetés: x és v értékek számítása gyorsulás alapján
# F(xt,vt,m)

def lepes(xt, vt, dt, F, m):

    at = F(xt, vt, m) / m       # gyorsulás mozgásegyenletből
    v_new = vt + at * dt        # új sebesség
    x_new = xt + v_new * dt     # új helyzet
    return (x_new , v_new)
```

[5]:
```
# Néhány eredő erő erőfüggvénye
# Alakra megfelel a léptetésnél használt erőfüggvény alaknak

# F(xt,vt,m)

# 1D rugóerő ------------------------------------
def F_rugo(x, v, m):
    D = 50.0
    return -D*x

# szabadesés közegellenállással -------------------
def F_eses(x, v, m):
    g = 9.81  # grav gyorsulás
    C = 0.47  # alakténye ző
    A = 0.01  # keresztmetszet mozgásirányban
    rho = 1.2 # közeg sűrűség
    F = m * g
    if v > 0:
        F -= 0.5 * C * A * rho * v * abs(v)
    return F
```

9.3 Harmonikus rezgőmozgás (F_rugo)

[17]:
```
# kezdőértékek: kezdeti hely, sebesség ...
x0 = 0.0     # kezdeti hely
v0 = 2.0     # kezdő sebesség
m = 16.0     # tömeg
dt = 0.01    # időlépés
t_max = 5.0  # végső időpont
t0 = 0.0     # kezdeti időpont

# változók inicializálása
t, x, v = t0, x0, v0

# Mi legyen az erő függvény?
```

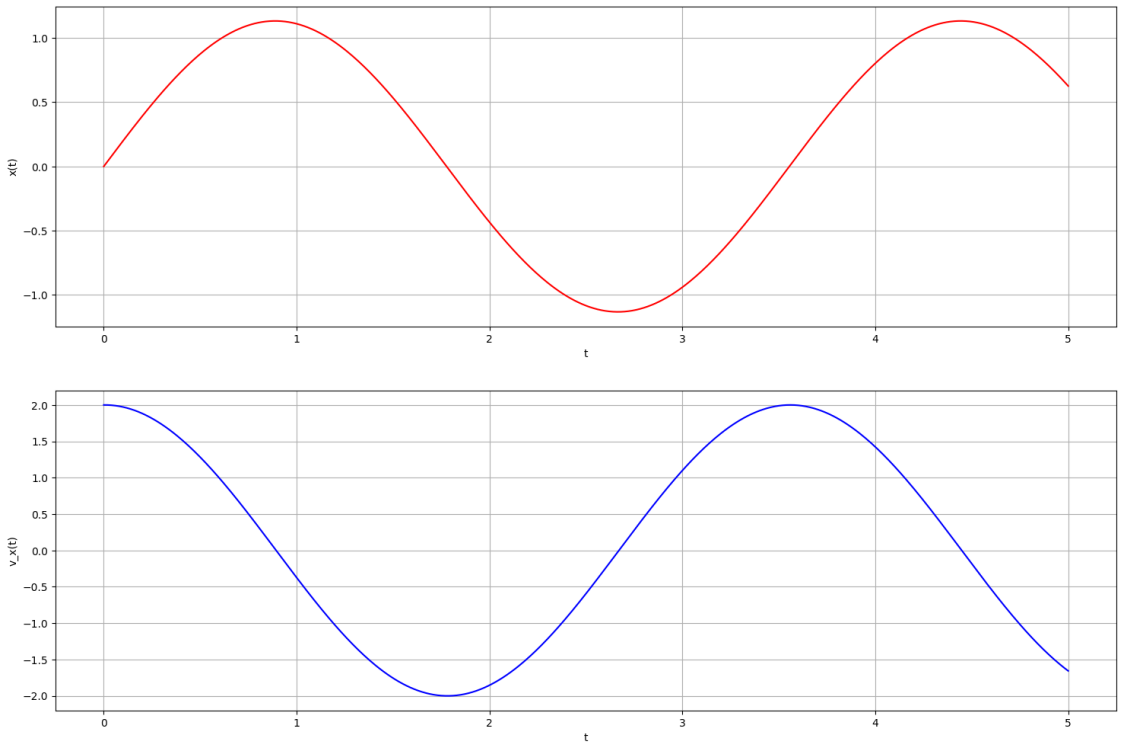
```
F = F_rugo
```

[18]:
```
# listákba gyűjtjük az eredményt
t_list = []
x_list = []
v_list = []
```

[19]:
```
# tényleges futtatás CIKLUS
while (t <= t_max):
    x_list.append(x)
    v_list.append(v)
    x, v = lepes(x, v, dt, F, m)      #hely és sebesség léptetése F_eredo␣
 ↪erőfüggvénnyel

    t_list.append(t)
    t += dt                             # idő léptetése
```

[20]:
```
# most kirajzoljuk: egy ábrán több grafikon
fig = plt.figure(figsize=(18,12))
# két rész-grafikon
ax1 = fig.add_subplot(211)
ax2 = fig.add_subplot(212)
ax1.plot(t_list, x_list, color = "red") # az elsőbe az rx(t)
ax2.plot(t_list, v_list, color = "blue") # a másodikba a vx(t)
# grafikonok testreszabása
ax1.grid()
ax2.grid()
ax1.set_xlabel("t")
ax1.set_ylabel("x(t)")
ax2.set_xlabel("t")
ax2.set_ylabel("v_x(t)")
plt.show()
```



9.4 Szervezzük ki ezt az munkafolyamatot függvények segítségével

- Egy függvény, ami elvégzi a számolást (eredménye az idő, hely sebesség adatok)
- És egy másik ami listák/tömbök alapján ábrázol

9.4.1 Számoló függvény

Mik a számoló függvény paraméterei? - kezdeti hely,kezdő sebesség - időlépés, végső időpont - tömeg - használandó erőfüggvény (Előző F_rugo helyett tetszőleges F)

Mi legyen a visszatérési érték? - idő adatok - hely adatok - sebesség adatok

[21]:
```
def data_x_v(x0, v0 ,dt ,t_max, m, F):
    t0 = 0.0

    # változók inicializálása
    t,x,v = t0, x0, v0

    # listákba gyűjtjük az eredményt
    t_list=[]
    x_list=[]
    v_list=[]

    # tényleges futtatás CIKLUS
```

```
    while (t <= t_max):
        x_list.append(x)
        v_list.append(v)
        x , v = lepes(x,v,dt,F,m)       # tetszőleges F függvény

        t_list.append(t)
        t += dt

    # tömbbé alakítjuk az eredményt
    x_arr=np.asarray(x_list)
    v_arr=np.asarray(v_list)
    t_arr=np.asarray(t_list)           # legyen az idő is tömbben

    return t_arr, x_arr, v_arr
```

9.4.2 Ábrázoló függvény

Mik az ábrázoló függvény paraméterei? - idő adatok - hely adatok - sebesség adatok

Mi legyen a visszatérési érték? - semmi - "mellékahtásként" készüljön el az ábra

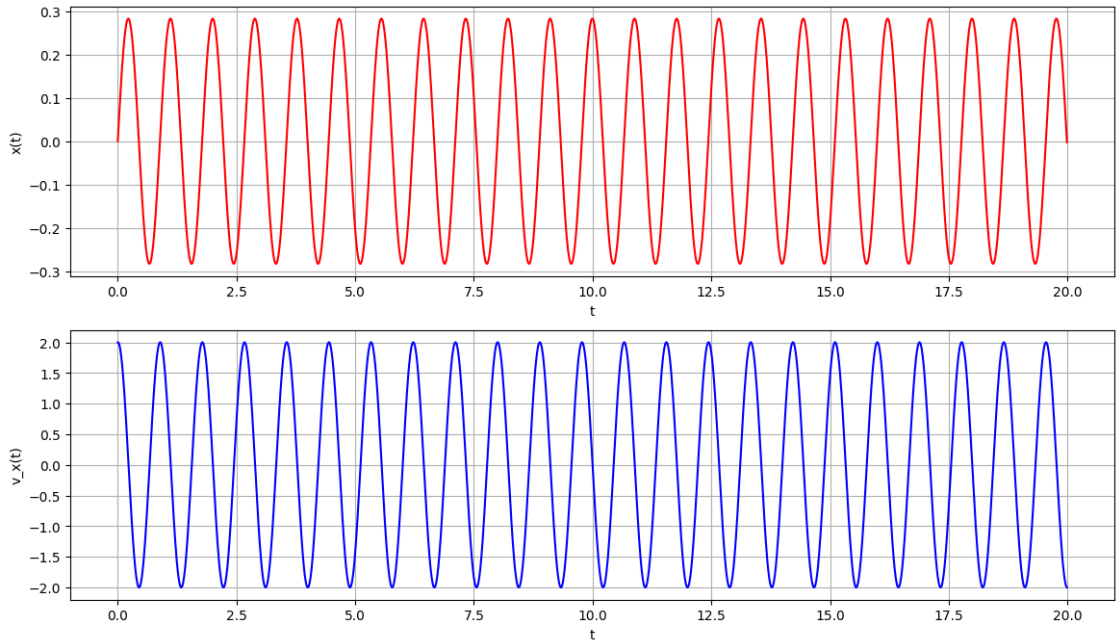
[22]:
```
def abra_1D_mozgas(t_data,x_data,v_data):
    # most kirajzoljuk: egy ábrán több grafikon
    fig=plt.figure(figsize=(14,8))
    # két rész-grafikon
    ax1=fig.add_subplot(211)
    ax2=fig.add_subplot(212)
    ax1.plot(t_data, x_data, color="red") # az elsőbe az rx(t)
    ax2.plot(t_data, v_data, color="blue") # a másodikba a vx(t)
    # grafikonok testreszabása
    ax1.grid()
    ax2.grid()
    ax1.set_xlabel("t")
    ax1.set_ylabel("x(t)")
    ax2.set_xlabel("t")
    ax2.set_ylabel("v_x(t)")
    plt.show()
```

9.4.3 Harmonikus rezgőmozgás újra

[23]:
```
x0 = 0.0     # kezdeti hely
v0 = 2.0     # kezdő sebesség
m = 1.0     # tömeg
dt = 0.01    # időlépés
t_max = 20.0  # végső időpont

t_data,x_data,v_data = data_x_v(x0,v0,dt,t_max, m, F_rugo)
```

```
abra_1D_mozgas(t_data,x_data,v_data)
```



9.4.4 Szabadesés közegellenállással (F_eses)

Kellően hosszú idő alatt a sebesség állandósul

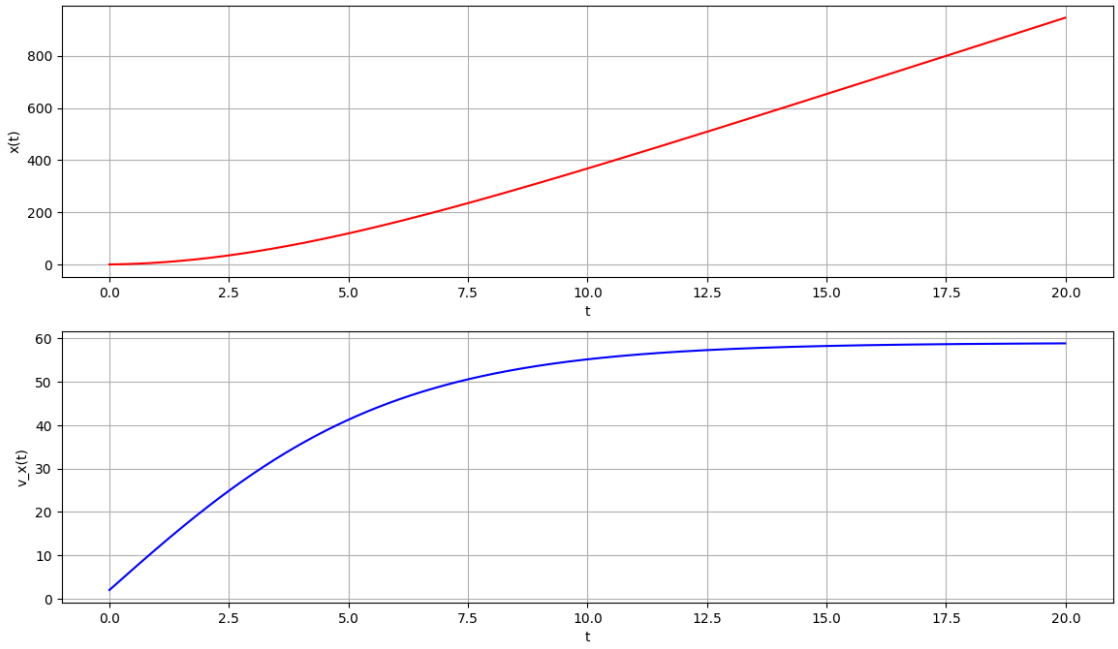
- De mikor következik ez be adott paraméterek esetén?
- Mekkora magasságból esve állandosul a sebesség?

Játszunk a paraméterekkel: Mi történik, ha a tömeget változtatom?

[24]:
```
x0 = 0.0      # kezdeti hely
v0 = 2.0      # kezdő sebesség
m = 1.0      # tömeg
dt = 0.01     # időlépés
t_max = 20.0  # végső időpont

t_data, x_data, v_data = data_x_v(x0,v0,dt,t_max, m, F_eses)

abra_1D_mozgas(t_data,x_data,v_data)
```

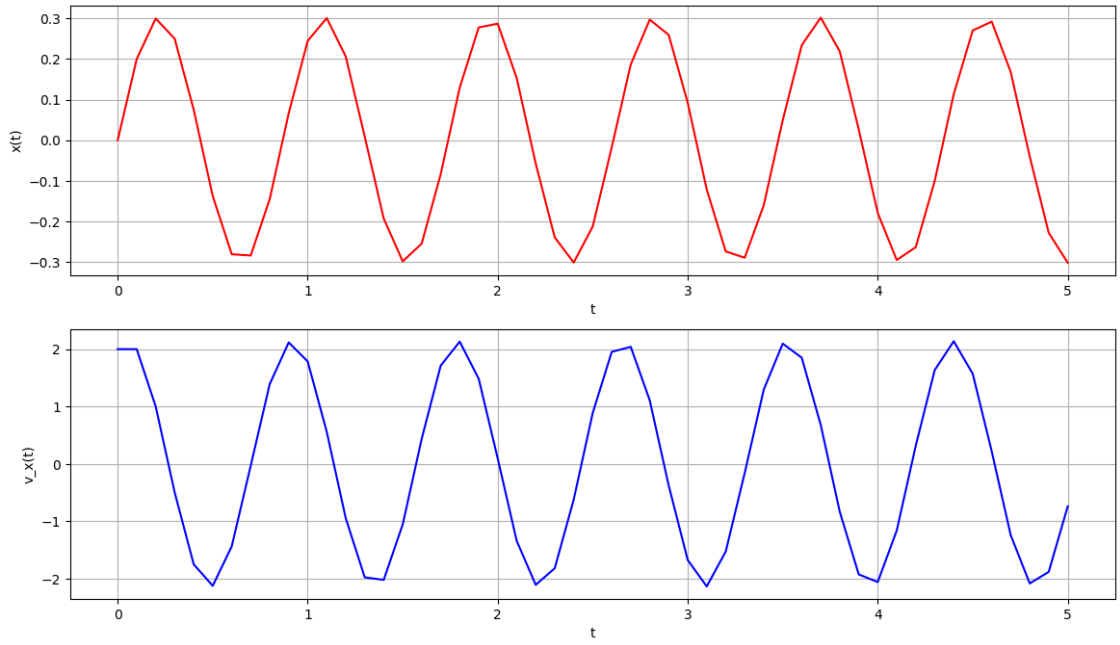


9.5 Harmonikus rezgőmozgás

Játszunk a paraméterekkel Mi történik ha a tömeget változtatom? Vagy ha az időlépés dt túl nagy?

[26]:
```
x0 = 0.0      # kezdeti hely
v0 = 2.0      # kezdő sebesség
m = 1.0      # tömeg
dt = 0.1     # időlépés
t_max = 5.0  # végső időpont

t_data,x_data,v_data = data_x_v(x0,v0,dt,t_max, m, F_rugo)
abra_1D_mozgas(t_data,x_data,v_data)
```



9.5.1 mekkora a rezgés amplitúdója, maximális sebessége?

[27]:
```
x_data.max()   # amplitudo
```

[27]:
```
0.30215752643125615
```

[28]:
```
x_data.min()
```

[28]:
```
-0.30236091873805915
```

[29]:
```
v_data.max()  # maximális sebesség
```

[29]:
```
2.137523784024098
```

[30]:
```
# szebben kiíratva:
print(f"A rezgés amplitúdója: {x_data.max():0.2f} m")
print(f"A rezgés maximális sebessége: {v_data.max():0.2f} m/s")
```

```
A rezgés amplitúdója: 0.30 m
A rezgés maximális sebessége: 2.14 m/s
```

9.5.2 Mekkora volt a periódusidő?

2 szomszédos maximumhely időbeli távolsága

De hol vannak a maximum helyek?

Ott maximális egy függvény értéke, ahol előtte nőtt, utána pedig csökkent. Azaz x_i-ben helyi maximum van, ha: $x_i > x_{i-1}$ és $x_i > x_{i+1}$.

[31]:
```
# x helyi maximumainak kiválasztása
x_loc_max = np.zeros(x_data.shape, dtype=bool)  # x_arr-ral egy méretű bool␣
 ↪(logikai) tömb foglalása

# legyen az érték nagyobb, mint az előtte és a utána levő is:
x_loc_max[1:-1] = (x_data[1:-1]>x_data[0:-2])  & (x_data[1:-1] > x_data[2:])
# x_loc_max: ott True, ahol helyi maximum van

# maximumok időpontjai
t_loc_max = t_data[x_loc_max]

print("Maximumhelyek:", t_loc_max)
```

```
Maximumhelyek: [0.2 1.1 2.  2.8 3.7 4.6]
```

[32]:
```
# első két max különbsége:
print(f"Első két maximum különbsége: {t_loc_max[1]-t_loc_max[0]:0.7f} s")
```

```
Első két maximum különbsége: 0.9000000 s
```

[33]:
```
# tényleg változatlan a periódusidő?
print("Szomszédos maximumok különbsége:",t_loc_max[1:]-t_loc_max[0:-1])
```

```
Szomszédos maximumok különbsége: [0.9 0.9 0.8 0.9 0.9]
```

[34]:
```
# pontosabb periódusidő:
T=(t_loc_max[-1]-t_loc_max[0])/(len(t_loc_max)-1)
print(f"Pontosabb periódusidő:{T:0.7f} s")
```

```
Pontosabb periódusidő:0.8800000 s
```

9.6 Csillapodó rezgőmozgás

[36]:
```
# 1D rugó, lineáris közegellenállás

def F_rugo_kozeg_lin(r,v,m):
    C_lin = 1.0   #   F_közeg= -C_lin*v
    F=F_rugo(r,v,m) - C_lin*v
    return F


# 1D rugó, négyzetes közegellenállás

def F_rugo_kozeg_negyz(r,v,m):
    C_negyz=0.47
```

```
    A = 0.5
    rho = 1.2
    F=F_rugo(r,v,m) - 0.5*C_negyz*A*rho*v*np.abs(v)
    return F


# 1D rugó, súrlódásos fékeződés

def F_rugo_surl(r,v,m):
    mu  = 0.15
    g = 9.81
    if np.abs(v)<1e-10:
        F=F_rugo(r,v,m)
    else:
        F=F_rugo(r,v,m) - mu*m*g*v/(np.abs(v))
    return F
```

[37]:
```
x0 = 0.0      # kezdeti hely
v0 = 2.0      # kezdő sebesség
m = 5.0      # tömeg
dt = 0.01     # időlépés
t_max = 20.0  # végső időpont

t_data,x_data,v_data = data_x_v(x0,v0,dt,t_max, m, F_rugo_kozeg_lin)
abra_1D_mozgas(t_data,x_data,v_data)
```