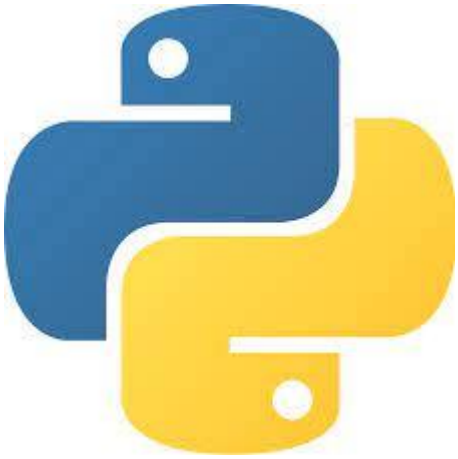
Bevezető gyakorlatok
az emelt szintű informatika érettségi
programozási feladataihoz
Python nyelven



Klement András

2022-23

Utolsó módosítás: 2023. 5. 14.

Naplemente

Az arany nap lankadva ballag, parazsa hamvad.

Tűzgolyónk süllyedő útja nyírfák mögé bukik.

Meleg est lehelete terjed szerte felettem.

A hűsítő tó fölé ezüst holdsugár kúszik.

Hatalmas vadalmafa alatt

csendesen mereng egy delejes

tündérszárnyú kígyóbűvölő.

Körmend, 2018. február 11.

Klement András

Tartalomjegyzék

1. Algoritmikus alapszerkezetek: elágazások és ciklusok

```
"""
1. Algoritmikus alapszerkezetek: elágazások és ciklusok
@author Klemand66
"""

"""
Telepítés, fejlesztői környezet
Mielőtt bármit elkezdenénk, telepítsük gépünkre a Python legújabb változatát.
A telepítőprogram a https://www.python.org/downloads/ oldalról tölthető le.
Kattints a Download the latest version for Windows alatti gombra.
Nálam a Donwload Python 3.10.5 volt, a továbbiakban erre fogok hivatkozni.
Célszerű a C:/Python/ mappába menteni a python-3.10.5-amd64.exe telepítőfájlt.
Futtatáskor ne felejtsd el kipipálni az Add Python 3.10 to PATH kijelölőnégyzetet!

Telepítés után a Start menüben a Python 3.10 mappában az első helyen van az IDLE 3.10 64 bit,
az integrált fejlesztői környezetet, amit célszerű a Start menüben vagy a tálcán rögzíteni.
Ebben fogjuk írni és futtatni a programjainkat. Indításakor megnyílik az IDLE Shell 3.10.5 ablak.
Először egy új fájlt kérünk, amit a C:/Python valamely almappájába mentünk,
ahonnan később a munka folytatásához meg is tudjuk nyitni.
Ha kész vagy tesztelni akarjuk, mentsük el és a RUN menüből vagy az F5-tel futtassuk.

Megjegyzés: Futtatni a py kiterjesztésű programjainkat a fájlkezelőből is lehet
dupla kattintással is, de akkor futás után rögtön bezáródik a programablak,
ezért minden program végén kérni fogjuk az ENTER lenyomását.

A hibák könnyebb azonosításához célszerű bekapcsolni az Options menüben a sorszámok mutatását.
"""

from random import *
from math import *

""" többsoros megjegyzés kezdete, ill. vége
# egysoros megjegyzés kezdete, vége nincs

Azokat az utasításokat, függvényeket, melyek nem tartoznak a Python alapkészletéhez,
szokás szerint a program elején importálnunk kell.
Importálhatjuk a modul egyetlen elemét, pl.
from random import randrange
A range(7) függvény az egész számok sorozatát állítja elő: 0,1,2,3,4,5,6. A 7 már nincs benne!
A randrange(7) értelemszerűen egy véletlenszámot ad meg ebből a sorozatból, pl. 4-et.

Ha nem akarjuk a modul minden szükséges elemét külön-külön importálni, importáljuk az egész modult.
Ezt kétféleképpen tehetjük meg:
1. from random import *
Itt a * jelenti azt, hogy a modul minden elemére vonatkozik az importálás.
Ekkor a modul nevét nem használjuk az elemek előtt: randrange(7)
2. import random
Ez az egész modult importálja, de használatkor az elemek előtt ki kell írnunk a modul nevét is:
random.randrange(7)

Mi az első módszert fogjuk használni. Ekkor nem kell azon gondolkodni,
hogy pl. az abs() függvény meghívásakor elé kell-e írnunk a math modul nevét: math.abs()
Pythonban ugyanis nem kell, sőt nem is szabad, mert része az alapkészletnek,
míg pl. a Javaban kellett hozzá a math modul.
"""


print("\nAlgoritmikus alapszerkezetek: elágazások és ciklusok\n")
"""
A print() utasítással tudunk a monitorra írni.
A szöveget idézőjelek vagy aposztrófok közé kell tenni, több elem közé vesszőt teszünk.
Ezek között a szóköz az alapértelmezett elválasztójel, amit meg tudunk változtatni:
print(a,b,sep="tetszőleges karaktersorozat")
Alapértelmezés a kiírás végén a soremelés, ezt így módosíthatjuk: print(a,b,end="")

\n a szövegbe illesztve sortörés, \t pedig tabulátor,
A \ jelet funkcióval rendelkező karakterek megjelenítésére is használjuk,
pl. \" az idézőjel, \> a nagyobb jel megjelenítése.
"""
```

```
print("1. Feltételes utasítás függvényhívással\n")

print("Egy kockadobást szimulálunk. Ha hatos lesz, nyertél!\n")

"""
A véletlen szám előállítása nem tartozik a Python alap utasításkészletéhez,
ezért használatához a program elején importáljuk a random modul minden elemét.

A program utasításait mindig a sor elején kell kezdenünk.

Az összetett utasításokat (definíciók, elágazások, ciklusok)
ugyanolyan szerkezetű utasításblokkokban kell megadnunk:
Fejsor: (Mindig egy kulcsszóval kezdődik és kettősponttal végződik.)
    utasítás (pontosan egy tabulátornyi, azaz 4 szóköznyi behúzással)
    ...
    blokkzáró utasítás

Az utasításblokkokat egymásba is ágyazhatjuk:
Fejsor:
    utasítás
    ...
    Fejsor:
        utasítás
        ...
        blokkzáró utasítás
    ...
    blokkzáró utasítás

A dobáshoz függvényt definiálunk.
A függvények és eljárások definícióit a főprogram elé tesszük.
Változók, függvény- és eljárásnevek esetén soha nem használok ékezeteket.

A függvény szerkezete:
def a_függvény_neve(esetleges paraméterek vesszővel elválasztva):
    utasítás
    ...
    utasítás
    return visszatérő_érték (Elágazás esetén minden ágban szerepelnie kell!)
"""

def kockadobas(): # A zárójel nem hagyható el akkor sem, ha nincs paraméter!
    dobas=randrange(1,7) # [1,7)-beli egész;
    # a randrange(1,7,2) értéke 1,3,5 lehetne
    # más megoldás: dobas=int(6*random())+1 , ahol random() [0,1)-beli valós szám
    return dobas # a return után adjuk meg a függvény visszatérő értékét

"""
Ugyanígy definiálhatunk eljárást is, csak annak nincs visszatérő értéke.
Adhatunk neki is bemenő paramétert.

A kiíratásokat a 2016-os Python 3.6-ban bevezetett f-string tette egyszerűvé és áttekinthetővé.
Az f-stringek segítségével a kiíratások szövegébe tetszőleges Python-kódot illeszthetünk be.
Használata csak a 2016 utáni érettségi megoldásokban jelent meg,
mi minden összetett kiíratásban már ezt fogjuk használni.
A sima szövegtől való megkülönböztetésül " helyett '-t fogunk használni.
A változókat kapcsos zárójelbe tesszük.
"""

def vonal(m1,m2,m3):
    print(f'\n==========={m1}==========={m2}==========={m3}==========\n')

k=kockadobas() # A k változó a kockadobás() függvény által visszaadott értéket veszi fel.

print(f'A dobás értéke: {k}. ')

"""
Az egyszerű feltétel szerkezete:
if feltétel:
    a feltétel teljesülése esetén végrehajtandó utasítás
    ...
    a feltétel teljesülése esetén végrehajtandó utasítás
"""

if k==6: # k=6: értékadás, k==6: az egyenlőség vizsgálata
    print("Nyertél!")

vonal("","  PYTHON 3.10.5 ","")
```

```
print("2. Kétirányú elágazás \n")
print("Újra feldobjuk a kockánkat.\n")

k=kockadobas() # A k változó korábbi értékét felülírja az új értékadás.
print(f'A dobás értéke: {k}. ')

"""
A kétágú feltétel szerkezete:
if feltétel:
    a feltétel teljesülése esetén végrehajtandó utasítás
    ...
    a feltétel teljesülése esetén végrehajtandó utasítás
else:
    a feltétel nem teljesülése esetén végrehajtandó utasítás
    ...
    a feltétel nem teljesülése esetén végrehajtandó utasítás
"""

if k==6:
    print("Ne bízd el magad! Egyszer fenn, máskor lenn.")
else:
    print("Ne csüggedj! A szerencse forgandó.")

vonal(" "," "," ")


print("3. Többirányú elágazás \n")
print("Újra feldobjuk a kockánkat.\n")

k=kockadobas()
print(f'A dobás értéke: {k}. ')

"""
A többágú feltétel szerkezete:
if feltétel:
    a feltétel teljesülése esetén végrehajtandó utasítás
    ...
    a feltétel teljesülése esetén végrehajtandó utasítás
elif feltétel:
    végrehajtandó utasítás, ha a korábbi feltétel nem teljesül, de ez igen
    ...
    végrehajtandó utasítás, ha a korábbi feltétel nem teljesül, de ez igen
...
elif feltétel:
    végrehajtandó utasítás, ha a korábbi feltételek egyike sem teljesül, de ez igen
    ...
    végrehajtandó utasítás, ha a korábbi feltételek egyike sem teljesül, de ez igen
...
else:
    végrehajtandó utasítás, ha egyetlen feltétel sem teljesül
    ...
    végrehajtandó utasítás, ha egyetlen feltétel sem teljesül
"""

if k==6:
    print("Mindig törekedj a maximumra, de maradj szerény!")
elif k==4 or k==5: # Az or operátor a vagy logikai művelet.
    print(f'Csak {6-k} hiányzott!')
elif k<4 and k%2==1: # Az and operátor az és logikai művelet,
                     # a % operátor a maradékos osztás maradékát adja meg (itt: páratlan-e)
    print("Kicsi a bors, de erős!")
else:
    print(f'Ez igen! A {k} az egyetlen páros prímszám!')

karakter=str(k)
# Az str() számot szöveggé alakít, az int() szöveget egész számmá,
# a float() pedig egy tizedesponttal megadott stringet valós számmá.

vonal(" "+karakter+" "," <<< dobás & ASCII kódja >>> "," "+str(ord(karakter))+" ")
# Az ord() egy karakter ASCII-kódját adja meg, a chr() a kódhoz tartozó karaktert.
```

```
print("4. For-ciklus\n")

"""
Szerkezete:
for i in range(kezdőérték,kilépési érték,lépésköz):
    ciklusmag utasítás
    ...
    ciklusmag utasítás

A range() egy számsorozatot határoz meg, csak a kilépési érték kötelező.
Ha nem adunk meg kezdőértéket, az automatikusan 0 lesz, negatív is lehet.
Ha nem adunk meg lépésközt, az automatikusan 1 lesz.
i a ciklusváltozó, ami a definiált sorozat minden tagját bejárja.
"""

print("Az első 12 egész szám köbe, egész harmada, négyzetének utolsó jegye, gyöke\n")

"""
A gyök függvényhez a math modult is importálnunk kell, amit kétféleképpen tehetünk meg:
 - import math estén a math.sqrt() alakot kellene használnunk,
 - from math import * esetén viszont csak az sqrt() kell!
"""

for i in range(1,13):
    print(f'{i:2} köbe: {i**3:4}, egész harmada: {i//3:1}, \
négyzetének utolsó jegye: {i**2%10:1}, gyöke: {sqrt(i):6.4f}')
    #Hosszú kódsorokat így tördelhetünk. A \ után megjegyzés sem kerülhet a sorba!
    # {:2}: szélesség=2,  **: hatványozás  //: egészosztás,
    # %: maradék, sqrt(): gyök,  {:6.4f}: szélesség=6  tiedesjegy=4
print()
vonal("==========","  O  ","==========")


print("5. Egymásba ágyazott for-ciklusok\n")

"""
Szerkezete:
for i in range(kezdőérték,kilépési érték,lépésköz):
    ciklusmag utasítás
    ...
    for j in range(kezdőérték,kilépési érték,lépésköz):
        ciklusmag utasítás
        ...
        ciklusmag utasítás
    ...
    ciklusmag utasítás


i a külső, j a belső ciklusváltozó.
A külső ciklus minden egyes lépésénél a teljes belső ciklust bejárja,
azaz i változik lassan, j gyorsan.
"""

print("Rajzolunk\n")

for i in range(1,15):
    print("xxxxxx----------------------------------------", end="")
    # Letiltjuk a soremelést.
    for j in range(1,i+1):
        print("+", end="")
    print()

# Most visszafelé:
for i in range(13,0,-1):
    print("xxxxxx----------------------------------------", end="")
    for j in range(1,i+1):
        print("+", end="")
    print()

vonal("  ","  ","  ")
```

```
print("6. Szorzótábla\n")

print("  *\t",end="")
for i in range(1,11):
    print(f'{i:3}\t',end="")
print("\n")

for i in range(1,11):
    print(f'{i:3}\t',end="")
    for j in range(1,11):
        print(f'{i*j:3}\t',end="")
    print()


vonal(" 1x1 "," 5x6 "," 7x8 ")


print("Elöltesztelő ciklus\n")

"""
Szerkezete:

A ciklusváltozó kezdőértékének megadása
while belépési_feltétel:
    ciklusmag utasítás
    ...
    ciklusváltozó értékének módosítása
    ...

Addig lép be a ciklusmagba, amíg teljesül a belépési feltétel,
lehet, hogy egyszer sem.
"""

print("7. A ciklusváltozó szabályosan változik\n")

print("Addig íratjuk ki a számok négyzetét, amíg el nem éri a 100-at \n")

i=1 # A ciklusváltozó az i, ami ugyan szabályosan változik,
    # de a belépés nem i-től, hanem egy kifejezésétől függ.
while i**2<100:
    print(f'{i:2}^2 = {i**2:2}')
    i+=1 # i=i+1, az i változó memóriadobozában lévő számot eggyel nagyobbra cseréljük.

vonal(" 1 "," 4 "," 9 ")


print("8. A ciklusváltozó szabálytalanul változik\n")

print("[-0,5,05)-beli véletlenszámokból vonunk gyököt,");
print("mindaddig, amíg nemnegatív számot állít elő a függvény:\n");
# A ciklusmag a gyökvonást tartalmazza, de lehet, hogy egyszer sem tudunk belépni.
# Megszámolhatjuk, hány sikeres belépés lesz.

sdb=0
v=random()-0.5 # A v ciklusváltozó itt teljesen szabálytalanul változik.
while v>=0:
    print(f'A véletlenszám: {v:.4f}, a gyöke: {sqrt(v):.4f}')
    sdb+=1 # Ez a számlálás nincs befolyással a belépésre, csak plusz információt eredményez.
    v=random()-0.5 # Itt módosul a ciklusváltozó, ami eldönti a következő belépést.

print(f'\nA sikeres belépések száma: {sdb}')
print(f'Megjegyzés: Az első nem megfelelő véletlenszám a {v:.4f} volt.')

vonal(" 0.0000 "," 0.5000 "," 0.7071 ")
```

```
print("9. A ciklusváltozó értékének megadása és módosítása a belépési feltételben\n")

print("Egy társasjátékban kockadobással döntjük el a továbblépést.")
print("Ha legalább 3-ast dobunk, annyit léphetünk előre és újat is dobhatunk.\n")

ldb=0 # Megszámoljuk, hányat léphetünk egy dobássorozatban.
while (k := kockadobas()) >= 3: # A feltételben lévő := értékadást a Python 3.8-ban vezették be.
                                # Zárójel nélkül logikai értéket ad!
    ldb+=k
    print(f'Sikeres dobás, {k} lépést tehetünk előre.')

print(f'\nÖsszesen {ldb} lépést tehettünk meg.')
print(f'Megjegyzés: Az első sikertelen dobás {k} volt.')

vonal(" 3 "," 5 "," 6 ")


print("Hátultesztelő ciklus \n");

"""
A Pythonban nincs hagyományos hátultesztelő ciklus.
Lényege, hogy a ciklusmagba egyszer mindenképpen be kell lépni,
és a ciklusmag végrehajtása után vizsgáljuk a bentmaradási/kilépési feltételt.

A 8. feladatban is előállítottunk legalább egy véletlen számot,
de még a ciklusmagba való belépés előtt vizsgáltuk, hogy lehet-e belőle gyököt vonni.
Értelmetlen is lett volna mindenképpen erőltetni a gyökvonást.

A problémát a következő "pythonos" szerkezettel szokták megoldani:

while True:
    ciklusmag utasítás
    ...
    if kilépési_feltétel:
        break
    ciklusmag utasítás
"""


print("10. Ismét [-0,5,05)-beli véletlenszámokból vonunk gyököt,");
print("de a while True ciklussal, mindaddig, amíg folyamatosan megtehetjük.\n");

sdb=0
while True: # Végtelen ciklust indít, amiből a break utasítással tudunk kiugrani.
            # Csak a True alak jó, nem lehet TRUE vagy true
    v=random()-0.5
    if v<0:
        break # Párja a continue, ami csak a ciklusmag hátralevő részét ugorja át,
              # tehát csak kihagyja az aktuális esetet.
    print(f'A véletlenszám: {v:.4f}, a gyöke: {sqrt(v):.4f}')
    sdb+=1

print(f'\nA sikeres belépések száma: {sdb}')
print(f'Megjegyzés: Az első nem megfelelő véletlenszám a {v:.4f} volt.')

vonal(" 0.0000 "," 0.5000 "," 0.7071 ")


print("11. Most addig dobjuk fel a kockánkat, amíg nem sikerül hatost dobni.");
print("Vajon hányadikra fog sikerülni?\n");

pdb=1 # A próbálkozások számát a ciklusmagban csak sikertelen dobás esetén növeljük.
      # Egy próbát mindenképpen kell tennünk.
while True:
    k = kockadobas()
    if k==6:
        break
    print(f'Sikertelen próbákozás: {k}')
    pdb+=1

print(f'\nBravó! A {pdb}. próbálkozásod sikerrel járt!')

vonal("," 6 ","")

input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak01\BevGyak01.py ==========================

Algoritmikus alapszerkezetek: elágazások és ciklusok

1. Feltételes utasítás függvényhívással

Egy kockadobást szimulálunk. Ha hatos lesz, nyertél!

A dobás értéke: 3.

==================== PYTHON 3.10.5 ====================

2. Kétirányú elágazás

Újra feldobjuk a kockánkat.

A dobás értéke: 5.
Ne csüggedj! A szerencse forgandó.

========== ========== ========== ==========

3. Többirányú elágazás

Újra feldobjuk a kockánkat.

A dobás értéke: 5.
Csak 1 hiányzott!

========== 5 ========== <<< dobás & ASCII kódja >>> ========== 53 ==========

4. For-ciklus

Az első 12 egész szám köbe, egész harmada, négyzetének utolsó jegye, gyöke

 1 köbe:    1, egész harmada: 0, négyzetének utolsó jegye: 1, gyöke: 1.0000
 2 köbe:    8, egész harmada: 0, négyzetének utolsó jegye: 4, gyöke: 1.4142
 3 köbe:   27, egész harmada: 1, négyzetének utolsó jegye: 9, gyöke: 1.7321
 4 köbe:   64, egész harmada: 1, négyzetének utolsó jegye: 6, gyöke: 2.0000
 5 köbe:  125, egész harmada: 1, négyzetének utolsó jegye: 5, gyöke: 2.2361
 6 köbe:  216, egész harmada: 2, négyzetének utolsó jegye: 6, gyöke: 2.4495
 7 köbe:  343, egész harmada: 2, négyzetének utolsó jegye: 9, gyöke: 2.6458
 8 köbe:  512, egész harmada: 2, négyzetének utolsó jegye: 4, gyöke: 2.8284
 9 köbe:  729, egész harmada: 3, négyzetének utolsó jegye: 1, gyöke: 3.0000
10 köbe: 1000, egész harmada: 3, négyzetének utolsó jegye: 0, gyöke: 3.1623
11 köbe: 1331, egész harmada: 3, négyzetének utolsó jegye: 1, gyöke: 3.3166
12 köbe: 1728, egész harmada: 4, négyzetének utolsó jegye: 4, gyöke: 3.4641


============================== O ==============================
```

5. Egymásba ágyazott for-ciklusok

Rajzolunk

```
xxxxxx----------------------------------------+
xxxxxx----------------------------------------++
xxxxxx----------------------------------------+++
xxxxxx----------------------------------------++++
xxxxxx----------------------------------------+++++
xxxxxx----------------------------------------++++++
xxxxxx----------------------------------------+++++++
xxxxxx----------------------------------------++++++++
xxxxxx----------------------------------------+++++++++
xxxxxx----------------------------------------++++++++++
xxxxxx----------------------------------------+++++++++++
xxxxxx----------------------------------------++++++++++++
xxxxxx----------------------------------------+++++++++++++
xxxxxx----------------------------------------++++++++++++
xxxxxx----------------------------------------+++++++++++
xxxxxx----------------------------------------++++++++++
xxxxxx----------------------------------------+++++++++
xxxxxx----------------------------------------++++++++
xxxxxx----------------------------------------+++++++
xxxxxx----------------------------------------++++++
xxxxxx----------------------------------------+++++
xxxxxx----------------------------------------++++
xxxxxx----------------------------------------+++
xxxxxx----------------------------------------++
xxxxxx----------------------------------------+

==========  ==========  ==========  ==========
```

6. Szorzótábla

*	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

========== 1x1 ========== 5x6 ========== 7x8 ==========

```
Elöltesztelő ciklus

7. A ciklusváltozó szabályosan változik

Addig íratjuk ki a számok négyzetét, amíg el nem éri a 100-at

 1^2 =  1
 2^2 =  4
 3^2 =  9
 4^2 = 16
 5^2 = 25
 6^2 = 36
 7^2 = 49
 8^2 = 64
 9^2 = 81

========== 1 ========== 4 ========== 9 ==========

8. A ciklusváltozó szabálytalanul változik

[-0,5,05)-beli véletlenszámokból vonunk gyököt,
mindaddig, amíg nemnegatív számot állít elő a függvény:

A véletlenszám: 0.4980, a gyöke: 0.7057
A véletlenszám: 0.1823, a gyöke: 0.4270
A véletlenszám: 0.0297, a gyöke: 0.1724
A véletlenszám: 0.3738, a gyöke: 0.6114
A véletlenszám: 0.2239, a gyöke: 0.4732

A sikeres belépések száma: 5
Megjegyzés: Az első nem megfelelő véletlenszám a -0.4502 volt.

========== 0.0000 ========== 0.5000 ========== 0.7071 ==========

9. A ciklusváltozó értékének megadása és módosítása a belépési feltételben

Egy társasjátékban kockadobással döntjük el a továbblépést.
Ha legalább 3-ast dobunk, annyit léphetünk előre és újat is dobhatunk.


Összesen 0 lépést tehettünk meg.
Megjegyzés: Az első sikertelen dobás 2 volt.

========== 3 ========== 5 ========== 6 ==========
```

```
Hátultesztelő ciklus

10. Ismét [-0,5,05)-beli véletlenszámokból vonunk gyököt,
de a while True ciklussal, mindaddig, amíg folyamatosan megtehetjük.

A véletlenszám: 0.2110, a gyöke: 0.4594
A véletlenszám: 0.4167, a gyöke: 0.6455
A véletlenszám: 0.4819, a gyöke: 0.6942
A véletlenszám: 0.1933, a gyöke: 0.4397

A sikeres belépések száma: 4
Megjegyzés: Az első nem megfelelő véletlenszám a -0.0066 volt.

========== 0.0000 ========== 0.5000 ========== 0.7071 ==========

11. Most addig dobjuk fel a kockánkat, amíg nem sikerül hatost dobni.
Vajon hányadikra fog sikerülni?


Bravó! A 1. próbálkozásod sikerrel járt!

==================== 6 ====================

A befejezéshez nyomd meg az ENTER billentyűt!
```

2. Adatszerkezetek: számok, karakterek, szöveg és listák

```
"""
2. Adatszerkezetek: számok, karakterek, szöveg és listák
@author Klemand66
"""

from random import *
from math import *

def cimiro(cim,karakter):
    print("\n",cim,sep="") # Az alapértelmezés sep=" " elcsúszást okozna
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

cimiro("Adatszerkezetek: számok, karakterek, szöveg és listák","*")



cimiro("1. Egész számok","=")

# A Pythonban nincs külön típus a rövid és hosszú egészekre, elvileg bármilyen nagyok lehetnek.

def veletlenegesz(alsohatar,felsohatar): # Beleértjük a felső határt is.
    szam=randrange(alsohatar,felsohatar+1) # A függvény nem érti bele.
    return szam

cimiro("Műveletek két 10 és 100 közötti véletlen egész számmal","-")

m = veletlenegesz(10,100)
n = veletlenegesz(10,100)

nagyobb=max(m,n)
kisebb=min(m,n)

print(f'Az első generált egész szám {m}, a második {n}, {nagyobb} a nagyobb.')
print(f'{m}+{n} = {m+n}, {nagyobb} - {kisebb} = {nagyobb-kisebb}, \
{m}*{n} = {m*n}, {nagyobb}^{kisebb} = {nagyobb**kisebb:.3E}') # tudományos alak 3 tizedesjeggyel

print(f'{nagyobb} // {kisebb} = {nagyobb//kisebb}, \
{nagyobb} mod {kisebb} = {nagyobb%kisebb}, {m} / {n} = {m/n:.4f}') # valós szám 4 tizedesjeggyel


cimiro("Faktoriálisok","-")

i,fakt=1,1 # többszörös értékadás

while fakt<1E+200:
    if i<=20 or i>100:
        if fakt<1E+9:
            print(f'{i:3}! = {fakt:10}') # kiíratás 10 szélességű mezőn (jobbra igazítva)
        else:
            print(f'{i:3}! = {fakt:10.3E}')  # 10 szélességű mező, 3 tizedesjegy
    elif i==21:
        print(" ...")
    i+=1
    fakt*=i

print("\n")
```

```
cimiro("2. Valós számok","=")

def veletlenvalos(alsohatar,felsohatar):
    szam= alsohatar + (felsohatar-alsohatar)*random() # felülről nyitott lesz
    return szam


cimiro("Műveletek két -100 és 100 közötti véletlen valós számmal","-")

a = veletlenvalos(-100,100)
b = veletlenvalos(-100,100)

print(f'Az első generált valós szám {a:3f}, a második {b:.3f}.')

nagyobb=max(a,b)
kisebb=min(a,b)

if kisebb<0:
    print(f'|{kisebb:.3f}| = {abs(kisebb):.3f}')
else:
    print("Egyik sem negatív.")

print(f'{nagyobb:.3f} * {pi:.3f} = {nagyobb*pi:.3f}, {abs(kisebb):.3f}^{e:.3f} = {abs(kisebb)**e:.3E}')
# A pi=3,141592653... és az e=2.718281828... közvetlenül elérhető

print(f'log10({abs(nagyobb):.3f}) = {log10(abs(nagyobb)):.3f}, \
{abs(nagyobb):.3f}^{kisebb:.3f} = {pow(abs(nagyobb),kisebb):.3E}')
# log2() a 2-es alapú, log() az e alapú logaritmus

print("\n")


cimiro("3. Karakterek és sztringek","=")

def megelozi_e(karakter1,karakter2): # logikai függvény
    abc = "aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz"
    if karakter1 in abc and karakter2 in abc:
        return abc.index(karakter1.lower())< abc.index(karakter2.lower())
    else:
        return karakter1 < karakter2

def abc_sorrendben(szo1,szo2): # eljárás, ami meghívja a megelőzi_e() függvényt
    hossz=min(len(szo1),len(szo2))
    szo1_kb=szo1.lower()
    szo2_kb=szo2.lower()
    i=0
    while i<hossz and szo1_kb[i]==szo2_kb[i]:
        i+=1

    if i< hossz:
        if megelozi_e(szo1_kb[i],szo2_kb[i]):
            print(f'{szo1} < {szo2}')
        else:
            print(f'{szo1} > {szo2}')
    else:
        if len(szo1)==len(szo2):
            print(f'{szo1} == {szo2}')
        elif len(szo1)<len(szo2):
            print(f'{szo1} < {szo2}')
        else:
            print(f'{szo1} > {szo2}')

"""
Ameddig a karakterek azonosak a két szóban, csak továbblépkedünk.
Kétféleképpen léphetünk ki a ciklusból:
1. Nem azonos a két szó aktuális karaktere,
    ekkor az előbb lévő karakter szava is előrébb lesz az ábécében.
2. Elfogytak a rövidebb szó karakterei,
    ekkor azonosak vagy a rövidebb előzi meg a hosszabbat.
"""

def maganhangzo_e(karakter):
    mgh = "aáeéiíoóöőuúüű"
    return karakter.lower() in mgh

kar="@"
print(f'A {kar} karakter ASCII-kódja {ord(kar)}\n')
```

```
cimiro("Az ASCII-kódokhoz tartozó karakterek","-")

for i in range(31,371):
    print(f'{i:3}: {chr(i)}\t',end="")
    if (i-30)%10==0:
        print() # 10 oszlopos kiíratás

print("\n")



cimiro('A "tündérszárnyú kígyóbűvölő" karaktereinek ASCII-kódjai',"-")

ekezetes = "tündérszárnyú kígyóbűvölő"

for i in range(len(ekezetes)):
    print(f'   {ekezetes[i]}',end="")
print()

for i in range(len(ekezetes)):
    print(f'{ord(ekezetes[i]):4}',end="")

print("\n")

print("Nagybetűs alakban:\n")

ekezetes_nb=ekezetes.upper()

for i in range(len(ekezetes_nb)):
    print(f'   {ekezetes_nb[i]}',end="")
print()

for i in range(len(ekezetes_nb)):
    print(f'{ord(ekezetes_nb[i]):4}',end="")

print("\n")



cimiro("Műveletek sztringekkel","=")

"""
Szerkezete:
sztring[kezdőkarakter sorszáma:befejező utáni karakter sorszáma: lépésköz]

Ha a kezdőkaraktert elhagyjuk, az első karakter lesz az, 0 a sorszáma.
Ha a befejező karaktert elhagyjuk, az utolsó karakter lesz az, len(szring)-1 vagy csak -1 a sorszáma,
így a sztring[-3: len(sztring)] vagy csak sztring[-3:] a sztring utolsó 3 karakterét adja meg.
Ha a lépésközt elhagyjuk, akkor automatikusan 1 lesz.
"""



cimiro("Rész-sztringek és összefűzésük","-")

print(f'{ekezetes_nb[:6]+ekezetes[-6:]}\n')
print(f'{ekezetes_nb[14:19]+ekezetes[6:12]}\n')
print(f'{ekezetes_nb[-11:-6]+ekezetes[6:12]}\n') # ahonnan rövidebb a számolás
print(f'{ekezetes_nb[-3:len(ekezetes_nb):2]+ekezetes[-3::2]}\n')

print("Az összefűzésben nem lehetnek számok, csak sztringek!\n")

nev="Körmendi Károly"
szulev=1988

adat=nev+": " +str(szulev)+" "
print(adat)
adat=3*adat # A szorzás többszörözi a sztringet.
print(adat)
print("\n")
```

```
cimiro("Sztring megfordítása","-")

print(ekezetes)

ekezetes_ford=""

for i in range(len(ekezetes),0,-1):
    ekezetes_ford+=ekezetes[i-1]
"""
Miért i-1 az index:
Az ekezetes sztringben len(ekezetes) indexű karakter már nincs!
Visszafelé lépkedve pedig 0 itt is a befejező utáni karakter indexe!
Az i értékei len(ekezetes),...,1, az indexek pedig visszafelé len(ekezetes)-1,...,0
"""

print(ekezetes_ford)
print("\n")



cimiro("Sztringek összehasonlítása","-")

print(f'{ekezetes[3:6]} in {ekezetes}: {ekezetes[3:6] in ekezetes}\n')

print(f'{ekezetes_nb} == {ekezetes}: {ekezetes_nb==ekezetes}')
print(f'{ekezetes_nb} != {ekezetes}: {ekezetes_nb!=ekezetes}')
print(f'{ekezetes_nb} <= {ekezetes}: {ekezetes_nb<=ekezetes}')
print(f'{ekezetes_nb} < {ekezetes}: {ekezetes_nb<ekezetes}')
print(f'{ekezetes_nb} >= {ekezetes}: {ekezetes_nb>=ekezetes}')
print(f'{ekezetes_nb} > {ekezetes}: {ekezetes_nb>ekezetes}')

print(f'\nVigyázat! Ez az összehasonlítás ASCII-kódok alapján történik!\n\
Pl. {"őr"} < {"úr"}: {"őr"<"úr"}\n')



cimiro("Sztringek összehasonlítása a magyar ábécé alapján","-")

print("Az összehasonlítást karakterenként végezzük, azaz most pl. csemege < cukor")
print("A kis- és nagybetűk között nem teszünk különbséget.")
print("Ha a szó valamelyik karaktere nincs az ábécében, akkor az ASCII-kód dönt.\n")

abc_sorrendben("asztal","asztal")
abc_sorrendben("csemege","cukor")
abc_sorrendben("hét","hétfő")
abc_sorrendben("Petrarca","Petőfi")
abc_sorrendben("tőr","túró")
abc_sorrendben("nyitva tartás","nyitvatartás")
abc_sorrendben("Kisalföld","Kis-Balaton")
abc_sorrendben("alfa@email.hu","alfahím")
abc_sorrendben(ekezetes,ekezetes_nb)
abc_sorrendben(ekezetes[-11:],ekezetes_nb[:6])

print("\n")



cimiro("Szöveg magánhangzóinak megszámolása","-")

db=0
for i in range(len(ekezetes_nb)):
    if maganhangzo_e(ekezetes_nb[i]):
        db+=1

print(f'A "{ekezetes_nb}" {db} magánhangzót tartalmaz.\n')
```

```
cimiro("4. Egydimenziós listák","=")

"""
A Pythonban nincs vektor, másnéven tömb adattípus, helyette a sokkal általánosabb lista.
A lista elemeinek nem kell azonos típusúnak lennie, nem kell előre megadni a méretét.
Felépítése:

lista=[]
Lista.append(új_elem)
...

A lista hossza, azaz elemeinek száma: len(lista)
Az i. elem elérése: lista[i], i=0,1,2,...,len(lista)-1
Részlista az i. elemtől a j-ig: lista[i:j+1]
Új elem beszúrása a lista i. helyére: lista.insert(i,új_elem)
A lista i. elemének módosítása: lista[i]=új_elem
A lista i. elemének törlése: del(lista[i])
Vigyázni kell, hogy csak a 0,1,2,...,len(lista)-1 tartományba eső indexet adjunk meg!
Megnézhetjük, hányszor szerepel egy elem a listában: lista.count(elem)
Megkereshetjük egy elem első előfordulásának indexét a listában: lista.index(elem)
Törölhetjük egy elem első előfordulását a listából: lista.remove(elem)
Megfordíthatjuk a teljes listát: lista.reverse()
Ha az elemek sorbarendezhetőek, rendezhetjük a listát: lista.sort()
Vigyázat! A sztringeket ez ASCII-kód szerint rendezi.

Lista másolása:
A lista2=lista1 utasítás nem hoz létre új listát, a két név ugyanarra a memóriatartományra mutat.
Ha lista2 valamelyik elemét megváltoztatjuk, az a lista1-ben is meg fog változni.
Ezt elkerülendő a szeletátadó utasítást használjuk: lista2=lista1[:]
Használható a lista egy részének átadására is. Pl. az első 5 elem átadása: lista3=lista1[:5]

A teljes lista kiíratása: print(list),
A lista bejárásával a kiíratáskor az elemek formázhatók:

A range() függvénnyel:
for i in range(len(lista)):
    print(f'{i:3}. elem: {lista[i]:szélesség_megadása}')

Az enumerate() függvénnyel:
for i,ertek in enumerate(lista):
    print(f'{i:3}. elem: {ertek:szélesség_megadása}')
Itt nem kell a lista hossza és a lista[i] elemre a megadott névvel hivatkozhatunk.

A listához hasonló adatszerkezet a tuple, de [] helyett ()-ben adjuk meg.
Ha egyszer megadtuk, már nem változtatható meg. Nem fogjuk használni.
"""
```

```
cimiro("Anna bevásárol a piacon","-")

print("Anna elindul a piacra üres kosárral.")
kosar=[]
print(f'kosár = {kosar}')
print("Az első árusnál krumplit vesz. Beteszi a kosara aljára.")
kosar.append("krumpli")
print(f'kosár = {kosar}')
print("A második árusnál zöldséget vesz. Beteszi a kosarába.")
kosar.append("zöldség")
print(f'kosár = {kosar}')
print("A harmadik árusnál almát vesz. Beteszi a kosarába.")
kosar.append("alma")
print(f'kosár = {kosar}')
print("A negyedik árusnál diót vesz. Becsúsztatja a kosarába a krumpli fölé.")
kosar.insert(1,"dió")
print(f'kosár = {kosar}')
print("Az ötödik árusnál szőlőt vesz. Beteszi a kosarába.")
kosar.append("szőlő")
print(f'kosár = {kosar}\n')

print("A range() függvénnyel formázva kiíratva:\n")
for i in range(len(kosar)):
    print(f'{i:3}. áru: {kosar[i]:^7}') # középre igazítva 7 mező szélességben
print()

print("Az enumerate() függvénnyel formázva kiíratva:\n")
for i,aru in enumerate(kosar):
    print(f'{i:3}. áru: {aru:^7}') # {aru:7}: balra, ez az alap, {aru:>7}: jobbra
print()

print("Anna hazaérve a konyhában kiborítja a kosár tartalmát az asztalra,")
print("ahová fordított sorrendben kerül:")

asztal=kosar[:]
asztal.reverse()
kosar[:]=[]

print(f'kosár = {kosar}')
print(f'asztal = {asztal}')
print()

print("A pedáns Anna névsorba rendezi az árukat az asztalon:")

asztal.sort()
print(f'asztal = {asztal}')
print()

print("Ezután Anna először a szőlőt teszi be a hűtőbe:")

asztal.remove("szőlő")
print(f'asztal = {asztal}')
print()

print("Belép a konyhába Anna férje és a diót keresi az asztalon:")

van_e="dió" in asztal
if van_e:
    hely=asztal.index("dió") # Ha nincs a listában, az index() hibajelzést ad!
else:
    hely="-"

print(f'Van-e dió az aztalon: {van_e}, hányadik helyen: {hely}\n')
```

```
cimiro("A leggyakoribb lottószám","-")

print("Generálunk egy listát, mely 1000 db 1 és 90 közötti lottószámot tartalmaz.")
print("Melyik szám fordul elő legtöbbször?\n")

lotto=[]
for i in range(1000):
    lotto.append(veletlenegesz(1,90))

print(f'Az először generált lottószám: {lotto[0]}, az utolsó: {lotto[len(lotto)-1]}.')
print(f'Az 1 gyakorisága: {lotto.count(1)}, a 90-é: {lotto.count(90)}.\n')

maxszam, maxdb=1, lotto.count(1)
"""
Kiindulásként azt feltételezzük, hogy az 1 fordult elő legtöbbször
Végigmegyünk 2-től 90-ig a számokon. Ha olyat találunk, ami többször fordul elő,
akkor az lesz az aktuális maximum.
"""

print("Az aktuális maximumok:\n")

for i in range(2,91):
    if lotto.count(i)>maxdb:
        maxszam, maxdb= i, lotto.count(i)
        print(f'szám: {maxszam:2}, előfordulás: {maxdb:3}')

print()
print(f'A legszerencsésebb lottószám: {maxszam}, előfordulásainak száma: {maxdb}\n.')



cimiro("5. Sztringek listává alakítása","=")


cimiro("Sztring átalakítása karakterlistává","-")

"""
Sztring karaktereit közvetlenül nem lehet módosítani,
de ha karakterlistává alakítjuk, akkor megtehetjük.
"""

print(f'Az átalakítandó sztring: {ekezetes_nb}')

"""
Nem foglalkozunk a nagybetűk átírásával,
az egyszerűség kedvéért a nagybetűs szöveget is csupa kisbetűssé alakítjuk.
"""
print(f'Kisbetűs alakra hozva: {ekezetes_nb.lower()}\n')

eklista=list(ekezetes_nb.lower())

print(f'Listába téve: {eklista[:13]}') # eleje
print(eklista[14:]) # vége, a szóköz maradt ki
print("\n")


cimiro("A ékezetes magánhangzók kicserélése távirati formára","-")


ekmgh=["á","é","í","ó","ö","ő","ú","ü","ű"]
tavmgh=["aa","ee","ii","oo","oe","ooe","uu","ue","uue"]

"""
Nagyon kell ügyelni rá, hogy mindkét listát
az összetartozó pároknak megfelelően készítsük el!
"""

print(f'Az ékezetes magánhangzók: {ekmgh}')
print(f'A távirati magánhangzók: {tavmgh}')
print()
```

```
for i in range(len(eklista)):
    if eklista[i] in ekmgh:
        n=ekmgh.index(eklista[i])
        eklista[i]=tavmgh[n]

"""
Végigmegyünk az átalakítandó listánk karakterein:
ha a vizsgált karakter benne van az ékezetes magánhangzók listájában,
lekérdezzük az indexét
és a listánkban kicseréljük a vizsgált karaktert a megfelelő indexű kifejezésre.
"""

print("Átalakítva:")
print(eklista[:13])
print(eklista[14:])
print("\n")



cimiro("A lista visszaalakítása sztringgé","-")

ekezetes_tav="".join(eklista)
"""
A join listaösszefűző függvény a lista elemeit egy sztringgé fűzi a megadott "ragasztó" segítségével.
Szerkezete: "ragasztó".join(sztring_lista)
"""
print(ekezetes_tav, "\n")

print("Visszaalakíthatjuk csupa nagybetűssé is:\n")

ekezetes_tav=ekezetes_tav.upper()
print(ekezetes_tav, "\n")

print("Vagy csak a kezdőbetűk legyenek nagybetűsek:\n")

ekezetes_tav=ekezetes_tav.title()
print(ekezetes_tav, "\n\n")



cimiro("Sztring feldarabolása listába","-")

het=" hétfő kedd szerda csütörtök péntek szombat vasárnap "

print("A sztring:")
print(het)
"""
A legtöbbször szóköz mentén darabolunk, de sokszor zavaró szóköz van
a szöveg elején vagy a végén, amit először eltávolítunk.
"""

het=het.strip()
print("\nA megtisztított sztring:")
print(het)

# Most jöhet a darabolás.

darhet=het.split(" ")
# a darabolókarakter megadása szóköz esetén nem szükséges: het.split()
print("\nA feldarabolt sztring:")
print(darhet)

print("\nA leghosszabb nap: ", end="")

maxnap, maxhossz=0, len(darhet[0])

for i in range(len(darhet)):
    if len(darhet[i])>maxhossz:
        maxnap, maxhossz=i,len(darhet[i])

print(f'{darhet[maxnap]}, karaktereinek száma: {maxhossz}.\n')
```

```
cimiro("6. Kétdimenziós listák","=")

print("1000 db 1 és 75 közötti véletlen egész számhármas előállítása,");
print("és tárolása kétdimenziós listában (táblázatban)\n");
print("A számhármasok hány százaléka határoz meg háromszöget?\n");


def haromszog_e(a,b,c):
    return a+b>c and a+c>b and b+c>a

szamharmasok=[] # ebbe a kétdimenziós listába olvassuk be a számhármasokat

"""
A kétdimenziós szamharmasok lista egydimenziós szamharmas listák listája.
A szamharmasok lista elemei tehát a 3 elemű szamharmas listák. Elemszáma 1000 lesz.
A szamharmasok[0] az első számhármast jelenti,
a szamharmasok[-1,1] az utolsó számhármas középső számát.
A szamharmasok lista felfogható egy táblázatnak, aminek 1000 sora és 3 oszlopa van.
A sorok for listával bejárhatók, és akkor már csak a soron belüli indexre kell hivatkozni.
Az oszlopok nem alkotnak listát, nem lehet alkalmazni rájuk pl. az in, index(), stb. lekérdezéseket,
pl. az első oszlop értékeit csak a sorok bejárásával érhetjük el.
"""

for i in range(1000):
    szamharmas=[] # ebbe az egydimenziós listába olvassuk be az előállított 3 számot
    for j in range(3):
        szamharmas.append(veletlenegesz(1,75))
    szamharmasok.append(szamharmas)

print(f'Az első számhármas: {szamharmasok[0]}, az utolsó: {szamharmasok[-1]}\n')

hszdb=0
for i in range(len(szamharmasok)):
    if haromszog_e(szamharmasok[i][0],szamharmasok[i][1],szamharmasok[i][2]):
        hszdb+=1

print(f'{hszdb} számhármas esetén teljesül a háromszögegyenlőtlenség,')
print(f'tehát a számhármasok {100*hszdb/len(szamharmasok):.2f}%-a határoz meg háromszöget.\n')


# Egyszerűbb algoritmussal:

hszdb=0
for sor in szamharmasok:
    if haromszog_e(sor[0],sor[1],sor[2]):
        hszdb+=1

print(f'Egyszerűbb algoritmussal: {hszdb} számhármas esetén teljesül a háromszögegyenlőtlenség,')
print(f'tehát a számhármasok {100*hszdb/len(szamharmasok):.2f}%-a határoz meg háromszöget.\n')



cimiro("A 75-öt tartalmazó számhármasok indexeinek meghatározása","-")

index75=[]
for i,sor in enumerate(szamharmasok):
    if 75 in sor:
        index75.append(i)

print(f'{len(index75)} db számhármasban szerepel a 75:\n{index75} \n')
```

```
cimiro("A távirati formára alakítás kétdimenziós listával","-")

eklista=list(ekezetes)

print(f'Az átalakítandó sztring: {ekezetes}\n')
print(f'Listába téve: {eklista[:13]}') # eleje
print(eklista[14:]) # vége, a szóköz maradt ki
print("\n")

"""
Most már eleve kisbetűs szöveget veszünk,
az esetleges nagybetűket láttuk, hogy könnyen át tudjuk alakítani.

A parlista kétdimenziós lista lesz, minden eleme maga is egy kételemű lista,
az első az ékezetes betű, a második a távirati kódja.
"""

mghparlista=[["á","aa"],["é","ee"],["í","ii"],["ó","oo"],["ö","oe"],["ő","ooe"],["ú","uu"],["ü","ue"],["ű","uue"]]

print(f'Az ékezetes és távirati magánhangzópárok listája:\n{mghparlista}\n')

"""
Az algoritmus hagyományos formában:
for i in range(len(eklista)):
    for j in range(len(mghparlista)):
        if eklista[i] == mghparlista[j][0]:
            eklista[i]=mghparlista[j][1]

Ugyanez egyszerűbben, "pythonosabban":
"""

for i,e in enumerate(eklista):
    for par in mghparlista:
        if e == par[0]:
            eklista[i]=par[1]

"""
Végigmegyünk az átalakítandó listánk karakterein:
mindegyinél végigmegyünk az ékezetes magánhangzók párlistáján is:
ha a vizsgált karakter megegyezik az aktuális ékezetes magánhangzóval,
a listánkban kicseréljük a vizsgált karaktert az aktuális pár másik tagjára.
"""

print("Átalakítva:")
print(eklista[:13])
print(eklista[14:])
print()

ekezetes_tav="".join(eklista)
print(f'Visszaalakítva sztringgé: {ekezetes_tav}\n\n')



cimiro("7. A szótár adatszerkezet","=")

"""
Látszatra nincs nagy különbség aközött, hogy az elempárokat
kétdimenziós listában vagy szótárral feleltetjük meg egymásnak.
A szótár nem is tartozik szorosan a klasszikus programozáshoz,
de a konkrét feladat kapcsán mégis megemlítjük, mert a program futásakor
az elérése sokkal gyorsabb, mint a kétdimenziós listáé.

A szótárban nincs sorrend, a megfeleltetés kulcs segítségével történik.
Egy szótárban nem lehet két azonos kulcs, az értékek viszont azonosak is lehetnek.

Szerkezete:
szotarnev={kulcsn:értékn,...,kulcsi:értéki,...,kulcsm:értékm}

Megadása:
1. Felsorolással, pl. hexaszotar={1:"1",2:"2",...,10:"A",11:"B",...,15:"F"}
2. Felépítéssel, pl.
    angolszotar={}
    angolszotar["egy"]="one"
    ...

Bővítése: szotarnev[uj_kulcs]=érték
```

```
Egy érték lekérdezése, változóhoz rendelése: valtozo=szotarnev[létező_kulcs]
Vigyázat, ilyenkor nemlétező kulcs megadása futási hibát ad!
Ezért fontos a kulcs ellenőrzése is: kulcs in szotarnev, pl.
if 12 in hexaszotar:
    print(hexaszotar[12])

Egy érték módosítása: szotarnev[létező_kulcs]=új_érték

Elempár törlése a szótárból: del szotarnev[a_törlendő_elempár_kulcsa]
A szótár hossza: len(szotarnev)
"""

hexaszotar={1:"1",2:"2",3:"3",4:"4",5:"5",6:"6",7:"7",8:"8",9:"9",
            10:"A",11:"B",12:"C",13:"D",14:"E",15:"F"}

hexaszotar[31]="1F"
if 5 in hexaszotar:
    print(f'Az 5 hexadecimális alakja: {hexaszotar[5]}')
if 5 in hexaszotar:
    hexaszotar[5]="0005"
    print(f'Az 5 hexadecimális alakja: {hexaszotar[5]}')
for i in range(3,10):
    if i==5:
        continue # a ciklusmag további részének végrahajtása nélkül továbblép (kihagyja az 5-öt)
    del hexaszotar[i]

print(f'\nA módosított szótár: {hexaszotar}\n\n')



cimiro("A távirati formára alakítás szótár segítségével","-")

print(f'Az átalakítandó sztring: {ekezetes}\n')
print(f'Listába téve: {eklista[:13]}') # eleje
print(eklista[14:]) # vége, a szóköz maradt ki
print("\n")

mghszotar={"á":"aa","é":"ee","í":"ii","ó":"oo","ö":"oe","ő":"ooe","ú":"uu","ü":"ue","ű":"uue"}

print(f'Az ékezetes és távirati magánhangzópárok szótára:\n{mghszotar}\n')

for i in range(len(eklista)):
    if eklista[i] in mghszotar:
        eklista[i]=mghszotar[eklista[i]]

"""
Végigmegyünk az átalakítandó listánk karakterein:
ha a vizsgált karakter, mint kulcs benne van a szótárban,
a listánkban kicseréljük a vizsgált karaktert a kulcsnak megfelelő értékre.
"""

print("Átalakítva:")
print(eklista[:13])
print(eklista[14:])
print()

ekezetes_tav="".join(eklista)
print(f'Visszaalakítva sztringgé: {ekezetes_tav}\n\n')


input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========== RESTART: C:\Python\KlInfoPy\BevGyak02\BevGyak02.py ==========

Adatszerkezetek: számok, karakterek, szöveg és listák
**************************************************


1. Egész számok
===============


Műveletek két 10 és 100 közötti véletlen egész számmal
------------------------------------------------------

Az első generált egész szám 57, a második 67, 67 a nagyobb.
57+67 = 124, 67 - 57 = 10, 57*67 = 3819, 67^57 = 1.220E+104
67 // 57 = 1, 67 mod 57 = 10, 57 / 67 = 0.8507

Faktoriálisok
-------------

  1! =          1
  2! =          2
  3! =          6
  4! =         24
  5! =        120
  6! =        720
  7! =       5040
  8! =      40320
  9! =     362880
 10! =    3628800
 11! =   39916800
 12! =  479001600
 13! =  6.227E+09
 14! =  8.718E+10
 15! =  1.308E+12
 16! =  2.092E+13
 17! =  3.557E+14
 18! =  6.402E+15
 19! =  1.216E+17
 20! =  2.433E+18
 ...
101! = 9.426E+159
102! = 9.614E+161
103! = 9.903E+163
104! = 1.030E+166
105! = 1.081E+168
106! = 1.146E+170
107! = 1.227E+172
108! = 1.325E+174
109! = 1.444E+176
110! = 1.588E+178
111! = 1.763E+180
112! = 1.975E+182
113! = 2.231E+184
114! = 2.544E+186
115! = 2.925E+188
116! = 3.393E+190
117! = 3.970E+192
118! = 4.685E+194
119! = 5.575E+196
120! = 6.690E+198
```

```
2. Valós számok
===============


Műveletek két -100 és 100 közötti véletlen valós számmal
--------------------------------------------------------

Az első generált valós szám -18.245355, a második -24.901.
|-24.901| = 24.901
-18.245 * 3.142 = -57.319, 24.901^2.718 = 6.242E+03
log10(18.245) = 1.261, 18.245^-24.901 = 3.939E-32



3. Karakterek és sztringek
==========================

A @ karakter ASCII-kódja 64


Az ASCII-kódokhoz tartozó karakterek
------------------------------------

 31:     32:     33: !   34: "   35: #   36: $   37: %   38: &   39: '   40: (
 41: )   42: *   43: +   44: ,   45: -   46: .   47: /   48: 0   49: 1   50: 2
 51: 3   52: 4   53: 5   54: 6   55: 7   56: 8   57: 9   58: :   59: ;   60: <
 61: =   62: >   63: ?   64: @   65: A   66: B   67: C   68: D   69: E   70: F
 71: G   72: H   73: I   74: J   75: K   76: L   77: M   78: N   79: O   80: P
 81: Q   82: R   83: S   84: T   85: U   86: V   87: W   88: X   89: Y   90: Z
 91: [   92: \   93: ]   94: ^   95: _   96: `   97: a   98: b   99: c  100: d
101: e  102: f  103: g  104: h  105: i  106: j  107: k  108: l  109: m  110: n
111: o  112: p  113: q  114: r  115: s  116: t  117: u  118: v  119: w  120: x
121: y  122: z  123: {  124: |  125: }  126: ~  127: □  128: □  129: □  130: □
131: □  132: □  133: □  134: □  135: □  136: □  137: □  138: □  139: □  140: □
141: □  142: □  143: □  144: □  145: □  146: □  147: □  148: □  149: □  150: □
151: □  152: □  153: □  154: □  155: □  156: □  157: □  158: □  159: □  160:
161: ¡  162: ¢  163: £  164: ¤  165: ¥  166: ¦  167: §  168: ¨  169: ©  170: ª
171: «  172: ¬  173: -  174: ®  175: ¯  176: °  177: ±  178: ²  179: ³  180: ´
181: µ  182: ¶  183: ·  184: ¸  185: ¹  186: º  187: »  188: ¼  189: ½  190: ¾
191: ¿  192: À  193: Á  194: Â  195: Ã  196: Ä  197: Å  198: Æ  199: Ç  200: È
201: É  202: Ê  203: Ë  204: Ì  205: Í  206: Î  207: Ï  208: Ð  209: Ñ  210: Ò
211: Ó  212: Ô  213: Õ  214: Ö  215: ×  216: Ø  217: Ù  218: Ú  219: Û  220: Ü
221: Ý  222: Þ  223: ß  224: à  225: á  226: â  227: ã  228: ä  229: å  230: æ
231: ç  232: è  233: é  234: ê  235: ë  236: ì  237: í  238: î  239: ï  240: ð
241: ñ  242: ò  243: ó  244: ô  245: õ  246: ö  247: ÷  248: ø  249: ù  250: ú
251: û  252: ü  253: ý  254: þ  255: ÿ  256: Ā  257: ā  258: Ă  259: ă  260: Ą
261: ą  262: Ć  263: ć  264: Ĉ  265: ĉ  266: Ċ  267: ċ  268: Č  269: č  270: Ď
271: ď  272: Đ  273: đ  274: Ē  275: ē  276: Ĕ  277: ĕ  278: Ė  279: ė  280: Ę
281: ę  282: Ě  283: ě  284: Ĝ  285: ĝ  286: Ğ  287: ğ  288: Ġ  289: ġ  290: Ģ
291: ģ  292: Ĥ  293: ĥ  294: Ħ  295: ħ  296: Ĩ  297: ĩ  298: Ī  299: ī  300: Ĭ
301: ĭ  302: Į  303: į  304: İ  305: ı  306: Ĳ  307: ĳ  308: Ĵ  309: ĵ  310: Ķ
311: ķ  312: ĸ  313: Ĺ  314: ĺ  315: Ļ  316: ļ  317: Ľ  318: ľ  319: Ŀ  320: ŀ
321: Ł  322: ł  323: Ń  324: ń  325: Ņ  326: ņ  327: Ň  328: ň  329: ŉ  330: Ŋ
331: ŋ  332: Ō  333: ō  334: Ŏ  335: ŏ  336: Ő  337: ő  338: Œ  339: œ  340: Ŕ
341: ŕ  342: Ŗ  343: ŗ  344: Ř  345: ř  346: Ś  347: ś  348: Ŝ  349: ŝ  350: Ş
351: ş  352: Š  353: š  354: Ţ  355: ţ  356: Ť  357: ť  358: Ŧ  359: ŧ  360: Ũ
361: ũ  362: Ū  363: ū  364: Ŭ  365: ŭ  366: Ů  367: ů  368: Ű  369: ű  370: Ų
```

```
A "tündérszárnyú kígyóbűvölő" karaktereinek ASCII-kódjai
---------------------------------------------------------

   t   ü   n   d   é   r   s   z   á   r   n   y   ú       k   í   g   y   ó   b   ű   v   ö   l   ő
 116 252 110 100 233 114 115 122 225 114 110 121 250  32 107 237 103 121 243  98 369 118 246 108 337

Nagybetűs alakban:

   T   Ü   N   D   É   R   S   Z   Á   R   N   Y   Ú       K   Í   G   Y   Ó   B   Ű   V   Ö   L   Ő
  84 220  78  68 201  82  83  90 193  82  78  89 218  32  75 205  71  89 211  66 368  86 214  76 336


Műveletek sztringekkel
======================


Rész-sztringek és összefűzésük
------------------------------

TÜNDÉRbűvölő

KÍGYÓszárny

KÍGYÓszárny

ÖŐöő

Az összefűzésben nem lehetnek számok, csak sztringek!

Körmendi Károly: 1988
Körmendi Károly: 1988 Körmendi Károly: 1988 Körmendi Károly: 1988



Sztring megfordítása
--------------------

tündérszárnyú kígyóbűvölő
őlövűbóygík úynrázsrédnüt



Sztringek összehasonlítása
--------------------------

dér in tündérszárnyú kígyóbűvölő: True

TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ == tündérszárnyú kígyóbűvölő: False
TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ != tündérszárnyú kígyóbűvölő: True
TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ <= tündérszárnyú kígyóbűvölő: True
TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ < tündérszárnyú kígyóbűvölő: True
TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ >= tündérszárnyú kígyóbűvölő: False
TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ > tündérszárnyú kígyóbűvölő: False

Vigyázat! Ez az összehasonlítás ASCII-kódok alapján történik!
Pl. őr < úr: False
```

```
Sztringek összehasonlítása a magyar ábécé alapján
--------------------------------------------------

Az összehasonlítást karakterenként végezzük, azaz most pl. csemege < cukor
A kis- és nagybetűk között nem teszünk különbséget.
Ha a szó valamelyik karaktere nincs az ábécében, akkor az ASCII-kód dönt.

asztal == asztal
csemege < cukor
hét < hétfő
Petrarca > Petőfi
tőr < túró
nyitva tartás < nyitvatartás
Kisalföld > Kis-Balaton
alfa@email.hu < alfahím
tündérszárnyú kígyóbűvölő == TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ
kígyóbűvölő < TÜNDÉR



Szöveg magánhangzóinak megszámolása
------------------------------------

A "TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ" 9 magánhangzót tartalmaz.


4. Egydimenziós listák
======================


Anna bevásárol a piacon
-----------------------

Anna elindul a piacra üres kosárral.
kosár = []
Az első árusnál krumplit vesz. Beteszi a kosara aljára.
kosár = ['krumpli']
A második árusnál zöldséget vesz. Beteszi a kosarába.
kosár = ['krumpli', 'zöldség']
A harmadik árusnál almát vesz. Beteszi a kosarába.
kosár = ['krumpli', 'zöldség', 'alma']
A negyedik árusnál diót vesz. Becsúsztatja a kosarába a krumpli fölé.
kosár = ['krumpli', 'dió', 'zöldség', 'alma']
Az ötödik árusnál szőlőt vesz. Beteszi a kosarába.
kosár = ['krumpli', 'dió', 'zöldség', 'alma', 'szőlő']

A range() függvénnyel formázva kiíratva:

  0. áru: krumpli
  1. áru:   dió
  2. áru: zöldség
  3. áru:  alma
  4. áru:  szőlő

Az enumerate() függvénnyel formázva kiíratva:

  0. áru: krumpli
  1. áru:   dió
  2. áru: zöldség
  3. áru:  alma
  4. áru:  szőlő
```

```
Anna hazaérve a konyhában kiborítja a kosár tartalmát az asztalra,
ahová fordított sorrendben kerül:
kosár = []
asztal = ['szőlő', 'alma', 'zöldség', 'dió', 'krumpli']

A pedáns Anna névsorba rendezi az árukat az asztalon:
asztal = ['alma', 'dió', 'krumpli', 'szőlő', 'zöldség']

Ezután Anna először a szőlőt teszi be a hűtőbe:
asztal = ['alma', 'dió', 'krumpli', 'zöldség']

Belép a konyhába Anna férje és a diót keresi az asztalon:
Van-e dió az aztalon: True, hányadik helyen: 1


A leggyakoribb lottószám
------------------------

Generálunk egy listát, mely 1000 db 1 és 90 közötti lottószámot tartalmaz.
Melyik szám fordul elő legtöbbször?

Az először generált lottószám: 23, az utolsó: 44.
Az 1 gyakorisága: 11, a 90-é: 11.

Az aktuális maximumok:

szám:  2, előfordulás:  15
szám:  4, előfordulás:  20

A legszerencsésebb lottószám: 4, előfordulásainak száma: 20
.

5. Sztringek listává alakítása
==============================


Sztring átalakítása karakterlistává
------------------------------------

Az átalakítandó sztring: TÜNDÉRSZÁRNYÚ KÍGYÓBŰVÖLŐ
Kisbetűs alakra hozva: tündérszárnyú kígyóbűvölő

Listába téve: ['t', 'ü', 'n', 'd', 'é', 'r', 's', 'z', 'á', 'r', 'n', 'y', 'ú']
['k', 'í', 'g', 'y', 'ó', 'b', 'ű', 'v', 'ö', 'l', 'ő']



A ékezetes magánhangzók kicserélése távirati formára
-----------------------------------------------------

Az ékezetes magánhangzók: ['á', 'é', 'í', 'ó', 'ö', 'ő', 'ú', 'ü', 'ű']
A távirati magánhangzók: ['aa', 'ee', 'ii', 'oo', 'oe', 'ooe', 'uu', 'ue', 'uue']

Átalakítva:
['t', 'ue', 'n', 'd', 'ee', 'r', 's', 'z', 'aa', 'r', 'n', 'y', 'uu']
['k', 'ii', 'g', 'y', 'oo', 'b', 'uue', 'v', 'oe', 'l', 'ooe']
```

```
A lista visszaalakítása sztringgé
---------------------------------

tuendeerszaarnyuu kiigyoobuuevoelooe

Visszaalakíthatjuk csupa nagybetűssé is:

TUENDEERSZAARNYUU KIIGYOOBUUEVOELOOE

Vagy csak a kezdőbetűk legyenek nagybetűsek:

Tuendeerszaarnyuu Kiigyoobuuevoelooe



Sztring feldarabolása listába
-----------------------------

A sztring:
 hétfő kedd szerda csütörtök péntek szombat vasárnap

A megtisztított sztring:
hétfő kedd szerda csütörtök péntek szombat vasárnap

A feldarabolt sztring:
['hétfő', 'kedd', 'szerda', 'csütörtök', 'péntek', 'szombat', 'vasárnap']

A leghosszabb nap: csütörtök, karaktereinek száma: 9.


6. Kétdimenziós listák
======================

1000 db 1 és 75 közötti véletlen egész számhármas előállítása,
és tárolása kétdimenziós listában (táblázatban)

A számhármasok hány százaléka határoz meg háromszöget?

Az első számhármas: [34, 20, 21], az utolsó: [74, 63, 66]

487 számhármas esetén teljesül a háromszögegyenlőtlenség,
tehát a számhármasok 48.70%-a határoz meg háromszöget.

Egyszerűbb algoritmussal: 487 számhármas esetén teljesül a háromszögegyenlőtlenség,
tehát a számhármasok 48.70%-a határoz meg háromszöget.


A 75-öt tartalmazó számhármasok indexeinek meghatározása
---------------------------------------------------------

43 db számhármasban szerepel a 75:
[13, 44, 52, 55, 98, 130, 133, 169, 179, 191, 194, 204, 271, 286, 299, 308, 309, 320, 368, 382, 389, 422, 448,
483, 519, 529, 546, 549, 556, 579, 604, 607, 625, 690, 695, 723, 724, 756, 787, 799, 840, 848, 982]
```

```
A távirati formára alakítás kétdimenziós listával
-------------------------------------------------

Az átalakítandó sztring: tündérszárnyú kígyóbűvölő

Listába téve: ['t', 'ü', 'n', 'd', 'é', 'r', 's', 'z', 'á', 'r', 'n', 'y', 'ú']
['k', 'í', 'g', 'y', 'ó', 'b', 'ű', 'v', 'ö', 'l', 'ő']


Az ékezetes és távirati magánhangzópárok listája:
[['á', 'aa'], ['é', 'ee'], ['í', 'ii'], ['ó', 'oo'], ['ö', 'oe'], ['ő', 'ooe'], ['ú', 'uu'], ['ü', 'ue'], ['ű',
'uue']]

Átalakítva:
['t', 'ue', 'n', 'd', 'ee', 'r', 's', 'z', 'aa', 'r', 'n', 'y', 'uu']
['k', 'ii', 'g', 'y', 'oo', 'b', 'uue', 'v', 'oe', 'l', 'ooe']

Visszaalakítva sztringgé: tuendeerszaarnyuu kiigyoobuuevoelooe



7. A szótár adatszerkezet
=========================

Az 5 hexadecimális alakja: 5
Az 5 hexadecimális alakja: 0005

A módosított szótár: {1: '1', 2: '2', 5: '0005', 10: 'A', 11: 'B', 12: 'C', 13: 'D', 14: 'E', 15: 'F', 31: '1F'}



A távirati formára alakítás szótár segítségével
-----------------------------------------------

Az átalakítandó sztring: tündérszárnyú kígyóbűvölő

Listába téve: ['t', 'ue', 'n', 'd', 'ee', 'r', 's', 'z', 'aa', 'r', 'n', 'y', 'uu']
['k', 'ii', 'g', 'y', 'oo', 'b', 'uue', 'v', 'oe', 'l', 'ooe']


Az ékezetes és távirati magánhangzópárok szótára:
{'á': 'aa', 'é': 'ee', 'í': 'ii', 'ó': 'oo', 'ö': 'oe', 'ő': 'ooe', 'ú': 'uu', 'ü': 'ue', 'ű': 'uue'}

Átalakítva:
['t', 'ue', 'n', 'd', 'ee', 'r', 's', 'z', 'aa', 'r', 'n', 'y', 'uu']
['k', 'ii', 'g', 'y', 'oo', 'b', 'uue', 'v', 'oe', 'l', 'ooe']

Visszaalakítva sztringgé: tuendeerszaarnyuu kiigyoobuuevoelooe


A befejezéshez nyomd meg az ENTER billentyűt!
```

3. Beolvasás, kiírás: a képernyő és a szöveges fájlok kezelése

```
"""
3. Beolvasás, kiírás: a képernyő és a szöveges fájlok kezelése
@author Klemand66
"""

from random import *
from math import *
from datetime import *
from locale import *

def cimiro(cim,karakter):
    print("\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")


cimiro("Beolvasás, kiírás: a képernyő és a szöveges fájlok kezelése","*")

cimiro("1. Egész számok beolvasása a billentyűzetről","=")

"""
Mivel az érettségin feltételezhetjük, hogy helyes számformátumot és értéket
adnak meg, kivételkezeléssel és ellenőrzéssel sehol sem kell foglalkoznunk.
"""

n=int(input("Kérek egy egész számot a billentyűzetről: "))

"""
Az input() függvény bekér egy adatot a billentyűzetről,
aminek a típusa minden esetben sztring lesz, amit azonban
megfelelő formátum esetén rögtön átalakíthatunk a kívánt típusra.
"""

m=int(input("\nKérek egy másik egész számot is a billentyűzetről: "))

print(f'\n{n} és {m} távolsága a számegyenesen {abs(n-m)}\n')

nagyobb=max(n,m)

print(f'A nagyobb szám {nagyobb}, bináris alakja: {nagyobb:b},\n\
oktális alakja: {nagyobb:o} és hexadecimális alakja: {nagyobb:x}\n')

"""
Ezeket az f-string alakokat változóba is tehetjük, pl. nagyobb16=f'{nagyobb:x}'.
Másik lehetőség: nagyobbhex=hex(nagyobb). Mindkét alak sztring típusú.
"""

nagyobb16=f'{nagyobb:x}'
nagyobbhex=hex(nagyobb)

print(f'f\'{chr(123)}nagyobb:x{chr(125)}\' = {nagyobb16}, típusa: {type(nagyobb16)}, \
hex(nagyobb) = {nagyobbhex}, típusa: {type(nagyobbhex)}\n')

cimiro("2. Tetszőleges (2-36) számrendszerbeli számok decimális alakja","=")

szj=["10: A", "11: B", "12: C", "13: D", "14: E", "15: F", "16: G", "17: H", "18: I", "19: J",
     "20: K", "21: L", "22: M", "23: N", "24: O", "25: P", "26: Q", "27: R", "28: S", "29: T",
     "30: U", "31: V", "32: W", "33: X", "34: W", "35: Z"]

print("Kettes számrendszerben a számjegyek csak 0 és 1 lehetnek!")
sz2=(input("Kérek egy kettes számrendszerbeli egész számot: "))
sz10=int(sz2,2)
print(f'A megadott {sz2} bináris szám alakja tízes számrendszerben: {sz10:,}')
print(f'Ez szám lesz: {sz10:,}; négyzete: {sz10**2:,}\n')
```

```
print(f'Tizenhatos számrendszerben az alfabetikus számjegyek :\n{szj[:6]}')
sz16=(input("Kérek egy tizenhatos számrendszerbeli egész számot: "))
sz10=int(sz16,16)
print(f'A megadott {sz16} hexadecimális szám alakja tízes számrendszerben: {sz10:,}')
print(f'Ez szám lesz: {sz10:,}; négyzete: {sz10**2:,}\n')

print("36-os számrendszerben az alfabetikus számjegyek:")
print(szj[:10])
print(szj[10:20])
print(szj[20:26])
sz36=(input("Kérek egy harminchatos számrendszerbeli egész számot: "))
sz10=int(sz36,36)
print(f'A megadott {sz36} 36-os számrendszerbeli szám alakja tízes számrendszerben: {sz10:,}')
print(f'Ez szám lesz: {sz10:,}; négyzete: {sz10**2:,}\n')

n=int((input("Kérem a számrendszer alapját (2-36): ")))
if n>10:
    print(f'{n} alapú számrendszerben az alfabetikus számjegyek:')
    print(szj[:min(10,n-10)])
if n>20:
    print(szj[10:min(20,n-10)])
if n>30:
    print(szj[20:n-10])

szn=(input(f'Kérek egy {n} alapú számrendszerbeli egész számot: '))
sz10=int(szn,n)
print(f'A megadott {szn} {n} alapú számrendszerbeli szám alakja tízes számrendszerben: {sz10:,}')
print(f'Ez szám lesz: {sz10:,}; négyzete: {sz10**2:,}\n\n')



cimiro("2. Egy hosszú egész szám beolvasása a billentyűzetről","=")

"""
A Pythonban nincsenek külön rövid és hosszú egészek, az egész szám elvileg bármilyen nagy lehet.
"""

cimiro("Egy billentyűzetről bekért szám számjegyeinek összeadogatása egyjegyű összegig","-")

"""
Egy tipikusan (duplán) hátultesztelő ciklusus feladat, hiszen
 - egyszer mindenképpen össze kell adnunk a számjegyeket
 - és a szám számjegyekre bontásnál is meg kell határoznunk legalább egy sszámjegyet.
"""

szam=int(input("Kérek egy tetszőlegesen nagy pozitív egész számot: "))
print()

masolat=szam # a True-ciklushoz

"""
A hátultesztelő ciklus hiányát úgy is orvosolhatjuk, hogy olyan "jól megválaszttott"
kezdőértéket adunk meg, hogy az elöltesztelő ciklus először biztosan belépjen a ciklusmagba.
Egyébként mindegy, hogy mit, mert általában úgyis rögtön megváltoztatjuk az értékét.
"""
szamjegyosszeg=10 # A 10 nem egyjegyű, garantálja az első belépést.

while szamjegyosszeg>9: # ha (még) többegyjegyű a számjegyösszeg, akkor belépünk
    szamjegyosszeg=0 # máris megváltoztatjuk az önkényes kezdőértéket
    while szam>0: # ha (még) legalább egyjegyű a szám, akkor van összeadandó számjegye
        maradek=szam%10 # az aktuális szám utolsó jegye
        szamjegyosszeg+=maradek # az utolsó jegy hozzáadása az aktuális számjegyösszeghez
        szam=szam//10 # a szám utolsó jegyének elhagyása, előbb-utóbb 0-t kapunk
    szam=szamjegyosszeg
    """
    ez akkor kell, ha a számjegyösszeg még nem egyjegyű,
    a számjegyösszeg lesz az új szám, amin újra el kell vézni a számjegyek összeadását
    """
    print(f'Részeredmény: {szamjegyosszeg}') # ez csak a látvány kedvéért kell

print(f'A végső számjegyösszeg: {szamjegyosszeg}\n')
```

```
print("Nézzük meg a BevGyak01-ben megismert True-ciklusokkal is!\n")

szam=masolat

# Itt nem kell önkényes kezdőérték.

while True:
    szamjegyosszeg=0
    while True:
       maradek=szam%10
       szamjegyosszeg+=maradek
       szam=szam//10
       if szam==0:
           break
    print(f'Részeredmény: {szamjegyosszeg}')
    if szamjegyosszeg<10:
        break
    szam=szamjegyosszeg

print(f'A végső számjegyösszeg: {szamjegyosszeg}\n\n')



cimiro("3. Valós számok beolvasása a billentyűzetről","=")

"""
A tizedestörteket tizedesponttal kell beírni!
A kapott megfelelő formátumú sztringet a float() függvény alakítja valós számmá.
"""

x=float(input("Kérek egy valós számot a billentyűzetről: "))

y=float(input("\nKérek egy másik valós számot is a billentyűzetről: "))

print(f'\n{x} és {y} távolsága a számegyenesen {abs(x-y):.3f}\n')

print("A mértani közepük: ",end="")

if x>=0 and y>=0:
    print(f'{sqrt(x*y):.3f}')
else:
   print("A mértani közepet csak nemnegatív számokra értelmezzük.")

print("\n")



cimiro("4. Műveletek a billentyűzetről beolvasott sztringekkel","=")

cimiro("Hexadecimális RGB színkód elemzése","-")

kod16=input("Kérem a hexadecimális RGB-kódot, pl. A256FF: ").upper()
# Rögtön nagybetűssé alakítjuk.

szinlista=["vörös", "zöld", "kék"]
kodlista10=[int(kod16[:2],16), int(kod16[2:4],16), int(kod16[4:],16)]
kodlista10str=[str(int(kod16[:2],16)), str(int(kod16[2:4],16)), str(int(kod16[4:],16))]
"""
A belső int() a hexadecimális sztringet decimális egész számmá alakítja,
a külső str() ezt konvertálja sztring típussá, mert csak sztringeket lehet összefűzni.
"""

foszin=max(kodlista10)
foindex=kodlista10.index(foszin)

print(f'\nA megadott szín hexadecimális kódja: {"#"+kod16}, decimális kódja: {",".join(kodlista10str)}')
# A join listaösszefűző függvény szerkezete: "ragasztó".join(sztring_lista)

print(f'\nA kód összetevői tízes számrendszerben: \
vörös: {kodlista10[0]}, zöld: {kodlista10[1]}, kék: {kodlista10[2]}.\n')

print(f'A főszín: {szinlista[foindex]}, maximális értékének {100*kodlista10[foindex]/255:.2f}%-a,\n\
a felhasznált színmennyiség {100*kodlista10[foindex]/sum(kodlista10):.2f}%-a,\n\
a teljes színmennyiségnek pedig {100*kodlista10[foindex]/(3*255):.2f}%-a.\n\n')
```

```
cimiro("A billentyűzetről beolvasott szöveg magánhangzóinak füzére","-")

def mgh_e(kar):
    mgh = "aáeéiíoóöőuúüűAÁEÉIÍOÓÖŐUÚÜŰ"
    return kar in mgh


mondat=input("Írj be egy magyar mondatot: ")

mghfuzer=""
"""
for i in range(len(mondat)):
    betu=mondat[i]
    if mgh_e(betu):
        mghfuzer+=betu

Ugyanez egyszerűbben:
"""
for betu in mondat:
    if mgh_e(betu):
        mghfuzer+=betu

print(f'A megadott mondat magánhangzóinak füzére: {mghfuzer}.\n\n')


cimiro("Egész számok sorozatának beolvasása","-")

sor=input("Kérem a számsorozat tagjait (legalább egyet) egy sorban szóközzel elválasztva: ")
# A sor egy sztring, amit megtisztítunk és feldarabolunk

darsor=sor.strip().split()
"""
A strip() a szélekről tünteti el a felesleges szóközöket,
a split() a szóközök mentén listába darabolja a sztringet.
Tetszőleges karakter mentén lehet darabolni: split("karakter")
"""

# A darsor elemei is sztringek, egész számmá alakítva egy új listába helyezzük őket:
X=[]
"""
for i in range(len(darsor)):
    X.append(int(darsor[i]))
"""
for elem in darsor:
    X.append(int(elem))

print(f'\nA számsorozat: {X}, {len(X)} tagból áll, {X.count(0)} db 0 van köztük. ')
"""
Figyeljük meg a len() függvény és a count() metódus eltérő szekezetét.
A len() argumentuma az X, ehhez rendel egy számot, az X hosszát,
míg a count() metódus értelmezési tartománya az X, argumentuma az elem, amit megszámol.
"""

print(f'A legkisebb szám: {min(X)}, a legnagyobb: {max(X)}')
print(f'A számok összege: {sum(X)}, átlaga: {sum(X)/len(X):.2f}\n\n')
```

```
cimiro("5. Beolvasás szövegfájlból","=")

cimiro("a) A 3szog.txt szövegfájl első sora tartalmazza a beolvasandó adatsorok számát.","-")

print("Ezután minden sorban szóközzel elválasztva egy-egy adathármas szerepel.");
print("Olvassuk be egy kétdimenziós listába (tömbbe) az adathármasokat!");
print("Ezután írassuk ki a képernyőre, hogy háromszöget határoznak-e meg,");
print("és ha igen, akkor adjuk meg a kerületüket, a területüket,");
print("a beírt és a körülírt körük sugarát 2 tizedesjeggyel!\n");

betxt = open("3szog.txt","r")
"""
A beolvasandó szövefájlnak a programmal azonos mappában kell lennie!
Az "r" alapértelmezés, nem kötelező kiírni.
"""

elsosor=betxt.readline()
sordb=int(elsosor.strip())

szamharmasok=[]

"""
for i in range(sordb):
    sor=betxt.readline() # egy beolvasott sor ad meg egy számhármast
    darsor=sor.strip().split()
    szamharmas=[]
    for j in range(3):
        szamharmas.append(float(darsor[j]))
    szamharmasok.append(szamharmas) # az elkészített azámhármast hozzáfűzzük a számhármasokhoz
"""

for i in range(sordb):
    sor=betxt.readline() # egy beolvasott sor ad meg egy számhármast
    darsor=sor.strip().split()
    szamharmas=[]
    for elem in darsor:
        szamharmas.append(float(elem))
    szamharmasok.append(szamharmas) # az elkészített azámhármast hozzáfűzzük a számhármasokhoz

betxt.close()
"""
A sima open-es megnyitásnál nekünk kell lezárnunk a fájlt.
Az érettségin a lezárás elmaradása pontvesztést jelent.
"""

def haromszog_e(a,b,c):
    return a+b>c and a+c>b and b+c>a
"""
for i in range(sordb):
    a=szamharmasok[i][0]
    b=szamharmasok[i][1]
    c=szamharmasok[i][2]
"""

for elem in szamharmasok:
    a=elem[0]
    b=elem[1]
    c=elem[2]
    if haromszog_e(a,b,c):
        print(f'{a:.2f}, {b:.2f} és {c:.2f} háromszöget határoznak meg.')
        k=a+b+c
        s=k/2
        t=sqrt(s*(s-a)*(s-b)*(s-c))
        rb=t/s
        rk=a*b*c/(4*t)
        print(f'Kerülete: {k:.2f}, területe: {t:.2f}, \
beírt körének sugara: {rb:.2f}, körülírt körének sugara: {rk:.2f}.\n')
    else:
        print(f'{a:.2f}, {b:.2f} és {c:.2f} nem határoznak meg háromszöget.\n')
```

```
cimiro("b) A szövegfájl első sora nem tartalmazza a beolvasandó adatsorok számát.","-")

"""
A Pythonban nem kell tudnunk előre, hogy a fájl max. hány sort fog tartalmazni.
Az érettségi feladatokban az ilyen jellegű információk számunkra feleslegesek,
de más nyelvek esetén, pl. a Javaban előre kell deklarálni a változók maximális méretét.

Az adatsorok számát, ill. a tároló lista méretét lekérdezhetjük utólag a len() függyvénnyel.
Vagyis az a) pontban megismert módszert mindig helyettesíthetjük ezzel.

Ami miatt mégis érdekes lehet a későbbiekben is,
az az egyedi sorok beolvasására szolgáló sor=betxt.readline() utasítás.
Sok feladatban előfordul, hogy az első néhány sor eltérő szerkezetű,
azokat így beolvassuk, feldolgozzuk, és utána alkalmazzuuk a most következő módszereket.

Az ékezetes karakterek a Pythonban is problémát okozhatnak.

A Microsoft Windowsban az ANSI kifejezés az operációs rendszer által támogatott karakterkészletet jelenti,
amely Észak-Amerikában és Nyugat-Európában a CP1252 (codepage), Magyarországon a CP1250.
"""

print(f'Alapértelmezett kódolás a gépünkön: {getpreferredencoding()}\n')
# Ehhez kellett a locale modul importálása.

"""
Ha nem adunk meg karakterkódolást, a Python az alapértelmezett kódolást fogja használni.
Ha a beolvasandó szövegben nincsenek ékezetes karakterek, akkor nem is lesz semmi gond,
akár ANSI, akár Utf-8 kódolású a beolvasandó szöveg. A típusát a jegyzettömbben az állapotsorban látjuk,
és ott meg is tudjuk változtatni: Mentés másként és a Kódolásnál kiválasztjuk a kívánt típust.
Ez az oka annak, hogy az érettségi feladatokban nem adnak ékezetes szövegeket
és nem is várnak el ékezetes kiíratást (bár azzal már nem lenne gond, kiírná az alapértelmezett
kódolásával).

De mi is a megoldás ékezetes txt fájl beolvasásánál? Sok helyen tanácsolják, hogy adjuk meg a kódolást
az open() függvényben beolvasásnál és kiíratásnál is: encoding="utf-8". Ez azonban csak akkor segít,
ha az alapértelmezett kódolás a CP1250, a beolvasandó szövegfájl viszont Utf-8 kódolású ékezetes fájl.
Ha viszont mégis ANSI kódolású ékezetes szövegfájlt kell beolvasnunk, éppen ez fogja a hibát okozni!

Azonban szerencsére két egyszerű módon is orvosolhatjuk a problémát:
1. Nézzük meg a jegyzettömbben a beolvasandó szövegfájlt és a kódolását, és csak akkor írjuk be
az encoding="utf-8" paramétert, ha Windows-t használunk és a szövegfájlunk ékezetes Utf-8 kódolású fájl.
2. Az ékezetes Utf-8 kódolású szövegfájlunkat mentsük el ANSI kódolással és nem kell tennünk semmi mást.

A kiírt fájl alapértelmezésként Utf-8 kódolású lesz.
"""



cimiro("A sorok azonos típusúak és szerkezetűek, nincsenek fehér sorok",".")

print("A szemely.txt ékezetes szövegfájl minden sorában szóközzel elválasztva")
print("egy vezetéknév, egy keresztnév, egy személyi szám és a születéskor nyert")
print("életbiztosítási összeg szerepel (Ft-ban megadva). Olvassuk be az adatokat egy listába!\n")


szemelyek=[] # ebbe a kétdimenziós listába olvassuk be a szövegfájlt

"""
A lista szerkezete:
Az i. személy vezetékneve: szemelyek1[i][0], sztring
keresztneve: szemelyek1[i][1], sztring
személyi száma: szemelyek1[i][2], sztring
biztosítási alapja: szemelyek1[i][3], egész szám

Mikor érdemes kétdimenziós listát (táblázatot) használni egy rekord leírásához külön listák helyett?
Elsősorban akkor, ha rendezésre is szükségünk van,
mert így bármelyik szempont (kulcs) szerint egy lépésben tudjuk rendezni a táblázatot úgy,
hogy az összetartozó értékek mindig együtt mozognak (lásd 6. Rendezés két szempont szerint).
Külön listák esetén minden cserét mindegyik listában külön kellene szinkronban megvalósítani.
Hátránya viszont, hogy a táblázat oszlopai így nem alkotnak listákat,
így nagyon hasznos listaműveleteket (pl. map, filter) nem tudunk közvetlenül alkalmazni.
Ezért mindig az adott feladat dönti el, milyen adatszerkezetet érdemes inkább használnunk.
Szerencsére lehetőségünk van a sima listák többdimenzióssá egyesítésére,
ill. a táblázatból oszloplisták elkészítésére is, ahol szükséges.
"""
```

```
betxt=open("szemely.txt")
# Ha a szövegfájl Utf-8 kódolású, akkor open("szemely.txt", encoding="utf-8" )

for sor in betxt: # egyszerűen végigmegyünk a megnyitott szövegfájl sorain
    darsor=sor.strip().split()      # megtisztítjuk és feldaraboljuk a sort
    szemely=[darsor[0], darsor[1], darsor[2], int(darsor[3])]
            # egy személy adatai: veznev, kernev, szemszam, biztalap (egész szám)
    szemelyek.append(szemely) # az elkészített szemely listát hozzáfűzzük a szemelyek listához

print(f'A szemely.txt fájl lezárása megtörtént: {betxt.closed}')
betxt.close() # kötelező a fájl kézi lezárása
print(f'A szemely.txt fájl lezárása megtörtént: {betxt.closed}')

"""
Sokan alkalmazzák a

with open("szemely.txt") as ff:
    for sor in ff:
        darsor=sor.strip().split()
        ...

szerkezetű beolvasást. Ennél a módszernél nem kellene kézi lezárás,
a blokk végén a program automatikusan lezárná a fájlt.
Viszont lehet, hogy épp emiatt felejtenénk el olyankor is a lezárást, amikor kötelező lenne.

Az érettségin nem szokott (csak nem nyomtatandó karakterekből álló)
fehér sor lenni a szövegfájlban, ezért itt nem is bonyolítjuk velük a beolvasást.
A 13. fejezetben a vers beolvasásánál láthatunk majd rá példát.
"""

print("\nA szemely.txt fájlból beolvasott adatlista:\n")
# formázott kiíratással:

for szemely in szemelyek:
    print(" ".join([szemely[0],szemely[1],szemely[2],str(szemely[3])]))
print()



cimiro("A sorok különböző típusúak vagy szerkezetűek, nincsenek fehér sorok",".")

print("A szemely.txt ékezetes szövegfájl egyik sorában szóközzel elválasztva")
print("egy személy vezetékneve, keresztneve, a következő sorban pedig  a személyi száma")
print("és a születéskor nyert életbiztosítási összege szerepel (Ft-ban megadva),")
print("majd így folytatódik a többi személyre is. Olvassuk be az adatokat egy listába!\n")

"""
Csupán annyi lesz a különbség, hogy a sor = next(betxt) utasítással
továbblépünk az adatblokk következő soraira, és a kívánt módon feldolgozzuk őket,
majd a for ciklus továbblép a következő blokkra.
"""

betxt = open("szemely1.txt")

szemelyek1=[]

for sor in betxt:
    darsor=sor.strip().split()
    szemely=[darsor[0], darsor[1]] # veznev, kernev
    sor = next(betxt) # továbblépünk a blokk második sorára
    darsor=sor.strip().split() # szemszam, biztalap
    szemely+=[darsor[0], int(darsor[1])] # a + művelet listák összefűzésére is alkalmas
    szemelyek1.append(szemely)

print(f'A szemely1.txt fájl lezárása megtörtént: {betxt.closed}')
betxt.close() # itt is kötelező a fájl kézi lezárása
print(f'A szemely1.txt fájl lezárása megtörtént: {betxt.closed}')

print("\nA szemely1.txt fájból beolvasott adatlista:\n")

for szemely in szemelyek1:
    print(" ".join([szemely[0],szemely[1],szemely[2],str(szemely[3])]))
print()
```

```
print("Ezután írassuk ki a képernyőre a vezetéknevet max. 10 karakterrel jobbra igazítva,")
print("két szóközzel elválasztva a keresztnév első három karakterét, ezután tabulátorokkal")
print("a személy nemét középre igazítva, születési évét, és életbiztosításának eredeti")
print("és az idei év végén várható értékét, ha tudjuk, hogy mindegyik év végén 3 %-kal nő!\n")


def nemfv(szsz):
    if szsz[:1]=="1" or szsz[:1]=="3":
        return "férfi"
    else:
        return "nő"

def szevfv(szsz):
    szev=int(szsz[1:3])
    if int(szsz[:1])<=2:
        return 1900 + szev
    else:
        return 2000 + szev



cimiro("A mai dátum lekérdezése",".")

# Ehhez kellett a datetime modul importálása.

maidatum=date.today()
ideiev=maidatum.year
aktho=maidatum.month
mainap=maidatum.day

print(f'A mai dátum: {maidatum}, év: {ideiev}, hó: {aktho}, nap: {mainap}\n')

most=datetime.now()
ideiev=most.year
aktho=most.month
mainap=most.day
aktora=most.hour
aktperc=most.minute
aktmp=most.second
aktmmp=most.microsecond

print(f'Az aktuális időpont: {most}, év: {ideiev}, hó: {aktho}, nap: {mainap},\n\
óra: {aktora}, perc: {aktperc}, másodpercp: {aktmp}, milliomodmásodperc: {aktmmp}.\n')


print("A személyek kért adatai a megfelelő formázással:\n")

"""
for i in range(len(szemelyek)):
    veznev=szemelyek[i][0][:min(10,len(szemelyek[i][0]))]
    kernev=szemelyek[i][1][:3]
    nem=nemfv(szemelyek[i][2])
    szev=szevfv(szemelyek[i][2])
    biztalap=szemelyek[i][3]
    ...
"""

for elem in szemelyek:
    veznev=elem[0][:min(10,len(szemelyek[i][0]))]
    kernev=elem[1][:3]
    nem=nemfv(elem[2])
    szev=szevfv(elem[2])
    biztalap=elem[3]
    evdb=ideiev-szev+1
    biztertek=biztalap*1.03**evdb
    print(f'{veznev:>10}  {kernev}\t{nem:^7}\t{szev}\t{biztalap:8,.0f} Ft,\t{biztertek:10,.0f} Ft.')
    # biztalap, biztertek: 8 ill. 10: szélesség, ,: ezredes tagolás, .0f: 0 tizedesjegy

print()
```

```
cimiro("6. Kiíratás szövegfájlba","=")

print("Írassuk ki az eredmeny.ki szövegfájlba az előző két feladat eredményét!")
print("Először írjuk felül az esetlegesen meglévő fájlt, és az első feladat után")
print("a második feladat eredménye pedig két üres sor után következzen.\n")

kitxt = open("eredmeny.ki","w") # "w" felülírja a régi fájlt, "a" hozzáfűzi az új szöveget

kitxt.write("Az a) feladat eredménye: \n\n");

"""
Csak át kell másolni a korábbi algoritmust, majd módosítani a kiíratást:
print helyett kitxt.write lesz,
mivel ennek a végén nincs automatikus sortörés, bele kell írni a \n-t.
Az f-stringek ugyanúgy működnek mint eddig.
"""

for elem in szamharmasok:
    a=elem[0]
    b=elem[1]
    c=elem[2]
    if haromszog_e(a,b,c):
        kitxt.write(f'{a:.2f}, {b:.2f} és {c:.2f} háromszöget határoznak meg.\n')
        k=a+b+c
        s=k/2
        t=sqrt(s*(s-a)*(s-b)*(s-c))
        rb=t/s
        rk=a*b*c/(4*t)
        kitxt.write(f'Kerülete: {k:.2f}, területe: {t:.2f},\
beírt körének sugara: {rb:.2f}, körülírt körének sugara: {rk:.2f}.\n\n\n')
    else:
        kitxt.write(f'{a:.2f}, {b:.2f} és {c:.2f} nem határoznak meg háromszöget.\n\n\n')

# A minden elágazásvégen szereplő három \n biztosítja a két üres sort is.


kitxt.write("A b) feladat eredménye: \n\n");

kitxt.write("A személyek kért adatai a megfelelő formázással:\n\n")

for elem in szemelyek:
    veznev=elem[0][:min(10,len(szemelyek[i][0]))]
    kernev=elem[1][:3]
    nem=nemfv(elem[2])
    szev=szevfv(elem[2])
    biztalap=elem[3]
    evdb=ideiev-szev+1
    biztertek=biztalap*1.03**evdb
    kitxt.write(f'{veznev:>10}  {kernev}\t{nem:^7}\t{szev}\t{biztalap:8,.0f} Ft,\t{biztertek:10,.0f}
Ft.\n')

kitxt.close()
print("A kiíratás és a szövegfájl lezárása sikeresen befejeződött.\n")

input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========== RESTART: C:\Python\KlInfoPy\BevGyak03\BevGyak03.py ==========

Beolvasás, kiírás: a képernyő és a szöveges fájlok kezelése
***********************************************************


1. Egész számok beolvasása a billentyűzetről
============================================

Kérek egy egész számot a billentyűzetről: 76

Kérek egy másik egész számot is a billentyűzetről: -86

76 és -86 távolsága a számegyenesen 162

A nagyobb szám 76, bináris alakja: 1001100,
oktális alakja: 114 és hexadecimális alakja: 4c

f'{nagyobb:x}' = 4c, típusa: <class 'str'>, hex(nagyobb) = 0x4c, típusa: <class 'str'>


2. Tetszőleges (2-36) számrendszerbeli számok decimális alakja
==============================================================

Kettes számrendszerben a számjegyek csak 0 és 1 lehetnek!
Kérek egy kettes számrendszerbeli egész számot: 11110100100
A megadott 11110100100 bináris szám alakja tízes számrendszerben: 1,956
Ez szám lesz: 1,956; négyzete: 3,825,936

Tizenhatos számrendszerben az alfabetikus számjegyek :
['10: A', '11: B', '12: C', '13: D', '14: E', '15: F']
Kérek egy tizenhatos számrendszerbeli egész számot: babfababa
A megadott babfababa hexadecimális szám alakja tízes számrendszerben: 50,129,975,994
Ez szám lesz: 50,129,975,994; négyzete: 2,513,014,493,159,016,288,036

36-os számrendszerben az alfabetikus számjegyek:
['10: A', '11: B', '12: C', '13: D', '14: E', '15: F', '16: G', '17: H', '18: I', '19: J']
['20: K', '21: L', '22: M', '23: N', '24: O', '25: P', '26: Q', '27: R', '28: S', '29: T']
['30: U', '31: V', '32: W', '33: X', '34: W', '35: Z']
Kérek egy harminchatos számrendszerbeli egész számot: python
A megadott python 36-os számrendszerbeli szám alakja tízes számrendszerben: 1,570,137,287
Ez szám lesz: 1,570,137,287; négyzete: 2,465,331,100,027,720,369

Kérem a számrendszer alapját (2-36): 14
14 alapú számrendszerben az alfabetikus számjegyek:
['10: A', '11: B', '12: C', '13: D']
Kérek egy 14 alapú számrendszerbeli egész számot: dada
A megadott dada 14 alapú számrendszerbeli szám alakja tízes számrendszerben: 37,824
Ez szám lesz: 37,824; négyzete: 1,430,654,976



2. Egy hosszú egész szám beolvasása a billentyűzetről
=====================================================


Egy billentyűzetről bekért szám számjegyeinek összeadogatása egyjegyű összegig
--------------------------------------------------------------------------------

Kérek egy tetszőlegesen nagy pozitív egész számot: 7777777777777

Részeredmény: 91
Részeredmény: 10
Részeredmény: 1
A végső számjegyösszeg: 1
```

```
Nézzük meg a BevGyak01-ben megismert True-ciklusokkal is!

Részeredmény: 91
Részeredmény: 10
Részeredmény: 1
A végső számjegyösszeg: 1



3. Valós számok beolvasása a billentyűzetről
=============================================

Kérek egy valós számot a billentyűzetről: 2.71

Kérek egy másik valós számot is a billentyűzetről: 3.14

2.71 és 3.14 távolsága a számegyenesen 0.430

A mértani közepük: 2.917



4. Műveletek a billentyűzetről beolvasott sztringekkel
======================================================


Hexadecimális RGB színkód elemzése
----------------------------------

Kérem a hexadecimális RGB-kódot, pl. A256FF: ffd700

A megadott szín hexadecimális kódja: #FFD700, decimális kódja: 255,215,0

A kód összetevői tízes számrendszerben: vörös: 255, zöld: 215, kék: 0.

A főszín: vörös, maximális értékének 100.00%-a,
a felhasznált színmennyiség 54.26%-a,
a teljes színmennyiségnek pedig 33.33%-a.



A billentyűzetről beolvasott szöveg magánhangzóinak füzére
----------------------------------------------------------

Írj be egy magyar mondatot:  A hűsítő tó fölé ezüst holdsugár kúszik.
A megadott mondat magánhangzóinak füzére: Aűíőööéeüouáúi.



Egész számok sorozatának beolvasása
------------------------------------

Kérem a számsorozat tagjait (legalább egyet) egy sorban szóközzel elválasztva: 12 -44 45 0 -5 68

A számsorozat: [12, -44, 45, 0, -5, 68], 6 tagból áll, 1 db 0 van köztük.
A legkisebb szám: -44, a legnagyobb: 68
A számok összege: 76, átlaga: 12.67
```

```
5. Beolvasás szövegfájlból
==========================


a) A 3szog.txt szövegfájl első sora tartalmazza a beolvasandó adatsorok számát.
--------------------------------------------------------------------------------

Ezután minden sorban szóközzel elválasztva egy-egy adathármas szerepel.
Olvassuk be egy kétdimenziós listába (tömbbe) az adathármasokat!
Ezután írassuk ki a képernyőre, hogy háromszöget határoznak-e meg,
és ha igen, akkor adjuk meg a kerületüket, a területüket,
a beírt és a körülírt körük sugarát 2 tizedesjeggyel!

5.30, 12.00 és 7.34 háromszöget határoznak meg.
Kerülete: 24.64, területe: 11.74, beírt körének sugara: 0.95, körülírt körének sugara: 9.94.

4.20, 55.00 és 44.00 nem határoznak meg háromszöget.

3.89, 7.98 és 5.72 háromszöget határoznak meg.
Kerülete: 17.59, területe: 10.40, beírt körének sugara: 1.18, körülírt körének sugara: 4.27.

2.23, 5.34 és 7.18 háromszöget határoznak meg.
Kerülete: 14.75, területe: 3.88, beírt körének sugara: 0.53, körülírt körének sugara: 5.51.


b) A szövegfájl első sora nem tartalmazza a beolvasandó adatsorok számát.
--------------------------------------------------------------------------

Alapértelmezett kódolás a gépünkön: cp1250


A sorok azonos típusúak és szerkezetűek, nincsenek fehér sorok
..............................................................

A szemely.txt ékezetes szövegfájl minden sorában szóközzel elválasztva
egy vezetéknév, egy keresztnév, egy személyi szám és a születéskor nyert
életbiztosítási összeg szerepel (Ft-ban megadva). Olvassuk be az adatokat egy listába!

A szemely.txt fájl lezárása megtörtént: False
A szemely.txt fájl lezárása megtörtént: True

A szemely.txt fájlból beolvasott adatlista:

Tündérszárnyú Írisz 28005162478 400000
Kígyóbűvölő Álmos 17612263333 300000
Űrhajós Ézsaiás 31208306789 500000
Úszó Őzike 40601038765 200000
Műlábú Üllő 15604031001 700000
Különc Údó 10302141111 50000
```

```
A sorok különböző típusúak vagy szerkezetűek, nincsenek fehér sorok
.....................................................................

A szemely.txt ékezetes szövegfájl egyik sorában szóközzel elválasztva
egy személy vezetékneve, keresztneve, a következő sorban pedig  a személyi száma
és a születéskor nyert életbiztosítási összege szerepel (Ft-ban megadva),
majd így folytatódik a többi személyre is. Olvassuk be az adatokat egy listába!

A szemely1.txt fájl lezárása megtörtént: False
A szemely1.txt fájl lezárása megtörtént: True

A szemely1.txt fájból beolvasott adatlista:

Tündérszárnyú Írisz 28005162478 400000
Kígyóbűvölő Álmos 17612263333 300000
Űrhajós Ézsaiás 31208306789 500000
Úszó Őzike 40601038765 200000
Műlábú Üllő 15604031001 700000
Különc Údó 10302141111 50000

Ezután írassuk ki a képernyőre a vezetéknevet max. 10 karakterrel jobbra igazítva,
két szóközzel elválasztva a keresztnév első három karakterét, ezután tabulátorokkal
a személy nemét középre igazítva, születési évét, és életbiztosításának eredeti
és az idei év végén várható értékét, ha tudjuk, hogy mindegyik év végén 3 %-kal nő!


A mai dátum lekérdezése
.......................

A mai dátum: 2022-11-10, év: 2022, hó: 11, nap: 10

Az aktuális időpont: 2022-11-10 16:25:09.905066, év: 2022, hó: 11, nap: 10,
óra: 16, perc: 25, másodpercp: 9, milliomodmásodperc: 905066.

A személyek kért adatai a megfelelő formázással:

      Tünd  Íri    nő    1980     400,000 Ft,     1,425,807 Ft.
      Kígy  Álm   férfi  1976     300,000 Ft,     1,203,569 Ft.
      Űrha  Ézs   férfi  2012     500,000 Ft,       692,117 Ft.
      Úszó  Őzi    nő    2006     200,000 Ft,       330,570 Ft.
      Műlá  Üll   férfi  1956     700,000 Ft,     5,072,150 Ft.
      Külö  Údó   férfi  1903      50,000 Ft,     1,735,549 Ft.


6. Kiíratás szövegfájlba
=========================

Írassuk ki az eredmeny.ki szövegfájlba az előző két feladat eredményét!
Először írjuk felül az esetlegesen meglévő fájlt, és az első feladat után
a második feladat eredménye pedig két üres sor után következzen.

A kiíratás és a szövegfájl lezárása sikeresen befejeződött.

A befejezéshez nyomd meg az ENTER billentyűt!
```

A 3szog.txt szövegfájl:

```
4
5.3 12 7.34
4.2 55 44
3.888 7.98 5.723
2.23 5.34 7.18
```

A szemely.txt szövegfájl:

```
Tündérszárnyú Írisz 28005162478 400000
Kígyóbűvölő Álmos 17612263333 300000
Űrhajós Ézsaiás 31208306789 500000
Úszó Őzike 40601038765 200000
Műlábú Üllő 15604031001 700000
Különc Údó 10302141111 50000
```

A szemely1.txt szövegfájl:

```
Tündérszárnyú Írisz
28005162478 400000
Kígyóbűvölő Álmos
17612263333 300000
Űrhajós Ézsaiás
31208306789 500000
Úszó Őzike
40601038765 200000
Műlábú Üllő
15604031001 700000
Különc Údó
10302141111 50000
```

Az eredmeny.ki szövegfájl:

```
Az a) feladat eredménye:

5.30, 12.00 és 7.34 háromszöget határoznak meg.
Kerülete: 24.64, területe: 11.74,beírt körének sugara: 0.95, körülírt körének sugara: 9.94.


4.20, 55.00 és 44.00 nem határoznak meg háromszöget.


3.89, 7.98 és 5.72 háromszöget határoznak meg.
Kerülete: 17.59, területe: 10.40,beírt körének sugara: 1.18, körülírt körének sugara: 4.27.


2.23, 5.34 és 7.18 háromszöget határoznak meg.
Kerülete: 14.75, területe: 3.88,beírt körének sugara: 0.53, körülírt körének sugara: 5.51.


A b) feladat eredménye:

A személyek kért adatai a megfelelő formázással:

      Tünd  Íri   nő    1980     400,000 Ft,     1,425,807 Ft.
      Kígy  Álm  férfi  1976     300,000 Ft,     1,203,569 Ft.
      Űrha  Ézs  férfi  2012     500,000 Ft,       692,117 Ft.
      Úszó  Őzi   nő    2006     200,000 Ft,       330,570 Ft.
      Műlá  Üll  férfi  1956     700,000 Ft,     5,072,150 Ft.
      Külö  Údó  férfi  1903      50,000 Ft,     1,735,549 Ft.
```

4. Elemi algoritmusok és a Python speciális függvényei

```
"""
4. Elemi algoritmusok és a Python speciális függvényei
@author Klemand66
"""

from math import *

def cimiro(cim,karakter):
    print("\n\n",cim,sep="") # két sort hagyunk ki a cím előtt
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

def cimiro2(cim,karakter): # kétsoros cím második sora, ez előtt nem hagyunk ki plusz sorokat
    print(cim)
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

cimiro("Elemi algoritmusok és a Python speciális függvényei","*")


cimiro("Elemi algoritmusok: összegzés, eldöntés, keresés, megszámolás,","=")
cimiro2("maximum-kiválasztás és kiválogatás","=")

print("Az elemi algoritmusok (programozási tételek) egy sorozat (vektor, tömb, lista)")
print("elemein végrehajtandó műveleteket végeznek el: ")
print(" - összegzés (általánosabban sorozatszámítás): összeadja, összefűzi, stb. az elemeket,")
print(" - eldöntés: meghatározza, hogy szerepel-e egy adott T tulajdonságú elem a sorozatban,")
print(" - keresés: egy adott T tulajdonságú elemet keres a sorozatban, és ha van, meg is ad egyet,")
print(" - megszámolás: meghatározza, hogy hány adott T tulajdonságú elem van a sorozatban,")
print(" - maximum-kiválasztás: meghatározza, hogy melyik a sorozat legnagyobb, leghosszabb, stb. eleme,")
print(" - kiválogatás: a T tulajdonságú elemek indexeit beteszi egy új sorozatba.\n\n")

# A számsorozat beolvasása a billentyűzetről az X tömbbe

print("Kérek egyesével min. 3 és max. 20 db egész számot! ");
print("Ha már nem akarsz több számot megadni, írj be egy betűt, nyomd meg a szóközt vagy simán
entert!\n");
# most vizsgálni fogjuk darabszámot és a típust is


def egesz_e(beirt):
    a=beirt.strip() # megtisztítjuk
    if len(a)==1:
        return a.isnumeric() # egyjegyű és numerikus karakter
    else:
        return(a[:1].isnumeric() or a[:1]=="+" or a[:1]=="-") and (a[1:].isnumeric())
        # (az első karaktere numerikus, + vagy - jel) és (az elsőt követőek numerikusak).

X=[]
i=1
folytat=True # A while True ciklus struktúrált formáját alkalmazzuk, nincs benne break utasítás

while i<=3 or (i<=20 and folytat): # 3 db egész számot mindenképp megkövetelünk
    beirt=input(f'{i:2}. szám: ')
    if egesz_e(beirt):       # a függvény csak a karaktereket vizsgálja, attól még sztring marad
        X.append(int(beirt)) # egész számmá alakítjuk és hozzáfűzzük a listához
        i+=1
    elif i>3:
        folytat=False

hossz=len(X)

print(f'\nA {hossz} elemű sorozat:\n{X}')
```

```
cimiro("1. Összegzés programozási tétel","=")

print("A megadott számsorozat elemeinek összegét határozzuk meg.\n");
# Mindig az aktuális összeghez adjuk a következő számot.

osszeg=0 # Az összegzés megkezdése előtt az aktuális összeg 0.

"""
for i in range(len(X)):
    print(f'{i+1:2}. részösszeg: {osszeg} + {X[i]} = ',end="")
    osszeg+=X[i]
    print(osszeg)
"""

for i,elem in enumerate(X):
    print(f'{i+1:2}. részösszeg: {osszeg} + {elem} = ',end="")
    osszeg+=elem
    print(osszeg)

print(f'\nA megadott számok összege: {osszeg}.\n')



cimiro("2. Eldöntés programozási tétel","=")

print("Megvizsgálja, hogy a megadott számok között van-e T tulajdonságú elem. ")
print("Ebben a programban a T tulajdonság azt jelenti,")
print("hogy az adott elem páros, de nem osztható 4-gyel.")
print("A vizsgálathoz célszerűen függvényt használunk.")
print("Így egy másik tulajdonság vizsgálata esetén csak a függvényt kell módosítani.\n")

def T(n):
    return n%4==2

# Az algoritmus:

i=0

while i < hossz and not T(X[i]):
    i+=1

van=(i<hossz)

"""
Akkor lépünk a ciklusmagba, ha még nem fogyott el a sorozat és az aktuális elem nem T tulajdonságú.
Ott nem teszünk mást, csak továbblépünk a sorozatban, hogy megvizsgáljuk a következő tagot.

A program két dolog miatt léphet ki a ciklusból
1. Végigért a sorozaton és az i értékét már hossz-ra növelte.
2. Talált T tulajdonságú elemet. Ha az utolsó volt az, már akkor sem növeli az i értéke hossz-ra.
Tehát az i alkalmas arra, hogy eldöntse a kérdést.
"""

print(f'a) while ciklussal: A megadott számok között van T tulajdonságú elem: {van}\n');

# Sokan ehhez is a "hivatalos" algoritmus helyett inkább kiugrásos for ciklust használnak:

van=False # meg kell adni a hamis kezdőértéket, hiszen már korábban is igazzá válhatott

"""
for i in range(hossz):   # itt i értéke legfeljebb hossz-1 lesz
    if T(X[i]):          # ha X egy eleme T tulajdonságú,
        van=True         # van értékét igazra állítjuk,
        break            # s mivel nincs további dolgunk a ciklussal, kiugrunk belőle
"""

for elem in X:           # végigjárjuk az X lista elemeit
    if T(elem):          # ha egy elem T tulajdonságú,
        van=True         # van értékét igazra állítjuk,
        break            # s mivel nincs további dolgunk a ciklussal, kiugrunk belőle
```

```
"""
Azért ez a kiugrás nem okozhat olyan zavarosságot és követhetetlenséget,
mint a programozás hajnalán a BASIC nyelv GOTO utasítása, amivel bárhonnan bárhova ugorhattunk.
Az ugrások kiküszöbölése miatt ragaszkodtak utána a struktúrált programozáshoz (lásd Pascal),
amelyben nincs GOTO, és egy elágazásnak vagy ciklusnak sem lehet két be- vagy kilépési pontja.

Viszont a break a GOTO-val ellentétben csak a ciklust szakítja meg, mindig a ciklus utáni sorra ugrik.
S bár a ciklusnak két kilépési pontja van, logikailag mégis egyszerűbb a while ciklusos algoritmusnál,
ahol a feltételben kell megadni az index figyelését és a (sokszor összetett) tulajdonság tagadását is.

Ez további problémát is felvet:
Ha a sorozatban nincs T tulajdonságú elem, akkor az utolsó elemnél i eléri a hossz értékét.
A program még visszamegy megvizsgálni a belépési feltételt.
Mivel először mindig az and művelet baloldalát elemzi, a megadott while i < hossz and not T(X[i]):
sorrendben kilép a ciklusból anélkül, hogy az X[i]-vel foglalkozna.
Szerencsére, mert X[hossz] már nem is létezik, hiszen i csak 1,2,...,hossz-1 lehet.

Viszont ha véletlenül fordítva írnánk a sorrendet: while not T(X[i]) and i < hossz:
akkor először szintén a baloldalt vizsgálva, X[hossz] nem megfelelő indexe miatt
"IndexError: list index out of range" hibajelzést kapnánk.
A while ciklusoknál és általában minden and művelet esetén tehát nagyon ügyelni kell a sorrendre!
"""

# Az eredmény elegánsabb kiíratással:
if van:
    valasz="van"
else:
    valasz="nincs"

print(f'b) for ciklussal: A megadott számok között {valasz} T tulajdonságú elem.\n')



cimiro("3. Keresés programozási tétel","=")

print("Megvizsgálja, hogy a megadott számok között van-e T tulajdonságú elem.")
print("Ha van, akkor megadja az első (vagy utolsó) T tulajdonságú elem sorszámát is.\n")

"""
Annyiban különbözik az eldöntéstől, hogy ha van T tulajdonságú elem a sorozatban,
akkor itt meg is adjuk az első (vagy utolsó) megfelelő elem indexét (esetleg az értékkel együtt).
De az indexet őrizzük meg, nem magát az értéket, hiszen az index alapján azt bármikor megkapjuk.
"""

# a) while ciklussal:

i=0

while i < hossz and not T(X[i]):
    i+=1

van=(i<hossz)

if van:
    sorszam=i

"""
Általában a Keresést szokás alkalmazni a Kiválasztás tétel helyett is,
amikor tudjuk, hogy van T tulajdonságú elem, bár akkor elég lenne ennyi:
while not T(X[i]):
    i+=1
sorszam=i
"""

if van:
    print(f'a) while ciklussal: {van}, első index: {sorszam}, érték: {X[sorszam]}')
else:
    print("a) while ciklussal: A megadott számok között nincs T tulajdonságú elem.")

print()
```

```
# b) for ciklussal:

van=False

"""
for i in range(hossz):
    if T(X[i]):
        van=True
        sorszam=i
        break
"""

for i,elem in enumerate(X):
    if T(elem):
        van=True
        sorszam=i
        break

if van:
    print(f'b) for ciklussal: {van}, első index: {sorszam}, érték: {X[sorszam]}')
else:
    print("b) for ciklussal: A megadott számok között nincs T tulajdonságú elem.")

print("\n")

"""
Az utolsó T tulajdonságú elemet is könnyen megkaphatjuk visszafelé lépkedve:
a) while ciklusnál i=hossz-1 kezdőértékkel és i-=1 léptetéssel, i>=0 feltétellel,
b) for ciklusnál range(hossz-1,-1,-1) argumentumokkal.
"""

# a) while ciklussal:

i=hossz-1

while i >-1 and not T(X[i]):
    i-=1

van=(i>=0)

if van:
    sorszam=i

if van:
    print(f'a) while ciklussal: {van}, utolsó index: {sorszam}, érték: {X[sorszam]}')
else:
    print("a) while ciklussal: A megadott számok között nincs T tulajdonságú elem.")

print()

# b) for ciklussal:

van=False

# Visszafelé haladásnál jó az öreg a háznál:

for i in range(hossz-1,-1,-1): # hossz -1 helyett simán  -1 írható
    if T(X[i]):
        van=True
        sorszam=i
        break

# Vigyázat! Ha most túlindexelünk, nem kapunk hibajelzést, hanem visszaugrik az utolsó elemre!

if van:
    print(f'b) for ciklussal: {van}, utolsó index: {sorszam}, érték: {X[sorszam]}\n')
else:
    print("b) for ciklussal: A megadott számok között nincs T tulajdonságú elem.\n")
```

```
cimiro("4. Megszámolás programozási tétel","=")

print("Meghatározza, hogy a megadott számok között hány T tulajdonságú elem van.\n")

# Mivel végig kell menni a sorozaton, itt mindig a for ciklust alkalmazzuk

Tdb = 0

"""
for i in range(hossz):
    if (T(X[i - 1])): # if T(X[i-1])==True
        Tdb+=1
"""

for elem in X:
    if (T(elem)): # if T(elem)==True
        Tdb+=1

print(f'A megadott számok között {Tdb} db T tulajdonságú elem van.')



cimiro("5. Maximum-kiválasztás programozási tétel","=")

cimiro("a) A maximális értékű elemek közül a legelső indexének meghatározása","-")

# Most nem foglalkkozunk a T tulajdonsággal.

maxindex=0
# Abból indulunk ki, hogy az első elem a legnagyobb, azután keresünk nála nagyobbat, ha van.

"""
for i in range(1,hossz):      # az első elemet nem hasonlítjuk önmagával, a másodikkal kezdünk
    if X[i]>X[maxindex]:      # Ha a vizsgált i. elem nagyobb mint az aktuális maximum,
        maxindex=i            # akkor az ő indexe lesz az új maxindex
"""

for i,elem in enumerate(X): # ez az első elemmel kezd, de ez sok vizet nem zavar
    if elem>X[maxindex]:      # Ha a vizsgált elem nagyobb mint az aktuális maximum,
        maxindex=i            # akkor az ő indexe lesz az új maxindex

"""
Vigyázat, az enumerate(X, start=1) nem azt jelenti, hogy a második elemmel kezdjen,
hanem azt, hogy az első elem indexe 0 helyett 1 legyen, ez pedig túlcsordulást okozhat.
"""

print(f'A maximális elem (vagy ezek közül az első) indexe: {maxindex}')
print(f'Az index ismeretében maga a maximális elem(ek) értéke is megadható: {X[maxindex]}')
```

```
cimiro("b) A T tulajdonsággal rendelkező (első) maximális elem indexének meghatározása","-")

# Most csak a T tulajdonságú elemeket vizsgáljuk.

Tmaxindex=None # None az abszolút semmi, csak nagy kezdőbetűvel szabad írni!
"""
Az első elem nem biztos, hogy T tulajdonságú, most nemlétező indexből indulunk ki.
Vigyázat! Általában a -1-et választjuk, de a Pythonban az létezik, az utolsó elemet jelenti!
"""

"""
for i in range(hossz):
    if T(X[i]):                  # ha X[i] T tulajdonságú elem,
        if Tmaxindex==None:      # akkor ha ő az első T tulajdonságú elem,
            Tmaxindex=i          # akkor a Tmaxindexnek i-vel már valós kezdőértéket adhatunk,
        elif X[i]>X[Tmaxindex]:  # különben megnézzük, hogy X[i] nagyobb-e mint az eddigi legnagyobb,
            Tmaxindex=i          # és ha igen, akkor i lesz az új Tmaxindex
"""

for i,elem in enumerate(X):
    if T(elem):                  # ha a vizsgált elem T tulajdonságú,
        if Tmaxindex==None:      # akkor ha ő az első T tulajdonságú elem,
            Tmaxindex=i          # akkor a Tmaxindexnek az i-vel már valós kezdőértéket adhatunk,
        elif elem>X[Tmaxindex]:  # különben megnézzük, hogy nagyobb-e mint az eddigi legnagyobb elem,
            Tmaxindex=i          # és ha igen, akkor az ő indexe lesz az új Tmaxindex


if Tmaxindex>=0:
    print(f'A T tulajdonságú elemek közül a(z első) legnagyobb elem indexe: {Tmaxindex}.')
    print(f'A sorszám ismeretében maga az elem is könnyen megadható: {X[Tmaxindex]}\n')
else:
    print("Nincs az adatok között T tulajdonságú elem.\n")



cimiro("6. Kiválogatás programozási tétel","=")

print("A T tulajdonságú elemek indexeinek kiválogatása egy új listába\n")

"""
Általában egyszerűbb csak az indexeket tárolni, mint magukat az elemeket!
Az index alapján mindig elérhetjük az elemet is.
"""

TXind=[]

for i in range(hossz):
    if T(X[i]):
        TXind.append(i)

"""
Mikor melyik az egyszerűbb:
for i,elem in enumerate(X):
    if T(elem):
        TXind.append(i)
"""

print(f'A kiválogatott indexek listája: {TXind}\n')

print("Az indexek és a hozzájuk tartozó elemek:\n")

szel=max(len(str(min(X))),len(str(max(X))))+1

"""
Az f-stringben a szám szélességének megadását is változóba tehetjük:
a legkisebb (lehet, hogy hosszú negatív) elem és a legnagyobb elem szélessége közül a nagyobb +1
"""

"""
for i in range(len(TXind)):
    print(f'index: {TXind[i]:2}, az elem értéke: {X[TXind[i]]:{szel}}')
"""

for elem in TXind:
    print(f'index: {elem:2}, az elem értéke: {X[elem]:{szel}}')

print()
```

```
cimiro("7. A Python speciális listafüggvényei","=")

"""
A Pythonnak vannak olyan beépített listafüggvényei, amelyek feleslegessé teszik a programozási tételek
alkalmazását. (Amelyekhez a functools modul importálása is kell, azokkal most nem foglalkozunk.)

Ehhez először a lambda függvéyt kell megismernünk.
Olyankor használjuk, amikor nem akarunk egy nevesített függvényt definiálni.
Legegyszerűbb alakjában a lambda kulcsszó után megadunk néhány változót, a lambda függvény argumentumait,
majd kettősponttal elválasztva egyetlen kifejezést, amit a változókra alkalmazni akarunk.
Pl. kif=lambda x,y,z: (x+y)*z
print(kif(2,3,4)), ami 20-at fog eredményezni.

Hagyományosan ez így nézne ki:
def fv(x,y,z):
    return (x+y)*z
kif=fv(2,3,4)
print(kif)

Tehát a lambda után elhagyhatjuk a def kulcsszót, a függvény nevét és a return kulcsszót is.

Ennél sokkal fontosabb azonban a listák esetén játszott szerepe.
Mi ezzel foglalkozunk, egy listából egy új lista megalkotására fogjuk használni segédfüggvényként.

A lambda kulcsszó után egy változóval hivatkozunk a lista tetszőleges elemére,
majd kettőspont után megadunk egy az elemre vonatkozó kifejezést,
ezután pedig vesszővel elválasztva a listát, amelyre alkalmazni akarjuk.

Pl. előállítunk egy listát, amely az X listánk minden elemének az abszolút értékét tartalmazza:
Y=list(map(lambda i: abs(i),X))
A map függvény feltérképezi az X lista elemeit, és végrehajtódik rajtuk a lambda művelet.
Tulajdonképpen képes helyettesíteni a for ciklust. A list() függvény pedig rögtön el is készíti az
előállított elemekből az új listát.
Megjegyzés: Az ilyen beépített alapfüggvényeknél, mint az abs, int, str, stb.
még a lambdát sem kell kiírni, elég annyi, hogy Y=list(map(abs,X)),
azaz elég a függvény neve és a lista, aminek az elemeire alkalmazza azt a feltérképező map függvény.

Hasonlóan fontos és hasznos a filter() szűrőfüggvény is.
Pl. a T tuljdonságú elemek kiválogatása: TX=list(filter(lambda i: T(i),X))
A T tuljdonságú elemek indexeinek kiválogatása: TXI=list(filter(lambda i: T(X[i]),range(len(X))))
"""
```

```
print("A lista elemeinek abszolút értékei:")
Y=list(map(abs,X))
print(f'Az eredeti lista: {X}\nAz elemek abszolút értékeinek listája: {Y}')


print("\nEgy sztringlista egész listává alakítása a map függvénnyel\n")
# Nagyon hasznos csoportos beolvasásnál.
S=input("Kérek öt egész számot szóközzel elválasztva: ")
Slista=S.strip().split()
Elista=list(map(int,Slista))
print(f'A megadott sztring: {S}\nA sztringlista: {Slista}\nAz egész lista: {Elista}')


print("\nA T tulajdonságú elemek kiválogatása:")
TX=list(filter(lambda i: T(i),X)) #TX csak a T tulajdonságú elemeket tartalmazza
print(TX)


print("\nA T tulajdonságú elemek indexeinek kiválogatása:")
TXI=list(filter(lambda i: T(X[i]),range(len(X)))) # most az indexek sorozatát szűrjük
print(TXI)


print("\nÖsszegzés: az X lista elemeinek összege:")
osszeg=sum(X) # a sum() függvény összegzi a számokat tartalmazó lista elemeit
print(osszeg)


print("\nEldöntés: van-e T tulajdonságú elem a listában:")
van=len(TX)>0 # a T tulajdonságú elemek listája nem üres
print(van)

# vagy másképp:
print("\nEldöntés másképp: van-e T tulajdonságú elem a listában:")
XT=list(map(lambda i: T(i),X)) # XT az X elemeihez rendeli, hogy megvan-e a T tulajdonságuk
van=True in (XT) # XT elemei között szerepel a True, nem mind False
print(f'{X}\n{XT}\n{van}')


print("\nKeresés: T tulajdonságú elem értéke és indexe (első találat) a listában:")
if len(TX)>0:
    Telem=TX[0] # az értéket kapjuk meg először,
    Tindex=X.index(Telem) # ahhoz keressük meg az indexet az eredeti listában
    print(f'Index: {Tindex}, érték: {Telem}.') # kiíratáskor az indexet szoktuk előre venni
else:
    print("A listában nincs T tulajdonságú elem.")


print("\nMegszámolás: a T tulajdonságú elemek száma a listában:")
Tdb=len(TX) # a T tulajdonságú elemek listájának hossza
print(Tdb)


print("\nMaximumkiválasztás: az X lista legnagyobb eleme és indexe (első találat):")
maxertek=max(X) # a max() függvény megadja a számokat tartalmazó lista legnagyobb elemét
maxindex=X.index(maxertek)
print(f'Index: {maxindex}, érték: {maxertek}.')


print("\nMaximumkiválasztás: az X lista legnagyobb T tulajdonságú eleme és indexe (első találat):")
if len(TX)>0:
    Tmaxertek=max(TX) # most a T tulajdonságú elemek listájának legnagyobb elemét keressük
    Tmaxindex=X.index(Tmaxertek)
    print(f'Index: {Tmaxindex}, érték: {Tmaxertek}.')
else:
    print("A listában nincs T tulajdonságú elem.")
```

```
cimiro("8. A Python speciális listafüggvényeinek alkalmazása kétdimenziós listáknál","=")

cimiro("a) A 3szog.txt szövegfájl feldolgozása","-")

def heron(hsz):
    a,b,c=hsz[0],hsz[1],hsz[2]
    s=(a+b+c)/2
    return sqrt(s*(s-a)*(s-b)*(s-c))

print("Olvassuk be szamharmasok kétdimenziós listába a 3szog.txt szövegfájlt,");
print("melynek első sora tartalmazza a beolvasandó adatsorok számát,")
print("ezután pedig minden sorban szóközzel elválasztva egy-egy adathármas szerepel.\n");

betxt = open("3szog.txt","r")

sordb=int(betxt.readline().strip())

szamharmasok=[]

for i in range(sordb):
    darsor=betxt.readline().strip().split()
    szamharmas=list(map(float,darsor))
    szamharmasok.append(szamharmas)

betxt.close()

print("\nSzűrjük ki a szamharmasok listából a haromszogek kétdimenziós listába azokat");
print("a rekordokat, amelyek háromszöget határoznak meg!");

haromszogek=list(filter(lambda x: x[0]+x[1]>x[1] and x[0]+x[2]>x[1] and x[1]+x[2]>x[0] ,szamharmasok))

for hsz in haromszogek:
    for oldal in hsz:
        print(f'{oldal:8.2f}',end="")
    print()

print("\nKészítsünk egy-egy listát, melyek a háromszögek kerületét, területét,");
print("a beírt és a körülírt körük sugarát tartalmazzák!");

keruletek=list(map(lambda x: x[0]+x[1]+x[2], haromszogek))
print(f'A kerületek:              ',end="")
for k in keruletek:
        print(f'{k:8.2f}',end="")
print()

teruletek=list(map(lambda x: heron(x), haromszogek))
print(f'A területek:              ',end="")
for t in teruletek:
        print(f'{t:8.2f}',end="")
print()

beirtkorok=list(map(lambda x: 2*heron(x)/(x[0]+x[1]+x[2]), haromszogek))
print(f'A beírt körök sugarai:    ',end="")
for bk in beirtkorok:
        print(f'{bk:8.2f}',end="")
print()

korulirtkorok=list(map(lambda x: x[0]*x[1]*x[2]/(4*heron(x)), haromszogek))
print(f'A körülírt körök sugarai: ',end="")
for kk in korulirtkorok:
        print(f'{kk:8.2f}',end="")
print()

print("\nKészítsünk hszadatok néven egy új kétdimenziós listát,");
print("mely a háromszögek oldalait és kiszámított adatait tartalmazza!");

hszadatok=[]
for i in range(len(haromszogek)):
    hszadatok.append(haromszogek[i]+[keruletek[i],teruletek[i],beirtkorok[i],korulirtkorok[i]])
    # az új 7 elemű rekord: az oldalak meglévő 3 elemű listája + a most elkésztett 4 elemű lista

for hsz in hszadatok:
    for adat in hsz:
        print(f'{adat:8.2f}',end="")
    print()
```

```
print("\nAdjuk meg annak a háromszögnek az adatait, melynek legnagyobb a területe.");

maxterhsz=max(hszadatok, key=lambda adat: adat[4])

for adat in maxterhsz:
    print(f'{adat:8.2f}',end="")
print()


print("\nAdjuk meg annak a háromszögnek az adatait, ");
print("melyben a legkisebb a legnagyobb és a legkisebb oldal eltérése!");

minelthsz=min(hszadatok, key=lambda x: max(x[0],x[1],x[2])-min(x[0],x[1],x[2]))

for adat in minelthsz:
    print(f'{adat:8.2f}',end="")
print()


cimiro("b) A szemely.txt szövegfájl feldolgozása","-")

def nemfv(szsz):
    if szsz[:1]=="1" or szsz[:1]=="3":
        return "férfi"
    else:
        return "nő"

print("A szemely.txt ékezetes szövegfájl minden sorában szóközzel elválasztva")
print("egy vezetéknév, egy keresztnév, egy személyi szám és a születéskor nyert")
print("életbiztosítási összeg szerepel (Ft-ban megadva).")
print("Olvassuk be az adatokat a kétdimenziós szemelyek listába!\n")

betxt=open("szemely.txt")

szemelyek=[]
for sor in betxt:
    darsor=sor.strip().split()
    szemely=[darsor[0], darsor[1], darsor[2], int(darsor[3])]
    szemelyek.append(szemely)

betxt.close()

print("Írjuk ki a képernyőre a legrövidebb nevű személy nevét és nemét!\n")

print("A min függvénnyel:")

lrn=min(szemelyek, key=lambda x:len(x[0]+x[1]))
print(f'{lrn[0]} {lrn[1]} {nemfv(lrn[2])}')

print("\nSzűréssel (ha több is van, az összeset):")

# A szűréshez ismernünk kell a minimális hosszat
minhossz=len(lrn[0]+lrn[1])

lrnlista=list(filter(lambda x: len(x[0]+x[1])== minhossz,szemelyek))

for lrn in lrnlista:
    print(f'{lrn[0]} {lrn[1]} {nemfv(lrn[2])}')

print("\nAdjuk össze a születéskori életbiztosítások értékeit!")
# A sum függvény használatához oszloplistát készítünk.
osszeg=sum(list(map(lambda x:x[3],szemelyek)))
print(f'Az összeg: {osszeg} Ft.\n')


input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak04\BevGyak04.py ==========================


Elemi algoritmusok és a Python speciális függvényei
***************************************************

Elemi algoritmusok: összegzés, eldöntés, keresés, megszámolás,
==============================================================

maximum-kiválasztás és kiválogatás
==================================

Az elemi algoritmusok (programozási tételek) egy sorozat (vektor, tömb, lista)
elemein végrehajtandó műveleteket végeznek el:
 - összegzés (általánosabban sorozatszámítás): összeadja, összefűzi, stb. az elemeket,
 - eldöntés: meghatározza, hogy szerepel-e egy adott T tulajdonságú elem a sorozatban,
 - keresés: egy adott T tulajdonságú elemet keres a sorozatban, és ha van, meg is ad egyet,
 - megszámolás: meghatározza, hogy hány adott T tulajdonságú elem van a sorozatban,
 - maximum-kiválasztás: meghatározza, hogy melyik a sorozat legnagyobb, leghosszabb, stb. eleme,
 - kiválogatás: a T tulajdonságú elemek indexeit beteszi egy új sorozatba.


Kérek egyesével min. 3 és max. 20 db egész számot!
Ha már nem akarsz több számot megadni, írj be egy betűt, nyomd meg a szóközt vagy simán entert!

 1. szám:   n
 1. szám: x
 1. szám: 3.14
 1. szám: 3
 2. szám: -14
 3. szám: 25
 4. szám: -28
 5. szám:  13
 6. szám: 0
 7. szám: 48
 8. szám:  48
 9. szám: 88
10. szám: -76
11. szám: 12
12. szám: 66
13. szám: n

A 12 elemű sorozat:
[3, -14, 25, -28, 13, 0, 48, 48, 88, -76, 12, 66]


1. Összegzés programozási tétel
===============================

A megadott számsorozat elemeinek összegét határozzuk meg.

 1. részösszeg: 0 + 3 = 3
 2. részösszeg: 3 + -14 = -11
 3. részösszeg: -11 + 25 = 14
 4. részösszeg: 14 + -28 = -14
 5. részösszeg: -14 + 13 = -1
 6. részösszeg: -1 + 0 = -1
 7. részösszeg: -1 + 48 = 47
 8. részösszeg: 47 + 48 = 95
 9. részösszeg: 95 + 88 = 183
10. részösszeg: 183 + -76 = 107
11. részösszeg: 107 + 12 = 119
12. részösszeg: 119 + 66 = 185

A megadott számok összege: 185.
```

```
2. Eldöntés programozási tétel
==============================

Megvizsgálja, hogy a megadott számok között van-e T tulajdonságú elem.
Ebben a programban a T tulajdonság azt jelenti,
hogy az adott elem páros, de nem osztható 4-gyel.
A vizsgálathoz célszerűen függvényt használunk.
Így egy másik tulajdonság vizsgálata esetén csak a függvényt kell módosítani.

a) while ciklussal: A megadott számok között van T tulajdonságú elem: True

b) for ciklussal: A megadott számok között van T tulajdonságú elem.



3. Keresés programozási tétel
==============================

Megvizsgálja, hogy a megadott számok között van-e T tulajdonságú elem.
Ha van, akkor megadja az első (vagy utolsó) T tulajdonságú elem sorszámát is.

a) while ciklussal: True, első index: 1, érték: -14

b) for ciklussal: True, első index: 1, érték: -14


a) while ciklussal: True, utolsó index: 11, érték: 66

b) for ciklussal: True, utolsó index: 11, érték: 66



4. Megszámolás programozási tétel
==================================

Meghatározza, hogy a megadott számok között hány T tulajdonságú elem van.

A megadott számok között 2 db T tulajdonságú elem van.


5. Maximum-kiválasztás programozási tétel
==========================================



a) A maximális értékű elemek közül a legelső indexének meghatározása
---------------------------------------------------------------------

A maximális elem (vagy ezek közül az első) indexe: 8
Az index ismeretében maga a maximális elem(ek) értéke is megadható: 88


b) A T tulajdonsággal rendelkező (első) maximális elem indexének meghatározása
-------------------------------------------------------------------------------

A T tulajdonságú elemek közül a(z első) legnagyobb elem indexe: 11.
A sorszám ismeretében maga az elem is könnyen megadható: 66
```

```
6. Kiválogatás programozási tétel
==================================

A T tulajdonságú elemek indexeinek kiválogatása egy új listába

A kiválogatott indexek listája: [1, 11]

Az indexek és a hozzájuk tartozó elemek:

index:  1, az elem értéke:  -14
index: 11, az elem értéke:   66



7. A Python speciális listafüggvényei
======================================

A lista elemeinek abszolút értékei:
Az eredeti lista: [3, -14, 25, -28, 13, 0, 48, 48, 88, -76, 12, 66]
Az elemek abszolút értékeinek listája: [3, 14, 25, 28, 13, 0, 48, 48, 88, 76, 12, 66]

Egy sztringlista egész listává alakítása a map függvénnyel

Kérek öt egész számot szóközzel elválasztva: 3 -14 25 -28 14
A megadott sztring: 3 -14 25 -28 14
A sztringlista: ['3', '-14', '25', '-28', '14']
Az egész lista: [3, -14, 25, -28, 14]

A T tulajdonságú elemek kiválogatása:
[-14, 66]

A T tulajdonságú elemek indexeinek kiválogatása:
[1, 11]

Összegzés: az X lista elemeinek összege:
185

Eldöntés: van-e T tulajdonságú elem a listában:
True

Eldöntés másképp: van-e T tulajdonságú elem a listában:
[3, -14, 25, -28, 13, 0, 48, 48, 88, -76, 12, 66]
[False, True, False, False, False, False, False, False, False, False, False, True]
True

Keresés: T tulajdonságú elem értéke és indexe (első találat) a listában:
Index: 1, érték: -14.

Megszámolás: a T tulajdonságú elemek száma a listában:
2

Maximumkiválasztás: az X lista legnagyobb eleme és indexe (első találat):
Index: 8, érték: 88.

Maximumkiválasztás: az X lista legnagyobb T tulajdonságú eleme és indexe (első találat):
Index: 11, érték: 66.
```

```
8. A Python speciális listafüggvényeinek alkalmazása kétdimenziós listáknál
============================================================================


a) A 3szog.txt szövegfájl feldolgozása
--------------------------------------

Olvassuk be szamharmasok kétdimenziós listába a 3szog.txt szövegfájlt,
melynek első sora tartalmazza a beolvasandó adatsorok számát,
ezután pedig minden sorban szóközzel elválasztva egy-egy adathármas szerepel.


Szűrjük ki a szamharmasok listából a haromszogek kétdimenziós listába azokat
a rekordokat, amelyek háromszöget határoznak meg!
    5.30   12.00    7.34
    3.89    7.98    5.72
    2.23    5.34    7.18

Készítsünk egy-egy listát, melyek a háromszögek kerületét, területét,
a beírt és a körülírt körük sugarát tartalmazzák!
A kerületek:                24.64   17.59   14.75
A területek:                11.74   10.40    3.88
A beírt körök sugarai:        0.95    1.18    0.53
A körülírt körök sugarai:     9.94    4.27    5.51

Készítsünk hszadatok néven egy új kétdimenziós listát,
mely a háromszögek oldalait és kiszámított adatait tartalmazza!
    5.30   12.00    7.34   24.64   11.74    0.95    9.94
    3.89    7.98    5.72   17.59   10.40    1.18    4.27
    2.23    5.34    7.18   14.75    3.88    0.53    5.51

Adjuk meg annak a háromszögnek az adatait, melynek legnagyobb a területe.
    5.30   12.00    7.34   24.64   11.74    0.95    9.94

Adjuk meg annak a háromszögnek az adatait,
melyben a legkisebb a legnagyobb és a legkisebb oldal eltérése!
    3.89    7.98    5.72   17.59   10.40    1.18    4.27


b) A szemely.txt szövegfájl feldolgozása
----------------------------------------

A szemely.txt ékezetes szövegfájl minden sorában szóközzel elválasztva
egy vezetéknév, egy keresztnév, egy személyi szám és a születéskor nyert
életbiztosítási összeg szerepel (Ft-ban megadva).
Olvassuk be az adatokat a kétdimenziós szemelyek listába!

Írjuk ki a képernyőre a legrövidebb nevű személy nevét és nemét!

A min függvénnyel:
Úszó Őzike nő

Szűréssel (ha több is van, az összeset):
Úszó Őzike nő
Különc Údó férfi

Adjuk össze a születéskori életbiztosítások értékeit!
Az összeg: 2150000 Ft.

A befejezéshez nyomd meg az ENTER billentyűt!
```

5. Rendezések: Klasszikus rendezési algoritmusok és a Python saját rendezései

```
"""
5. Rendezések: Klasszikus rendezési algoritmusok
és a Python saját rendezései
@author Klemand66
"""

def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

def cimiro2(cim,karakter):
    print(cim)
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")



cimiro("Rendezések: Klasszikus rendezési algoritmusok","*")
cimiro2("és a Python saját rendezései","*")

print("Először megadunk egy 10 elemű rendezetlen listát,")
print("majd különböző klasszikus algoritmusokkal nagyság szerint rendezzük az elemeket.")

# Az alaplista megadása
alaplista = [84, 48, 56, 76, 38, 44, 68, 28, 74, 18]

print(f'A rendezetlen alaplista:\n{alaplista}\n')



cimiro("1. Egyszerű cserés rendezés","=")

print("Az első lépésben az első elemet hasonlítjuk össze a másodikkal,")
print("és ha kell, kicseréljük őket, így a kisebb kerül az első helyre. ")
print("Utána az elsőt a harmadikkal,negyedikkel,..., utolsóval vetjük össze,")
print("így a belső ciklus lefutásakor a legkisebb elem lesz az első helyen.")
print("Vele már nem foglalkozunk, a második elemet fogjuk a mögötte levőkhöz hasonlítani,")
print("és ha kisebbet találunk nála, megcseréljük őket...")
print("A legutolsó lépésben az utolsó előtti elemet hasonlítjuk össze az utolsó elemmel.")
print("Hátránya a sok mozgatás és a távoli elemek cseréje.\n")

X=alaplista[:]

"""
Az X=alaplista értékadás esetén csak látszólag lenne két listánk,
mindkét név ugyanarra a memóriaterületre mutatna és együtt változnának, így csak az X fog.
"""
n=len(X)

for i in range(n-1):
    for j in range(i+1,n):
        if X[j]<X[i]:
            print(f'{X}   {X[i]}<-->{X[j]}',end="") # a folyamat követése, látványelem
            X[i],X[j]=X[j],X[i] # az elemek cseréje
            print(f'   {X}')
    print() # a belső ciklus minden lefutása után sort emelünk
print("\n")
```

```
cimiro("Két szempont szerinti rendezés egyszerű cserés rendezéssel","-")

print("Két szempont szerinti rendezésnél van egy főszempont és azon belül az mellékszempont.")
print("Pl. Rendezzük a sorozatot (listát) az utolsó számjegy és azon belül nagyság szerint.")
print("Ilyenkor először mellékszempont szerint kell rendeznünk, majd a főszempont szerint,")
print("de az már nem ronthatja el a mellészempont szerinti rendezést a főszemponton belül.")
print("Nézzük meg, mi történik, ha a már elkészült mellékszempont szerinti rendezés után")
print("a főszempont szerinti rendezésnél is az egyszerű cserés rendezést használjuk.\n")

for i in range(n-1):
    for j in range(i+1,n):
        if (X[j]%10)<(X[i]%10):
            print(f'{X}   {X[i]}<-->{X[j]}',end="")
            X[i],X[j]=X[j],X[i]
            print(f'   {X}')
    print()
print()

print("Láthatjuk, hogy a 6-os és 8-as utolsó számjegynél is elromlott a korábbi rendezettség,")
print("tehát az egyszerű cserés rendezés nem alkalmas a több szempont szerinti rendezésre.\n")
print("Ennek oka a távoli elemek cseréje.")

cimiro("2. Minimum-kiválasztásos rendezés","=")

print("Csak minden ciklus végén cseréljük ki az aktuális első elemet")
print("a mögötte levő legkisebbel, melynek az indexét követjük nyomon.")
print("Előnye, hogy kevesebb mozgatás történik, mint az egyszerű cserés rendezésnél.\n")

X=alaplista[:]

for i in range(n-1):
    min=i
    for j in range(i+1,n):
        if X[j]<X[min]:
            min=j
    print(f'{X}   {X[i]}<-->{X[min]}',end="")
    X[i],X[min]=X[min],X[i]
    print(f'   {X}')
    print()
print("\n")


cimiro("Két szempont szerinti rendezés minimum-kiválasztásos rendezéssel","-")

print("Rendezzük ismét a listánkat az utolsó számjegy és azon belül nagyság szerint.")
print("Nézzük meg, mi történik, ha a már elkészült mellékszempont szerinti rendezés után")
print("a főszempont szerinti rendezésnél is a buborékos rendezést használjuk.\n")

for i in range(n-1):
    min=i
    for j in range(i+1,n):
        if (X[j]%10)<(X[min]%10): # az utolsó számjegyeket hasonlítjuk össze
            min=j
    print(f'{X}   {X[i]}<-->{X[min]}',end="")
    X[i],X[min]=X[min],X[i]
    print(f'   {X}')
    print()
print()

print("Láthatjuk, hogy a 8-as utolsó számjegynél elromlott a korábbi rendezettség,")
print("tehát a minimum-kiválasztásos rendezés sem alkalmas a több szempont szerinti rendezésre.")
print("Hiába van ugyanis kevesebb csere, de azok itt is távoli elemek között történnek.")
```

```
cimiro("3. Buborékos rendezés","=")

print("Először az első elemtől az n-1-edikig eljutva minden elemet összehasonlítunk")
print("a mögötte levővel, és ha a sorrendjük nem jó, akkor megcseréljük őket.")
print("Ezáltal a kisebb elemek a sorozat eleje, a nagyobbak pedig a vége felé")
print("mozdulnak el, és a legnagyobb biztosan a végére kerül.")
print("A következő ciklusban már csak az n-2-ik elemig kell eljutnunk,")
print("végül pedig csak az elsőt kell összehasonlítanunk a másodikkal.\n")

X=alaplista[:]

for i in range(n-1,0,-1):
    for j in range(i):
        if X[j+1]<X[j]:
            print(f'{X}   {X[j]}<-->{X[j+1]}',end="")
            X[j],X[j+1]=X[j+1],X[j]
            print(f'   {X}')
    print()
print("\n")


cimiro("Két szempont szerinti rendezés buborékos rendezéssel","-")

print("Rendezzük ismét a listánkat az utolsó számjegy és azon belül nagyság szerint.")
print("Nézzük meg, mi történik, ha a már elkészült mellékszempont szerinti rendezés után")
print("a főszempont szerinti rendezésnél is a buborékos rendezést használjuk.\n")

for i in range(n-1,0,-1):
    for j in range(i):
        if (X[j+1]%10)<(X[j]%10):
            print(f'{X}   {X[j]}<-->{X[j+1]}',end="")
            X[j],X[j+1]=X[j+1],X[j]
            print(f'   {X}')
    print()
print()

print("Láthatjuk, hogy ez tényleg jól működik, mindenhol megmaradt a korábbi rendezettség,")
print("Tehát a buborékos rendezés alkalmas több szempont szerinti rendezésre is.")
print("Ez annak köszönhető, hogy mindig csak szomszédos elemeket cserélünk.\n")



cimiro("4. Beillesztéses rendezés","=")

print("A módszer lényege: ")
print("- Egyetlen elem mindig rendezett. ")
print("- Ha van egy rendezett részsorozatunk, ")
print("abba mindig be tudunk illeszteni egy új elemet a megfelelő helyre.")
print("Ezt a módszert alkalmazzuk pl. bizonyítványok névsorba rendezéséhez.\n")

X=alaplista[:]

for i in range(1,n):
    j=i
    while j>=1 and X[j]<X[j-1]:
        print(f'{X}   {X[j]}<-->{X[j-1]}',end="")
        X[j],X[j-1]=X[j-1],X[j]
        print(f'   {X}')
        j-=1
    print()

print()
```

```
cimiro("Két szempont szerinti rendezés beillesztéses rendezéssel","-")

print("Rendezzük ismét a listánkat az utolsó számjegy és azon belül nagyság szerint.")
print("Nézzük meg, mi történik, ha a már elkészült mellékszempont szerinti rendezés után")
print("a főszempont szerinti rendezésnél is a beillesztéses rendezést használjuk.\n")

for i in range(1,n):
    j=i
    while j>=1 and (X[j]%10)<(X[j-1]%10):
        print(f'{X}   {X[j]}<-->{X[j-1]}',end="")
        X[j],X[j-1]=X[j-1],X[j]
        print(f'   {X}')
        j-=1
    print()

print()

print("Láthatjuk, hogy ez tényleg jól működik, mindenhol megmaradt a korábbi rendezettség,")
print("Tehát a beillesztéses rendezés is alkalmas több szempont szerinti rendezésre is.")
print("Ez annak köszönhető, hogy itt is mindig csak szomszédos elemeket cserélünk.\n")


cimiro("5. A Python beépített rendezései","=")

"""
Kettő is van belőlük, a sorted() függvény és a sort() metódus.
Szerkezetük:

rendezett_lista=sorted(rendezendő_lista, reverse=False vagy True, key=kulcsfüggvény)
Rendezett listának megadhatjuk a rendezendő listát is, akkor helyben rendez.
Alapértelmezés a reverse=False, azt nem kell kiírni.
Ha fordított sorrendben akarunk rendezni, akkor reverse=True.
A kulcs sem kötelező, akkor használjuk, ha valamilyen különleges szempont szerint
akarjuk rendezni a listát, pl. az utolsó számjegy szerint.

rendezendő_lista.sort(reverse=False vagy True, key=kulcsfüggvény)
Ez mindig helyben rendez, nem tudunk neki új listanevet megadni.

Mindkét rendezés alkalmas a több szempont szerinti rendezésre is.
"""


cimiro("Rendezés a sorted() függvénnyel","-")

X=alaplista[:]

print(f'Az eredeti lista:\n{X}\n')

print("A nagyság szerint rendezett lista:")

RX=sorted(X)
print(RX)

print("\nA nagyság szerint fordítva rendezett lista:")

FRX=sorted(X,reverse=True)
print(FRX)

print("\nKulcs (az utolsó számjegy) szerint rendezett lista a lambda függvénnyel:")

KLRX=sorted(X,key=lambda i: i%10)
# A kulcsot általában helyben adjuk meg a lambda függvénnyel
print(KLRX)
```

```
print("\nKulcs (az utolsó számjegy) szerint rendezett lista nevesített függvénnyel:")

def utolsoszamjegy(i): # a függvény paramétere a rendezendő lista eleme
    return i%10

KNRX=sorted(X,key=utolsoszamjegy)
"""
A kulcsot egy általunk készített nevesített függvénnyel is megadhatjuk.
Vegyük észre, hogy csak a függyvény nevét kell megadni,
a paramétere automatikusan a rendezendő lista eleme.
Pl. a beépített abs() függvény estén is csak annyit írunk: key=abs
(csak a megadott pozitív számok esetén annak most nem sok értelme lenne).
"""
print(KNRX)

print(f'\nLássuk az eredeti listát, az nem változott meg, mindig új listába rendeztünk:\n{X}')

print("\nMost megváltoztatjuk, helyben rendezzük a listát a számok számjegyeinek összege szerint:")

#Itt érdemes függvényt írni, meg lehetne írni a lambda függvénnyel is, de áttekinthetetlen lenne.

def szamjegyosszeg(i):
    osszeg=0
    hossz=len(str(i))
    for j in range(hossz):
        osszeg+=i%10
        i=i//10
    return osszeg

X=sorted(X,key=szamjegyosszeg)
print(X)

print("\nEz sokkal szebb lesz, ha először nagyság szerint rendezzük a listát,\n\
majd utána a számjegyösszeg szerint:")

X=alaplista[:]
print(X)
X=sorted(X)
print(X)
X=sorted(X,key=szamjegyosszeg)
print(X)

print()


cimiro("Rendezés a sort() metódussal","-")

print("Rendezzük a számlistánkat először nagyság szerint,\n\
majd utána a számok 50-től való távolsága szerint.\n\
A fő szempont tehát az 50-től való távolság,\n\
ha ez azonos, akkor következnek majd nagyság szerint.\n")

def tav50(i):
    return abs(i-50)

X=alaplista[:]
print(X)
X.sort()
print(X)
X.sort(key=tav50)
print(X)


print("\n")
input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak05\BevGyak05.py ==========================


Rendezések: Klasszikus rendezési algoritmusok
*********************************************

és a Python saját rendezései
****************************

Először megadunk egy 10 elemű rendezetlen listát,
majd különböző klasszikus algoritmusokkal nagyság szerint rendezzük az elemeket.
A rendezetlen alaplista:
[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]


1. Egyszerű cserés rendezés
===========================

Az első lépésben az első elemet hasonlítjuk össze a másodikkal,
és ha kell, kicseréljük őket, így a kisebb kerül az első helyre.
Utána az elsőt a harmadikkal,negyedikkel,..., utolsóval vetjük össze,
így a belső ciklus lefutásakor a legkisebb elem lesz az első helyen.
Vele már nem foglalkozunk, a második elemet fogjuk a mögötte levőkhöz hasonlítani,
és ha kisebbet találunk nála, megcseréljük őket...
A legutolsó lépésben az utolsó előtti elemet hasonlítjuk össze az utolsó elemmel.
Hátránya a sok mozgatás és a távoli elemek cseréje.

[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]   84<-->48   [48, 84, 56, 76, 38, 44, 68, 28, 74, 18]
[48, 84, 56, 76, 38, 44, 68, 28, 74, 18]   48<-->38   [38, 84, 56, 76, 48, 44, 68, 28, 74, 18]
[38, 84, 56, 76, 48, 44, 68, 28, 74, 18]   38<-->28   [28, 84, 56, 76, 48, 44, 68, 38, 74, 18]
[28, 84, 56, 76, 48, 44, 68, 38, 74, 18]   28<-->18   [18, 84, 56, 76, 48, 44, 68, 38, 74, 28]

[18, 84, 56, 76, 48, 44, 68, 38, 74, 28]   84<-->56   [18, 56, 84, 76, 48, 44, 68, 38, 74, 28]
[18, 56, 84, 76, 48, 44, 68, 38, 74, 28]   56<-->48   [18, 48, 84, 76, 56, 44, 68, 38, 74, 28]
[18, 48, 84, 76, 56, 44, 68, 38, 74, 28]   48<-->44   [18, 44, 84, 76, 56, 48, 68, 38, 74, 28]
[18, 44, 84, 76, 56, 48, 68, 38, 74, 28]   44<-->38   [18, 38, 84, 76, 56, 48, 68, 44, 74, 28]
[18, 38, 84, 76, 56, 48, 68, 44, 74, 28]   38<-->28   [18, 28, 84, 76, 56, 48, 68, 44, 74, 38]

[18, 28, 84, 76, 56, 48, 68, 44, 74, 38]   84<-->76   [18, 28, 76, 84, 56, 48, 68, 44, 74, 38]
[18, 28, 76, 84, 56, 48, 68, 44, 74, 38]   76<-->56   [18, 28, 56, 84, 76, 48, 68, 44, 74, 38]
[18, 28, 56, 84, 76, 48, 68, 44, 74, 38]   56<-->48   [18, 28, 48, 84, 76, 56, 68, 44, 74, 38]
[18, 28, 48, 84, 76, 56, 68, 44, 74, 38]   48<-->44   [18, 28, 44, 84, 76, 56, 68, 48, 74, 38]
[18, 28, 44, 84, 76, 56, 68, 48, 74, 38]   44<-->38   [18, 28, 38, 84, 76, 56, 68, 48, 74, 44]

[18, 28, 38, 84, 76, 56, 68, 48, 74, 44]   84<-->76   [18, 28, 38, 76, 84, 56, 68, 48, 74, 44]
[18, 28, 38, 76, 84, 56, 68, 48, 74, 44]   76<-->56   [18, 28, 38, 56, 84, 76, 68, 48, 74, 44]
[18, 28, 38, 56, 84, 76, 68, 48, 74, 44]   56<-->48   [18, 28, 38, 48, 84, 76, 68, 56, 74, 44]
[18, 28, 38, 48, 84, 76, 68, 56, 74, 44]   48<-->44   [18, 28, 38, 44, 84, 76, 68, 56, 74, 48]

[18, 28, 38, 44, 84, 76, 68, 56, 74, 48]   84<-->76   [18, 28, 38, 44, 76, 84, 68, 56, 74, 48]
[18, 28, 38, 44, 76, 84, 68, 56, 74, 48]   76<-->68   [18, 28, 38, 44, 68, 84, 76, 56, 74, 48]
[18, 28, 38, 44, 68, 84, 76, 56, 74, 48]   68<-->56   [18, 28, 38, 44, 56, 84, 76, 68, 74, 48]
[18, 28, 38, 44, 56, 84, 76, 68, 74, 48]   56<-->48   [18, 28, 38, 44, 48, 84, 76, 68, 74, 56]

[18, 28, 38, 44, 48, 84, 76, 68, 74, 56]   84<-->76   [18, 28, 38, 44, 48, 76, 84, 68, 74, 56]
[18, 28, 38, 44, 48, 76, 84, 68, 74, 56]   76<-->68   [18, 28, 38, 44, 48, 68, 84, 76, 74, 56]
[18, 28, 38, 44, 48, 68, 84, 76, 74, 56]   68<-->56   [18, 28, 38, 44, 48, 56, 84, 76, 74, 68]

[18, 28, 38, 44, 48, 56, 84, 76, 74, 68]   84<-->76   [18, 28, 38, 44, 48, 56, 76, 84, 74, 68]
[18, 28, 38, 44, 48, 56, 76, 84, 74, 68]   76<-->74   [18, 28, 38, 44, 48, 56, 74, 84, 76, 68]
[18, 28, 38, 44, 48, 56, 74, 84, 76, 68]   74<-->68   [18, 28, 38, 44, 48, 56, 68, 84, 76, 74]

[18, 28, 38, 44, 48, 56, 68, 84, 76, 74]   84<-->76   [18, 28, 38, 44, 48, 56, 68, 76, 84, 74]
[18, 28, 38, 44, 48, 56, 68, 76, 84, 74]   76<-->74   [18, 28, 38, 44, 48, 56, 68, 74, 84, 76]

[18, 28, 38, 44, 48, 56, 68, 74, 84, 76]   84<-->76   [18, 28, 38, 44, 48, 56, 68, 74, 76, 84]
```

Két szempont szerinti rendezés egyszerű cserés rendezéssel

Két szempont szerinti rendezésnél van egy főszempont és azon belül az mellékszempont.
Pl. Rendezzük a sorozatot (listát) az utolsó számjegy és azon belül nagyság szerint.
Ilyenkor először mellékszempont szerint kell rendeznünk, majd a főszempont szerint,
de az már nem ronthatja el a mellészempont szerinti rendezést a főszemponton belül.
Nézzük meg, mi történik, ha a már elkészült mellékszempont szerinti rendezés után
a főszempont szerinti rendezésnél is az egyszerű cserés rendezést használjuk.

```
[18, 28, 38, 44, 48, 56, 68, 74, 76, 84]   18<-->44   [44, 28, 38, 18, 48, 56, 68, 74, 76, 84]

[44, 28, 38, 18, 48, 56, 68, 74, 76, 84]   28<-->56   [44, 56, 38, 18, 48, 28, 68, 74, 76, 84]
[44, 56, 38, 18, 48, 28, 68, 74, 76, 84]   56<-->74   [44, 74, 38, 18, 48, 28, 68, 56, 76, 84]

[44, 74, 38, 18, 48, 28, 68, 56, 76, 84]   38<-->56   [44, 74, 56, 18, 48, 28, 68, 38, 76, 84]
[44, 74, 56, 18, 48, 28, 68, 38, 76, 84]   56<-->84   [44, 74, 84, 18, 48, 28, 68, 38, 76, 56]

[44, 74, 84, 18, 48, 28, 68, 38, 76, 56]   18<-->76   [44, 74, 84, 76, 48, 28, 68, 38, 18, 56]

[44, 74, 84, 76, 48, 28, 68, 38, 18, 56]   48<-->56   [44, 74, 84, 76, 56, 28, 68, 38, 18, 48]
```

Láthatjuk, hogy a 6-os és 8-as utolsó számjegynél is elromlott a korábbi rendezettség,
tehát az egyszerű cserés rendezés nem alkalmas a több szempont szerinti rendezésre.

Ennek oka a távoli elemek cseréje.

2. Minimum-kiválasztásos rendezés
==========

Csak minden ciklus végén cseréljük ki az aktuális első elemet
a mögötte levő legkisebbel, melynek az indexét követjük nyomon.
Előnye, hogy kevesebb mozgatás történik, mint az egyszerű cserés rendezésnél.

```
[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]   84<-->18   [18, 48, 56, 76, 38, 44, 68, 28, 74, 84]

[18, 48, 56, 76, 38, 44, 68, 28, 74, 84]   48<-->28   [18, 28, 56, 76, 38, 44, 68, 48, 74, 84]

[18, 28, 56, 76, 38, 44, 68, 48, 74, 84]   56<-->38   [18, 28, 38, 76, 56, 44, 68, 48, 74, 84]

[18, 28, 38, 76, 56, 44, 68, 48, 74, 84]   76<-->44   [18, 28, 38, 44, 56, 76, 68, 48, 74, 84]

[18, 28, 38, 44, 56, 76, 68, 48, 74, 84]   56<-->48   [18, 28, 38, 44, 48, 76, 68, 56, 74, 84]

[18, 28, 38, 44, 48, 76, 68, 56, 74, 84]   76<-->56   [18, 28, 38, 44, 48, 56, 68, 76, 74, 84]

[18, 28, 38, 44, 48, 56, 68, 76, 74, 84]   68<-->68   [18, 28, 38, 44, 48, 56, 68, 76, 74, 84]

[18, 28, 38, 44, 48, 56, 68, 76, 74, 84]   76<-->74   [18, 28, 38, 44, 48, 56, 68, 74, 76, 84]

[18, 28, 38, 44, 48, 56, 68, 74, 76, 84]   76<-->76   [18, 28, 38, 44, 48, 56, 68, 74, 76, 84]
```

Két szempont szerinti rendezés minimum-kiválasztásos rendezéssel
---

Rendezzük ismét a listánkat az utolsó számjegy és azon belül nagyság szerint.
Nézzük meg, mi történik, ha a már elkészült mellékszempont szerinti rendezés után
a főszempont szerinti rendezésnél is a buborékos rendezést használjuk.

[18, 28, 38, 44, 48, 56, 68, 74, 76, 84] 18<-->44 [44, 28, 38, 18, 48, 56, 68, 74, 76, 84]

[44, 28, 38, 18, 48, 56, 68, 74, 76, 84] 28<-->74 [44, 74, 38, 18, 48, 56, 68, 28, 76, 84]

[44, 74, 38, 18, 48, 56, 68, 28, 76, 84] 38<-->84 [44, 74, 84, 18, 48, 56, 68, 28, 76, 38]

[44, 74, 84, 18, 48, 56, 68, 28, 76, 38] 18<-->56 [44, 74, 84, 56, 48, 18, 68, 28, 76, 38]

[44, 74, 84, 56, 48, 18, 68, 28, 76, 38] 48<-->76 [44, 74, 84, 56, 76, 18, 68, 28, 48, 38]

[44, 74, 84, 56, 76, 18, 68, 28, 48, 38] 18<-->18 [44, 74, 84, 56, 76, 18, 68, 28, 48, 38]

[44, 74, 84, 56, 76, 18, 68, 28, 48, 38] 68<-->68 [44, 74, 84, 56, 76, 18, 68, 28, 48, 38]

[44, 74, 84, 56, 76, 18, 68, 28, 48, 38] 28<-->28 [44, 74, 84, 56, 76, 18, 68, 28, 48, 38]

[44, 74, 84, 56, 76, 18, 68, 28, 48, 38] 48<-->48 [44, 74, 84, 56, 76, 18, 68, 28, 48, 38]

Láthatjuk, hogy a 8-as utolsó számjegynél elromlott a korábbi rendezettség,
tehát a minimum-kiválasztásos rendezés sem alkalmas a több szempont szerinti rendezésre.
Hiába van ugyanis kevesebb csere, de azok itt is távoli elemek között történnek.

3. Buborékos rendezés
==========

Először az első elemtől az n-1-edikig eljutva minden elemet összehasonlítunk
a mögötte levővel, és ha a sorrendjük nem jó, akkor megcseréljük őket.
Ezáltal a kisebb elemek a sorozat eleje, a nagyobbak pedig a vége felé
mozdulnak el, és a legnagyobb biztosan a végére kerül.
A következő ciklusban már csak az n-2-ik elemig kell eljutnunk,
végül pedig csak az elsőt kell összehasonlítanunk a másodikkal.

```
[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]   84<-->48   [48, 84, 56, 76, 38, 44, 68, 28, 74, 18]
[48, 84, 56, 76, 38, 44, 68, 28, 74, 18]   84<-->56   [48, 56, 84, 76, 38, 44, 68, 28, 74, 18]
[48, 56, 84, 76, 38, 44, 68, 28, 74, 18]   84<-->76   [48, 56, 76, 84, 38, 44, 68, 28, 74, 18]
[48, 56, 76, 84, 38, 44, 68, 28, 74, 18]   84<-->38   [48, 56, 76, 38, 84, 44, 68, 28, 74, 18]
[48, 56, 76, 38, 84, 44, 68, 28, 74, 18]   84<-->44   [48, 56, 76, 38, 44, 84, 68, 28, 74, 18]
[48, 56, 76, 38, 44, 84, 68, 28, 74, 18]   84<-->68   [48, 56, 76, 38, 44, 68, 84, 28, 74, 18]
[48, 56, 76, 38, 44, 68, 84, 28, 74, 18]   84<-->28   [48, 56, 76, 38, 44, 68, 28, 84, 74, 18]
[48, 56, 76, 38, 44, 68, 28, 84, 74, 18]   84<-->74   [48, 56, 76, 38, 44, 68, 28, 74, 84, 18]
[48, 56, 76, 38, 44, 68, 28, 74, 84, 18]   84<-->18   [48, 56, 76, 38, 44, 68, 28, 74, 18, 84]

[48, 56, 76, 38, 44, 68, 28, 74, 18, 84]   76<-->38   [48, 56, 38, 76, 44, 68, 28, 74, 18, 84]
[48, 56, 38, 76, 44, 68, 28, 74, 18, 84]   76<-->44   [48, 56, 38, 44, 76, 68, 28, 74, 18, 84]
[48, 56, 38, 44, 76, 68, 28, 74, 18, 84]   76<-->68   [48, 56, 38, 44, 68, 76, 28, 74, 18, 84]
[48, 56, 38, 44, 68, 76, 28, 74, 18, 84]   76<-->28   [48, 56, 38, 44, 68, 28, 76, 74, 18, 84]
[48, 56, 38, 44, 68, 28, 76, 74, 18, 84]   76<-->74   [48, 56, 38, 44, 68, 28, 74, 76, 18, 84]
[48, 56, 38, 44, 68, 28, 74, 76, 18, 84]   76<-->18   [48, 56, 38, 44, 68, 28, 74, 18, 76, 84]

[48, 56, 38, 44, 68, 28, 74, 18, 76, 84]   56<-->38   [48, 38, 56, 44, 68, 28, 74, 18, 76, 84]
[48, 38, 56, 44, 68, 28, 74, 18, 76, 84]   56<-->44   [48, 38, 44, 56, 68, 28, 74, 18, 76, 84]
[48, 38, 44, 56, 68, 28, 74, 18, 76, 84]   68<-->28   [48, 38, 44, 56, 28, 68, 74, 18, 76, 84]
[48, 38, 44, 56, 28, 68, 74, 18, 76, 84]   74<-->18   [48, 38, 44, 56, 28, 68, 18, 74, 76, 84]

[48, 38, 44, 56, 28, 68, 18, 74, 76, 84]   48<-->38   [38, 48, 44, 56, 28, 68, 18, 74, 76, 84]
[38, 48, 44, 56, 28, 68, 18, 74, 76, 84]   48<-->44   [38, 44, 48, 56, 28, 68, 18, 74, 76, 84]
[38, 44, 48, 56, 28, 68, 18, 74, 76, 84]   56<-->28   [38, 44, 48, 28, 56, 68, 18, 74, 76, 84]
[38, 44, 48, 28, 56, 68, 18, 74, 76, 84]   68<-->18   [38, 44, 48, 28, 56, 18, 68, 74, 76, 84]

[38, 44, 48, 28, 56, 18, 68, 74, 76, 84]   48<-->28   [38, 44, 28, 48, 56, 18, 68, 74, 76, 84]
[38, 44, 28, 48, 56, 18, 68, 74, 76, 84]   56<-->18   [38, 44, 28, 48, 18, 56, 68, 74, 76, 84]

[38, 44, 28, 48, 18, 56, 68, 74, 76, 84]   44<-->28   [38, 28, 44, 48, 18, 56, 68, 74, 76, 84]
[38, 28, 44, 48, 18, 56, 68, 74, 76, 84]   48<-->18   [38, 28, 44, 18, 48, 56, 68, 74, 76, 84]

[38, 28, 44, 18, 48, 56, 68, 74, 76, 84]   38<-->28   [28, 38, 44, 18, 48, 56, 68, 74, 76, 84]
[28, 38, 44, 18, 48, 56, 68, 74, 76, 84]   44<-->18   [28, 38, 18, 44, 48, 56, 68, 74, 76, 84]

[28, 38, 18, 44, 48, 56, 68, 74, 76, 84]   38<-->18   [28, 18, 38, 44, 48, 56, 68, 74, 76, 84]

[28, 18, 38, 44, 48, 56, 68, 74, 76, 84]   28<-->18   [18, 28, 38, 44, 48, 56, 68, 74, 76, 84]
```

```
Két szempont szerinti rendezés buborékos rendezéssel
----------------------------------------------------

Rendezzük ismét a listánkat az utolsó számjegy és azon belül nagyság szerint.
Nézzük meg, mi történik, ha a már elkészült mellékszempont szerinti rendezés után
a főszempont szerinti rendezésnél is a buborékos rendezést használjuk.

[18, 28, 38, 44, 48, 56, 68, 74, 76, 84]   38<-->44   [18, 28, 44, 38, 48, 56, 68, 74, 76, 84]
[18, 28, 44, 38, 48, 56, 68, 74, 76, 84]   48<-->56   [18, 28, 44, 38, 56, 48, 68, 74, 76, 84]
[18, 28, 44, 38, 56, 48, 68, 74, 76, 84]   68<-->74   [18, 28, 44, 38, 56, 48, 74, 68, 76, 84]
[18, 28, 44, 38, 56, 48, 74, 68, 76, 84]   68<-->76   [18, 28, 44, 38, 56, 48, 74, 76, 68, 84]
[18, 28, 44, 38, 56, 48, 74, 76, 68, 84]   68<-->84   [18, 28, 44, 38, 56, 48, 74, 76, 84, 68]

[18, 28, 44, 38, 56, 48, 74, 76, 84, 68]   28<-->44   [18, 44, 28, 38, 56, 48, 74, 76, 84, 68]
[18, 44, 28, 38, 56, 48, 74, 76, 84, 68]   38<-->56   [18, 44, 28, 56, 38, 48, 74, 76, 84, 68]
[18, 44, 28, 56, 38, 48, 74, 76, 84, 68]   48<-->74   [18, 44, 28, 56, 38, 74, 48, 76, 84, 68]
[18, 44, 28, 56, 38, 74, 48, 76, 84, 68]   48<-->76   [18, 44, 28, 56, 38, 74, 76, 48, 84, 68]
[18, 44, 28, 56, 38, 74, 76, 48, 84, 68]   48<-->84   [18, 44, 28, 56, 38, 74, 76, 84, 48, 68]

[18, 44, 28, 56, 38, 74, 76, 84, 48, 68]   18<-->44   [44, 18, 28, 56, 38, 74, 76, 84, 48, 68]
[44, 18, 28, 56, 38, 74, 76, 84, 48, 68]   28<-->56   [44, 18, 56, 28, 38, 74, 76, 84, 48, 68]
[44, 18, 56, 28, 38, 74, 76, 84, 48, 68]   38<-->74   [44, 18, 56, 28, 74, 38, 76, 84, 48, 68]
[44, 18, 56, 28, 74, 38, 76, 84, 48, 68]   38<-->76   [44, 18, 56, 28, 74, 76, 38, 84, 48, 68]
[44, 18, 56, 28, 74, 76, 38, 84, 48, 68]   38<-->84   [44, 18, 56, 28, 74, 76, 84, 38, 48, 68]

[44, 18, 56, 28, 74, 76, 84, 38, 48, 68]   18<-->56   [44, 56, 18, 28, 74, 76, 84, 38, 48, 68]
[44, 56, 18, 28, 74, 76, 84, 38, 48, 68]   28<-->74   [44, 56, 18, 74, 28, 76, 84, 38, 48, 68]
[44, 56, 18, 74, 28, 76, 84, 38, 48, 68]   28<-->76   [44, 56, 18, 74, 76, 28, 84, 38, 48, 68]
[44, 56, 18, 74, 76, 28, 84, 38, 48, 68]   28<-->84   [44, 56, 18, 74, 76, 84, 28, 38, 48, 68]

[44, 56, 18, 74, 76, 84, 28, 38, 48, 68]   18<-->74   [44, 56, 74, 18, 76, 84, 28, 38, 48, 68]
[44, 56, 74, 18, 76, 84, 28, 38, 48, 68]   18<-->76   [44, 56, 74, 76, 18, 84, 28, 38, 48, 68]
[44, 56, 74, 76, 18, 84, 28, 38, 48, 68]   18<-->84   [44, 56, 74, 76, 84, 18, 28, 38, 48, 68]

[44, 56, 74, 76, 84, 18, 28, 38, 48, 68]   56<-->74   [44, 74, 56, 76, 84, 18, 28, 38, 48, 68]
[44, 74, 56, 76, 84, 18, 28, 38, 48, 68]   76<-->84   [44, 74, 56, 84, 76, 18, 28, 38, 48, 68]

[44, 74, 56, 84, 76, 18, 28, 38, 48, 68]   56<-->84   [44, 74, 84, 56, 76, 18, 28, 38, 48, 68]


Láthatjuk, hogy ez tényleg jól működik, mindenhol megmaradt a korábbi rendezettség,
Tehát a buborékos rendezés alkalmas több szempont szerinti rendezésre is.
Ez annak köszönhető, hogy mindig csak szomszédos elemeket cserélünk.
```

```
4. Beillesztéses rendezés
=========================

A módszer lényege:
- Egyetlen elem mindig rendezett.
- Ha van egy rendezett részsorozatunk,
abba mindig be tudunk illeszteni egy új elemet a megfelelő helyre.
Ezt a módszert alkalmazzuk pl. bizonyítványok névsorba rendezéséhez.

[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]   48<-->84   [48, 84, 56, 76, 38, 44, 68, 28, 74, 18]

[48, 84, 56, 76, 38, 44, 68, 28, 74, 18]   56<-->84   [48, 56, 84, 76, 38, 44, 68, 28, 74, 18]

[48, 56, 84, 76, 38, 44, 68, 28, 74, 18]   76<-->84   [48, 56, 76, 84, 38, 44, 68, 28, 74, 18]

[48, 56, 76, 84, 38, 44, 68, 28, 74, 18]   38<-->84   [48, 56, 76, 38, 84, 44, 68, 28, 74, 18]
[48, 56, 76, 38, 84, 44, 68, 28, 74, 18]   38<-->76   [48, 56, 38, 76, 84, 44, 68, 28, 74, 18]
[48, 56, 38, 76, 84, 44, 68, 28, 74, 18]   38<-->56   [48, 38, 56, 76, 84, 44, 68, 28, 74, 18]
[48, 38, 56, 76, 84, 44, 68, 28, 74, 18]   38<-->48   [38, 48, 56, 76, 84, 44, 68, 28, 74, 18]

[38, 48, 56, 76, 84, 44, 68, 28, 74, 18]   44<-->84   [38, 48, 56, 76, 44, 84, 68, 28, 74, 18]
[38, 48, 56, 76, 44, 84, 68, 28, 74, 18]   44<-->76   [38, 48, 56, 44, 76, 84, 68, 28, 74, 18]
[38, 48, 56, 44, 76, 84, 68, 28, 74, 18]   44<-->56   [38, 48, 44, 56, 76, 84, 68, 28, 74, 18]
[38, 48, 44, 56, 76, 84, 68, 28, 74, 18]   44<-->48   [38, 44, 48, 56, 76, 84, 68, 28, 74, 18]

[38, 44, 48, 56, 76, 84, 68, 28, 74, 18]   68<-->84   [38, 44, 48, 56, 76, 68, 84, 28, 74, 18]
[38, 44, 48, 56, 76, 68, 84, 28, 74, 18]   68<-->76   [38, 44, 48, 56, 68, 76, 84, 28, 74, 18]

[38, 44, 48, 56, 68, 76, 84, 28, 74, 18]   28<-->84   [38, 44, 48, 56, 68, 76, 28, 84, 74, 18]
[38, 44, 48, 56, 68, 76, 28, 84, 74, 18]   28<-->76   [38, 44, 48, 56, 68, 28, 76, 84, 74, 18]
[38, 44, 48, 56, 68, 28, 76, 84, 74, 18]   28<-->68   [38, 44, 48, 56, 28, 68, 76, 84, 74, 18]
[38, 44, 48, 56, 28, 68, 76, 84, 74, 18]   28<-->56   [38, 44, 48, 28, 56, 68, 76, 84, 74, 18]
[38, 44, 48, 28, 56, 68, 76, 84, 74, 18]   28<-->48   [38, 44, 28, 48, 56, 68, 76, 84, 74, 18]
[38, 44, 28, 48, 56, 68, 76, 84, 74, 18]   28<-->44   [38, 28, 44, 48, 56, 68, 76, 84, 74, 18]
[38, 28, 44, 48, 56, 68, 76, 84, 74, 18]   28<-->38   [28, 38, 44, 48, 56, 68, 76, 84, 74, 18]

[28, 38, 44, 48, 56, 68, 76, 84, 74, 18]   74<-->84   [28, 38, 44, 48, 56, 68, 76, 74, 84, 18]
[28, 38, 44, 48, 56, 68, 76, 74, 84, 18]   74<-->76   [28, 38, 44, 48, 56, 68, 74, 76, 84, 18]

[28, 38, 44, 48, 56, 68, 74, 76, 84, 18]   18<-->84   [28, 38, 44, 48, 56, 68, 74, 76, 18, 84]
[28, 38, 44, 48, 56, 68, 74, 76, 18, 84]   18<-->76   [28, 38, 44, 48, 56, 68, 74, 18, 76, 84]
[28, 38, 44, 48, 56, 68, 74, 18, 76, 84]   18<-->74   [28, 38, 44, 48, 56, 68, 18, 74, 76, 84]
[28, 38, 44, 48, 56, 68, 18, 74, 76, 84]   18<-->68   [28, 38, 44, 48, 56, 18, 68, 74, 76, 84]
[28, 38, 44, 48, 56, 18, 68, 74, 76, 84]   18<-->56   [28, 38, 44, 48, 18, 56, 68, 74, 76, 84]
[28, 38, 44, 48, 18, 56, 68, 74, 76, 84]   18<-->48   [28, 38, 44, 18, 48, 56, 68, 74, 76, 84]
[28, 38, 44, 18, 48, 56, 68, 74, 76, 84]   18<-->44   [28, 38, 18, 44, 48, 56, 68, 74, 76, 84]
[28, 38, 18, 44, 48, 56, 68, 74, 76, 84]   18<-->38   [28, 18, 38, 44, 48, 56, 68, 74, 76, 84]
[28, 18, 38, 44, 48, 56, 68, 74, 76, 84]   18<-->28   [18, 28, 38, 44, 48, 56, 68, 74, 76, 84]
```

Két szempont szerinti rendezés beillesztéses rendezéssel

Rendezzük ismét a listánkat az utolsó számjegy és azon belül nagyság szerint.
Nézzük meg, mi történik, ha a már elkészült mellékszempont szerinti rendezés után
a főszempont szerinti rendezésnél is a beillesztéses rendezést használjuk.

```
[18, 28, 38, 44, 48, 56, 68, 74, 76, 84]   44<-->38   [18, 28, 44, 38, 48, 56, 68, 74, 76, 84]
[18, 28, 44, 38, 48, 56, 68, 74, 76, 84]   44<-->28   [18, 44, 28, 38, 48, 56, 68, 74, 76, 84]
[18, 44, 28, 38, 48, 56, 68, 74, 76, 84]   44<-->18   [44, 18, 28, 38, 48, 56, 68, 74, 76, 84]

[44, 18, 28, 38, 48, 56, 68, 74, 76, 84]   56<-->48   [44, 18, 28, 38, 56, 48, 68, 74, 76, 84]
[44, 18, 28, 38, 56, 48, 68, 74, 76, 84]   56<-->38   [44, 18, 28, 56, 38, 48, 68, 74, 76, 84]
[44, 18, 28, 56, 38, 48, 68, 74, 76, 84]   56<-->28   [44, 18, 56, 28, 38, 48, 68, 74, 76, 84]
[44, 18, 56, 28, 38, 48, 68, 74, 76, 84]   56<-->18   [44, 56, 18, 28, 38, 48, 68, 74, 76, 84]

[44, 56, 18, 28, 38, 48, 68, 74, 76, 84]   74<-->68   [44, 56, 18, 28, 38, 48, 74, 68, 76, 84]
[44, 56, 18, 28, 38, 48, 74, 68, 76, 84]   74<-->48   [44, 56, 18, 28, 38, 74, 48, 68, 76, 84]
[44, 56, 18, 28, 38, 74, 48, 68, 76, 84]   74<-->38   [44, 56, 18, 28, 74, 38, 48, 68, 76, 84]
[44, 56, 18, 28, 74, 38, 48, 68, 76, 84]   74<-->28   [44, 56, 18, 74, 28, 38, 48, 68, 76, 84]
[44, 56, 18, 74, 28, 38, 48, 68, 76, 84]   74<-->18   [44, 56, 74, 18, 28, 38, 48, 68, 76, 84]
[44, 56, 74, 18, 28, 38, 48, 68, 76, 84]   74<-->56   [44, 74, 56, 18, 28, 38, 48, 68, 76, 84]

[44, 74, 56, 18, 28, 38, 48, 68, 76, 84]   76<-->68   [44, 74, 56, 18, 28, 38, 48, 76, 68, 84]
[44, 74, 56, 18, 28, 38, 48, 76, 68, 84]   76<-->48   [44, 74, 56, 18, 28, 38, 76, 48, 68, 84]
[44, 74, 56, 18, 28, 38, 76, 48, 68, 84]   76<-->38   [44, 74, 56, 18, 28, 76, 38, 48, 68, 84]
[44, 74, 56, 18, 28, 76, 38, 48, 68, 84]   76<-->28   [44, 74, 56, 18, 76, 28, 38, 48, 68, 84]
[44, 74, 56, 18, 76, 28, 38, 48, 68, 84]   76<-->18   [44, 74, 56, 76, 18, 28, 38, 48, 68, 84]

[44, 74, 56, 76, 18, 28, 38, 48, 68, 84]   84<-->68   [44, 74, 56, 76, 18, 28, 38, 48, 84, 68]
[44, 74, 56, 76, 18, 28, 38, 48, 84, 68]   84<-->48   [44, 74, 56, 76, 18, 28, 38, 84, 48, 68]
[44, 74, 56, 76, 18, 28, 38, 84, 48, 68]   84<-->38   [44, 74, 56, 76, 18, 28, 84, 38, 48, 68]
[44, 74, 56, 76, 18, 28, 84, 38, 48, 68]   84<-->28   [44, 74, 56, 76, 18, 84, 28, 38, 48, 68]
[44, 74, 56, 76, 18, 84, 28, 38, 48, 68]   84<-->18   [44, 74, 56, 76, 84, 18, 28, 38, 48, 68]
[44, 74, 56, 76, 84, 18, 28, 38, 48, 68]   84<-->76   [44, 74, 56, 84, 76, 18, 28, 38, 48, 68]
[44, 74, 56, 84, 76, 18, 28, 38, 48, 68]   84<-->56   [44, 74, 84, 56, 76, 18, 28, 38, 48, 68]
```

Láthatjuk, hogy ez tényleg jól működik, mindenhol megmaradt a korábbi rendezettség,
Tehát a beillesztéses rendezés is alkalmas több szempont szerinti rendezésre is.
Ez annak köszönhető, hogy itt is mindig csak szomszédos elemeket cserélünk.

```
5. A Python beépített rendezései
================================



Rendezés a sorted() függvénnyel
-------------------------------

Az eredeti lista:
[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]

A nagyság szerint rendezett lista:
[18, 28, 38, 44, 48, 56, 68, 74, 76, 84]

A nagyság szerint fordítva rendezett lista:
[84, 76, 74, 68, 56, 48, 44, 38, 28, 18]

Kulcs (az utolsó számjegy) szerint rendezett lista a lambda függvénnyel:
[84, 44, 74, 56, 76, 48, 38, 68, 28, 18]

Kulcs (az utolsó számjegy) szerint rendezett lista nevesített függvénnyel:
[84, 44, 74, 56, 76, 48, 38, 68, 28, 18]

Lássuk az eredeti listát, az nem változott meg, mindig új listába rendeztünk:
[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]

Most megváltoztatjuk, helyben rendezzük a listát a számok számjegyeinek összege szerint:
[44, 18, 28, 56, 38, 74, 84, 48, 76, 68]

Ez sokkal szebb lesz, ha először nagyság szerint rendezzük a listát,
majd utána a számjegyösszeg szerint:
[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]
[18, 28, 38, 44, 48, 56, 68, 74, 76, 84]
[44, 18, 28, 38, 56, 74, 48, 84, 76, 68]



Rendezés a sort() metódussal
----------------------------

Rendezzük a számlistánkat először nagyság szerint,
majd utána a számok 50-től való távolsága szerint.
A fő szempont tehát az 50-től való távolság,
ha ez azonos, akkor következnek majd nagyság szerint.

[84, 48, 56, 76, 38, 44, 68, 28, 74, 18]
[18, 28, 38, 44, 48, 56, 68, 74, 76, 84]
[48, 44, 56, 38, 68, 28, 74, 76, 18, 84]


A befejezéshez nyomd meg az ENTER billentyűt!
```

6. Kétdimenziós listák (táblázatok) rendezése több szempont szerint

```
"""
6. Kétdimenziós listák (táblázatok) rendezése több szempont szerint
@author Klemand66
"""

def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")



cimiro("Kétdimenziós listák (táblázatok) rendezése több szempont szerint","*")


cimiro("A feladat","=")

print("Egy iskolai csapat tagjainak vezetéknevét, első keresztnevét,")
print("születési évét és lakhelyét tároljuk a csapat.txt fájl")
print("soraiban szóközökkel elválasztva (max. 20 fő).\n")

print("Rendezzük a listát születési év, azon belül lakhely nevének hossza, végül név szerint.")

print("Először olvassuk be az adatokat,")
print("majd rendezzük a listát először az utolsó mellékszempont (név),")
print("majd második lépésben a magasabb rendű mellékszempont (lakhely nevének hossza),")
print("végül a főszempont (születési év) szerint.")

print("Az eredményt írassuk ki az eredetihez hasonló formában")
print("a képernyőre és az Rcsapat_a.txt, Rcsapat_b.txt, Rcsapat_c.txt fájlokba is!\n")


cimiro("A csapat.txt szövegfájl beolvasása és a csapat nevű kétdimenziós listába tétele","=")

"""
Mivel rendezni fogunk, kétdimenziós listát választunk az adatok tárolására.
Ha a kétdimenziós lista azonos hosszúságú listák listája, akkor lényegében egy táblázat,
melynek elemei a táblázat sorai, amelyek egy objektomot leíró rekord mezői.
"""

csapat=[]

betxt = open("csapat.txt")

"""
Melyik beolvasást válasszuk?
Nem tudjuk, hány sorból áll a lista.
(A max. 20 fő olyan programozási nyelveknél szükséges,
ahol előre meg kell adni a lista (tömb) méretét, mint pl. a Java esetén.)
Mivel a sorok azonos szerkezetűek, ugyanolyan feldolgozást igényelnek,
a legegyszerűbb a sima bejárás, a for ciklussal.
Most vizsgáljuk az esetleges fehér sorokat is, amiket átlépünk.
"""

for sor in betxt:
    sor=sor.strip()
    if sor=="":     # ha a sor megtisztítása után semmi sem marad,
        continue    # átlépjük a sort
    darsor=sor.split() # ha nem fehér sor, maradunk és feldaraboljuk
    csapattag=[] # egyenként hozzáfűzve a mezőket, áttekinthetőbb lesz
    csapattag.append(darsor[0]+" "+darsor[1]) # név
    csapattag.append(int(darsor[2])) # születési év
    csapattag.append(darsor[3]) # lakhely
    csapat.append(csapattag) # az elkészített csapattag listát hozzáfűzzük a csapat listához

betxt.close()

csapatmentes=csapat[:] # mentés a későbbi rendezésekhez
```

```
print("Ellenőrzés, a beolvasott csapat tagjai a szövegfájlnak megfelelő formázással:\n")

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')

print()


cimiro("a) Hagyományos rendezés több lépésben","=")

print("Először bármelyik rendezési algoritmust alkalmazhatjuk,\n\
a név szerinti rendezést végezzük el most az egyszerű cserés rendezéssel.\n")

"""
Ha az ékezetes betűket a magyar ábécé szerint kívánjuk rendezni,
akkor külön függvényt kell írnunk az összehasonlításra,
viszont az érettségin nem kell ékezetes betűket kezelni,
ezért most a beépített ASCII-kód szerinti összehasonlítást alkalmazzuk.
"""

n=len(csapat)

for i in range(n-1):
    for j in range(i+1,n):
        if csapat[j][0]<csapat[i][0]: # a belső lista eleme szerint hasonlítunk,
            csapat[i],csapat[j]=csapat[j],csapat[i] # de a teljes belső listákat cseréljük

print("Ellenőrzés, a beolvasott csapat tagjai a név szerinti rendezés után:\n")

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')

print("\nMásodszor már csak a buborékos vagy a beillesztéses algoritmust alkalmazhatjuk,\n\
a lakhely nevének hossza szerinti rendezést végezzük el most a buborékos rendezéssel.\n")

for i in range(n-1,0,-1):
    for j in range(i):
        if len(csapat[j+1][2])<len(csapat[j][2]): # a belső lista elemére függvényt alkalmazunk
            csapat[j],csapat[j+1]=csapat[j+1],csapat[j]

print("Ellenőrzés, a beolvasott csapat tagjai a lakhely nevének hossza szerinti rendezés után:\n")

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')

print("\nHarmadszor megint csak a buborékos vagy a beillesztéses algoritmust alkalmazhatjuk,\n\
a születési évszerinti rendezést is végezzük el most a buborékos rendezéssel.\n")

for i in range(n-1,0,-1):
    for j in range(i):
        if csapat[j+1][1]<csapat[j][1]: # ismét a belső lista egyik eleme szerint hasonlítunk
            csapat[j],csapat[j+1]=csapat[j+1],csapat[j]

print("Ellenőrzés, a beolvasott csapat tagjai a születési év szerinti rendezés után:\n")

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')

#Kiíratás az Rcsapat_a.txt szöveges fájlba

kitxt = open("Rcsapat_a.txt","w")

for tag in csapat:
    kitxt.write(f'{tag[0]} {tag[1]} {tag[2]}\n') # plusz egy \n, nincs automatikus soremelés!

kitxt.close()
```

```
cimiro("b) Hagyományos rendezés egy lépésben","=")

print("Az eredeti csapatlista:\n")

csapat=csapatmentes[:]

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')

print("\nMost bármelyik rendezési algoritmust alkalmazhatjuk, hiszen egyetlen lépésben rendezünk,\n\
válasszuk hát a legegyszerűbbet, az egyszerű cserés rendezést.\n")

for i in range(n-1):
    for j in range(i+1,n):
        if csapat[j][1]<csapat[i][1]\
            or (csapat[j][1]==csapat[i][1] and len(csapat[j][2])<len(csapat[i][2]))\
            or ((csapat[j][1]==csapat[i][1] and len(csapat[j][2])==len(csapat[i][2]))\
                and csapat[j][0]<csapat[i][0]):
            csapat[i],csapat[j]=csapat[j],csapat[i]

"""
A \ a sorok végén a hosszú sorok tördelésére kell, de a \ után már megjegyzést sem lehet betenni.
Az algoritmus lényege:
Elsődlegesen a főszempont dönt,
ha a főszempont szerint egyenlő, akkor az első mellékszempont dönt,
ha még az is egyenlő, akkor a második mellékszempont dönt.
"""

print("Ellenőrzés, lássuk a medvét:\n")

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')

#Kiíratás az Rcsapat_b.txt szöveges fájlba

kitxt = open("Rcsapat_b.txt","w")

for tag in csapat:
    kitxt.write(f'{tag[0]} {tag[1]} {tag[2]}\n') # plusz egy \n, nincs automatikus soremelés!

kitxt.close()
```

```
cimiro("c) A Python beépített rendezésének használata","=")

print("Az eredeti csapatlista:\n")

csapat=csapatmentes[:]

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')


print("\nElőször a név szerinti rendezést végezzük el a sort() rendezéssel.\n")

csapat.sort(key=lambda tag: tag[0])

print("Ellenőrzés, a beolvasott csapat tagjai a név szerinti rendezés után:\n")

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')


print("\nMásodszor a lakhely nevének hossza szerinti rendezést végezzük el a sort() rendezéssel.\n")

csapat.sort(key=lambda tag: len(tag[2]))

print("Ellenőrzés, a beolvasott csapat tagjai a lakhely nevének hossza szerinti rendezés után:\n")

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')


print("\nHarmadszor a születési év szerinti rendezést is végezzük el a sort() rendezéssel.\n")

csapat.sort(key=lambda tag: tag[1])

print("Ellenőrzés, a beolvasott csapat tagjai a születési év szerinti rendezés után:\n")

for tag in csapat:
    print(f'{tag[0]} {tag[1]} {tag[2]}')


#Kiíratás az Rcsapat_c.txt szöveges fájlba

kitxt = open("Rcsapat_c.txt","w")

for tag in csapat:
    kitxt.write(f'{tag[0]} {tag[1]} {tag[2]}\n') # plusz egy \n, nincs automatikus soremelés!

kitxt.close()

print("\n")
input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak06\BevGyak06.py ==========================


Kétdimenziós listák (táblázatok) rendezése több szempont szerint
****************************************************************



A feladat
=========

Egy iskolai csapat tagjainak vezetéknevét, első keresztnevét,
születési évét és lakhelyét tároljuk a csapat.txt fájl
soraiban szóközökkel elválasztva (max. 20 fő).

Rendezzük a listát születési év, azon belül lakhely nevének hossza, végül név szerint.
Először olvassuk be az adatokat,
majd rendezzük a listát először az utolsó mellékszempont (név),
majd második lépésben a magasabb rendű mellékszempont (lakhely nevének hossza),
végül a főszempont (születési év) szerint.
Az eredményt írassuk ki az eredetihez hasonló formában
a képernyőre és az Rcsapat_a.txt, Rcsapat_b.txt, Rcsapat_c.txt fájlokba is!



A csapat.txt szövegfájl beolvasása és a csapat nevű kétdimenziós listába tétele
================================================================================

Ellenőrzés, a beolvasott csapat tagjai a szövegfájlnak megfelelő formázással:

Kovács Aladár 1999 Őriszentpéter
Illés Imre 1998 Körmend
Vas Eszter 2002 Szentgotthárd
Zala Eszter 2001 Körmend
Alföldi Gizella 1999 Egyházasrádóc
Badacsonyi Béla 2000 Körmend
Arany Írisz 2000 Szombathely
Hentes Oszkár 1998 Vasvár
Fürge Ferenc 1999 Vasvár
Hajós Annamária 2001 Csákánydoroszló
Bíró Judit 1998 Körmend
Deák Dóra 1999 Körmend
Nagy János 1998 Körmend
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Zala Ibolya 2001 Szalafő
```

```
a) Hagyományos rendezés több lépésben
======================================

Először bármelyik rendezési algoritmust alkalmazhatjuk,
a név szerinti rendezést végezzük el most az egyszerű cserés rendezéssel.

Ellenőrzés, a beolvasott csapat tagjai a név szerinti rendezés után:

Alföldi Gizella 1999 Egyházasrádóc
Arany Írisz 2000 Szombathely
Badacsonyi Béla 2000 Körmend
Bíró Judit 1998 Körmend
Deák Dóra 1999 Körmend
Fürge Ferenc 1999 Vasvár
Hajós Annamária 2001 Csákánydoroszló
Hentes Oszkár 1998 Vasvár
Illés Imre 1998 Körmend
Kovács Aladár 1999 Őriszentpéter
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Nagy János 1998 Körmend
Vas Eszter 2002 Szentgotthárd
Zala Eszter 2001 Körmend
Zala Ibolya 2001 Szalafő

Másodszor már csak a buborékos vagy a beillesztéses algoritmust alkalmazhatjuk,
a lakhely nevének hossza szerinti rendezést végezzük el most a buborékos rendezéssel.

Ellenőrzés, a beolvasott csapat tagjai a lakhely nevének hossza szerinti rendezés után:

Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Fürge Ferenc 1999 Vasvár
Hentes Oszkár 1998 Vasvár
Badacsonyi Béla 2000 Körmend
Bíró Judit 1998 Körmend
Deák Dóra 1999 Körmend
Illés Imre 1998 Körmend
Nagy János 1998 Körmend
Zala Eszter 2001 Körmend
Zala Ibolya 2001 Szalafő
Arany Írisz 2000 Szombathely
Alföldi Gizella 1999 Egyházasrádóc
Kovács Aladár 1999 Őriszentpéter
Vas Eszter 2002 Szentgotthárd
Hajós Annamária 2001 Csákánydoroszló
```

```
Harmadszor megint csak a buborékos vagy a beillesztéses algoritmust alkalmazhatjuk,
a születési évszerinti rendezést is végezzük el most a buborékos rendezéssel.

Ellenőrzés, a beolvasott csapat tagjai a születési év szerinti rendezés után:

Hentes Oszkár 1998 Vasvár
Bíró Judit 1998 Körmend
Illés Imre 1998 Körmend
Nagy János 1998 Körmend
Fürge Ferenc 1999 Vasvár
Deák Dóra 1999 Körmend
Alföldi Gizella 1999 Egyházasrádóc
Kovács Aladár 1999 Őriszentpéter
Badacsonyi Béla 2000 Körmend
Arany Írisz 2000 Szombathely
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Zala Eszter 2001 Körmend
Zala Ibolya 2001 Szalafő
Hajós Annamária 2001 Csákánydoroszló
Vas Eszter 2002 Szentgotthárd


b) Hagyományos rendezés egy lépésben
==================================

Az eredeti csapatlista:

Kovács Aladár 1999 Őriszentpéter
Illés Imre 1998 Körmend
Vas Eszter 2002 Szentgotthárd
Zala Eszter 2001 Körmend
Alföldi Gizella 1999 Egyházasrádóc
Badacsonyi Béla 2000 Körmend
Arany Írisz 2000 Szombathely
Hentes Oszkár 1998 Vasvár
Fürge Ferenc 1999 Vasvár
Hajós Annamária 2001 Csákánydoroszló
Bíró Judit 1998 Körmend
Deák Dóra 1999 Körmend
Nagy János 1998 Körmend
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Zala Ibolya 2001 Szalafő

Most bármelyik rendezési algoritmust alkalmazhatjuk, hiszen egyetlen lépésben rendezünk,
válasszuk hát a legegyszerűbbet, az egyszerű cserés rendezést.

Ellenőrzés, lássuk a medvét:

Hentes Oszkár 1998 Vasvár
Bíró Judit 1998 Körmend
Illés Imre 1998 Körmend
Nagy János 1998 Körmend
Fürge Ferenc 1999 Vasvár
Deák Dóra 1999 Körmend
Alföldi Gizella 1999 Egyházasrádóc
Kovács Aladár 1999 Őriszentpéter
Badacsonyi Béla 2000 Körmend
Arany Írisz 2000 Szombathely
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Zala Eszter 2001 Körmend
Zala Ibolya 2001 Szalafő
Hajós Annamária 2001 Csákánydoroszló
Vas Eszter 2002 Szentgotthárd
```

```
c) A Python beépített rendezésének használata
==============================================

Az eredeti csapatlista:

Kovács Aladár 1999 Őriszentpéter
Illés Imre 1998 Körmend
Vas Eszter 2002 Szentgotthárd
Zala Eszter 2001 Körmend
Alföldi Gizella 1999 Egyházasrádóc
Badacsonyi Béla 2000 Körmend
Arany Írisz 2000 Szombathely
Hentes Oszkár 1998 Vasvár
Fürge Ferenc 1999 Vasvár
Hajós Annamária 2001 Csákánydoroszló
Bíró Judit 1998 Körmend
Deák Dóra 1999 Körmend
Nagy János 1998 Körmend
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Zala Ibolya 2001 Szalafő

Először a név szerinti rendezést végezzük el a sort() rendezéssel.

Ellenőrzés, a beolvasott csapat tagjai a név szerinti rendezés után:

Alföldi Gizella 1999 Egyházasrádóc
Arany Írisz 2000 Szombathely
Badacsonyi Béla 2000 Körmend
Bíró Judit 1998 Körmend
Deák Dóra 1999 Körmend
Fürge Ferenc 1999 Vasvár
Hajós Annamária 2001 Csákánydoroszló
Hentes Oszkár 1998 Vasvár
Illés Imre 1998 Körmend
Kovács Aladár 1999 Őriszentpéter
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Nagy János 1998 Körmend
Vas Eszter 2002 Szentgotthárd
Zala Eszter 2001 Körmend
Zala Ibolya 2001 Szalafő

Másodszor a lakhely nevének hossza szerinti rendezést végezzük el a sort() rendezéssel.

Ellenőrzés, a beolvasott csapat tagjai a lakhely nevének hossza szerinti rendezés után:

Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Fürge Ferenc 1999 Vasvár
Hentes Oszkár 1998 Vasvár
Badacsonyi Béla 2000 Körmend
Bíró Judit 1998 Körmend
Deák Dóra 1999 Körmend
Illés Imre 1998 Körmend
Nagy János 1998 Körmend
Zala Eszter 2001 Körmend
Zala Ibolya 2001 Szalafő
Arany Írisz 2000 Szombathely
Alföldi Gizella 1999 Egyházasrádóc
Kovács Aladár 1999 Őriszentpéter
Vas Eszter 2002 Szentgotthárd
Hajós Annamária 2001 Csákánydoroszló
```

```
Harmadszor a születési év szerinti rendezést is végezzük el a sort() rendezéssel.

Ellenőrzés, a beolvasott csapat tagjai a születési év szerinti rendezés után:

Hentes Oszkár 1998 Vasvár
Bíró Judit 1998 Körmend
Illés Imre 1998 Körmend
Nagy János 1998 Körmend
Fürge Ferenc 1999 Vasvár
Deák Dóra 1999 Körmend
Alföldi Gizella 1999 Egyházasrádóc
Kovács Aladár 1999 Őriszentpéter
Badacsonyi Béla 2000 Körmend
Arany Írisz 2000 Szombathely
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Zala Eszter 2001 Körmend
Zala Ibolya 2001 Szalafő
Hajós Annamária 2001 Csákánydoroszló
Vas Eszter 2002 Szentgotthárd


A befejezéshez nyomd meg az ENTER billentyűt!



Az Rcsapat_a.txt, Rcsapat_b.txt és Rcsapat_c.txt szövegfájlok:


Hentes Oszkár 1998 Vasvár
Bíró Judit 1998 Körmend
Illés Imre 1998 Körmend
Nagy János 1998 Körmend
Fürge Ferenc 1999 Vasvár
Deák Dóra 1999 Körmend
Alföldi Gizella 1999 Egyházasrádóc
Kovács Aladár 1999 Őriszentpéter
Badacsonyi Béla 2000 Körmend
Arany Írisz 2000 Szombathely
Mázsa Lenke 2001 Ják
Mázsa Nóra 2001 Ják
Zala Eszter 2001 Körmend
Zala Ibolya 2001 Szalafő
Hajós Annamária 2001 Csákánydoroszló
Vas Eszter 2002 Szentgotthárd
```

7. A maximumkiválasztás általánosítása: versenykiértékelések

```
"""
7. A maximumkiválasztás általánosítása: versenykiértékelések
@author Klemand66
"""

def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

def cimiro2(cim,karakter):
    print(cim)
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

cimiro("A maximumkiválasztás általánosítása: versenykiértékelések","*")

print("A verseny.txt szövegfájl sorai szóközzel elválasztva egy verseny")
print("max. 100 indulójának azonosítóit és az elért pontszámait tartalmazzák.")
# Az indulók maximális száma Pythonban nem érdekes
print("Határozzuk meg a k legmagasabb pontszámot, és az ezeket elérő versenyzők azonosítóit.")
print("Határozzuk meg az n legjobb eredményt elérő versenyzők helyezéseit (ugrásokkal).")
print("Adjuk meg egy kiválasztott versenyző elhelyezkedését a kétféle listában.")
print("Pl. XXX a 3. legmagasabb pontszámot érte el, amivel a 6-9. helyen végzett.")

# A feladatot megoldjuk rendezés nélkül és rendezéssel is.


cimiro("1. A feladat megoldása rendezés nélkül","=")

cimiro("A verseny.txt szövegfájl beolvasása és az azonositok, ill. a pontok listába tétele","-")

betxt = open("verseny.txt")

# Ha nem fogunk rendezni, akkor célszerűbb külön listákba olvasni az adatokat
azonositok=[]
pontok=[]

# Tudjuk, hogy a sorok azonos szerkezetűek és nincsenek fehér sorok

for sor in betxt:
    darsor=sor.strip().split()
    azonositok.append(darsor[0])      # azonosító
    pontok.append(int(darsor[1]))     # pontszám

betxt.close()

print("Ellenőrzés, a beolvasott listák a szövegfájlnak megfelelő formázással:\n")

for i in range(len(azonositok)):
    print(f'{azonositok[i]} {pontok[i]}')

print()

minpont=min(pontok)
maxpont=max(pontok)
maxpontok=[maxpont]
hatar=maxpont # most már csak a nála kisebbek legnagyobbikát keressük

while hatar>minpont:
    maxpont=minpont-1
    for pont in pontok:
        if pont>maxpont and pont<hatar:
            maxpont=pont
    maxpontok.append(maxpont)
    hatar=maxpont
```

```
k=int(input(f'\nAdd meg a k értékét (minimum 1 és maximum {len(maxpontok)}): '))
# Ilyenkor az érettségin feltételezhetjük, hogy a megadott szám megfelelő, nem kell vizsgálni.

print(f'\nA(z) {k} legmagasabb pontszám: {maxpontok[:k]}\n')

m=min(k, len(maxpontok))
for pont in maxpontok[:m]:
    print(f'{pont} pontot elért versenyzők:')
    for i,p in enumerate(pontok):
        if p==pont:
            print(f'\t{azonositok[i]}')


n=int(input(f'\nAdd meg az n értékét (minimum 1 és maximum {len(azonositok)}): '))

print(f'\nA(z) {n} legjobb eredményt elérő versenyzők helyezései (ugrásokkal):')

hatar=max(pontok)+1
hely=0
helydb=0

while helydb<n:
    maxpont=minpont-1
    hely=helydb+1
    for pont in pontok:
        if pont>maxpont and pont<hatar:
            maxpont=pont
    for i,pont in enumerate(pontok):
        if pont==maxpont:
            helydb+=1
            print(f'{hely:2}. helyezett {azonositok[i]:12}: {pontok[i]} pont')
    hatar=maxpont


az=input(f'\nAdd meg a kiválasztott versenyző azonosítóját: ')
print()

if az in azonositok:
    i=azonositok.index(az)
    pont=pontok[i]
    ponthely=maxpontok.index(pont)+1
    nagyobbak=list(filter(lambda x:x>pont,pontok))
    egyenlok=list(filter(lambda x:x==pont,pontok))
    print(f'{az} pontszáma {pont}. A(z) {ponthely}. legmagasabb pontszámot érte el, \
amivel a(z) {len(nagyobbak)+1}-{len(nagyobbak)+len(egyenlok)}. helyen végzett.\n')
else:
    print("Nincs ilyen azonosítójú versenyző.\n")



cimiro("2. A feladat megoldása rendezéssel","=")


cimiro("A verseny.txt szövegfájl beolvasása és a versenyzok nevű kétdimenziós listába tétele","-")
"""
Rendezés estén nem szerencsés külön listákba tenni az adatokat, hiszen szinkronban kell
bennük végezni minden elemcserét, ami a kétdimenziós listánál értelemszerűen teljesül.
"""

versenyzok=[]

betxt = open("verseny.txt")

for sor in betxt:
    darsor=sor.strip().split()
    versenyzo=[darsor[0],int(darsor[1])]
              # azonosító pontszám
    versenyzok.append(versenyzo)

betxt.close()

print("Ellenőrzés, a beolvasott verseny.txt adatsorai a szövegfájlnak megfelelő formázással:\n")

for v in versenyzok:
    print(f'{v[0]} {v[1]}')

print()
```

```
cimiro("A k legmagasabb pontszám, és az ezeket elérő versenyzők meghatározása,","-")


# Rendezzük a verseny listát pontok szerint csökkenően és azon belül azonosító szerint.

versenyzok.sort(key= lambda v:v[0])
versenyzok.sort(reverse=True,key= lambda v:v[1])

maxpontok=[]

for v in versenyzok: # a versenyzőket pontszám szerint sorba rendeztük
    if v[1] not in maxpontok: # v[1] az aktuális versenyző pontszáma
        maxpontok.append(v[1])

k=int(input(f'\nAdd meg a k értékét (minimum 1 és maximum {len(maxpontok)}): '))

print(f'\nA(z) {k} legmagasabb pontszám: {maxpontok[:k]}\n')

for pont in maxpontok[:k]:
    szurt=list(filter(lambda v : v[1]==pont,versenyzok))
    print(f'{pont} pontot elért versenyzők:')
    for v in szurt:
        print(f'\t{v[0]}')


n=int(input(f'\nAdd meg az n értékét (minimum 1 és maximum {len(azonositok)}): '))
print(f'\nA(z) {n} legjobb eredményt elérő versenyzők helyezései (ugrásokkal):\n')

hely=1
for pont in maxpontok:
    if hely>n:
        break
    else:
        szurt=list(filter(lambda v : v[1]==pont,versenyzok))
        for v in szurt:
            print(f'{hely:2}. helyezett {v[0]:12} {v[1]} pont')
        hely+=len(szurt)


az=input(f'\nAdd meg a kiválasztott versenyző azonosítóját: ')

"""
Kétdimenziós listában az elemek is listák, a táblázat sorai, azaz "vízszintes listák".
Az azonos típusú mezők, pl. az azonosítók nem alkotnak "függőleges listát",
mint korábban is említettem, nem alklmazhatóak rájuk az in, az index(), stb. lekérdezések,
ezért egy ciklust írunk az azonosítót tartalmazó elem, a vízszintes lista indexének meghatározására,
ill. a további hozzá tartozó mezők (a versenyző adatai) lekérdezésére.
"""

for i,elem in enumerate(versenyzok):
    if elem[0]==az:
        azin=i
        azp=elem[1]
        van=True
        break

if van:
    print(versenyzok.count(v[1]==azp))
    h=[] # listába tesszük a vele megegyező pontszámú versenyzők indexeit
    for i,elem in enumerate(versenyzok):
        if elem[1]==azp:
            h.append(i)
    print(f'\n{az} pontszáma {azp}. Az {maxpontok.index(azp)+1}. legmagasabb pontszámot érte el,')
    print(f'amivel a(z) {h[0]+1}-{h[-1]+1}. helyen végzett.\n')
else:
    print("Nincs ilyen azonosítójú versenyző.\n")

print("\nEllenőrzés, a rendezett verseny lista adatsorai:\n")

for v in versenyzok:
    print(f'{v[0]} {v[1]}')

print("\n")
input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak07\BevGyak07.py ==========================


A maximumkiválasztás általánosítása: versenykiértékelések
*********************************************************

A verseny.txt szövegfájl sorai szóközzel elválasztva egy verseny
max. 100 indulójának azonosítóit és az elért pontszámait tartalmazzák.
Határozzuk meg a k legmagasabb pontszámot, és az ezeket elérő versenyzők azonosítóit.
Határozzuk meg az n legjobb eredményt elérő versenyzők helyezéseit (ugrásokkal).
Adjuk meg egy kiválasztott versenyző elhelyezkedését a kétféle listában.
Pl. XXX a 3. legmagasabb pontszámot érte el, amivel a 6-9. helyen végzett.


1. A feladat megoldása rendezés nélkül
======================================


A verseny.txt szövegfájl beolvasása és az azonositok, ill. a pontok listába tétele
----------------------------------------------------------------------------------

Ellenőrzés, a beolvasott listák a szövegfájlnak megfelelő formázással:

zx428 67
3puputeve 72
matrix77 54
aladar 54
asdf 54
FFF 77
mz/x 67
123456 80
oroszlan 67
b66 66
fizikus 77
kishableany 66
zold 80
iqbetyar 54
karoly 72
c786 72


Add meg a k értékét (minimum 1 és maximum 6): 3

A(z) 3 legmagasabb pontszám: [80, 77, 72]

80 pontot elért versenyzők:
        123456
        zold
77 pontot elért versenyzők:
        FFF
        fizikus
72 pontot elért versenyzők:
        3puputeve
        karoly
        c786
```

```
Add meg az n értékét (minimum 1 és maximum 16): 10

A(z) 10 legjobb eredményt elérő versenyzők helyezései (ugrásokkal):
 1. helyezett 123456      : 80 pont
 1. helyezett zold        : 80 pont
 3. helyezett FFF         : 77 pont
 3. helyezett fizikus     : 77 pont
 5. helyezett 3puputeve   : 72 pont
 5. helyezett karoly      : 72 pont
 5. helyezett c786        : 72 pont
 8. helyezett zx428       : 67 pont
 8. helyezett mz/x        : 67 pont
 8. helyezett oroszlan    : 67 pont

Add meg a kiválasztott versenyző azonosítóját: fizikus

fizikus pontszáma 77. A(z) 2. legmagasabb pontszámot érte el, amivel a(z) 3-4. helyen végzett.



2. A feladat megoldása rendezéssel
==================================


A verseny.txt szövegfájl beolvasása és a versenyzok nevű kétdimenziós listába tétele
--------------------------------------------------------------------------------

Ellenőrzés, a beolvasott verseny.txt adatsorai a szövegfájlnak megfelelő formázással:

zx428 67
3puputeve 72
matrix77 54
aladar 54
asdf 54
FFF 77
mz/x 67
123456 80
oroszlan 67
b66 66
fizikus 77
kishableany 66
zold 80
iqbetyar 54
karoly 72
c786 72



A k legmagasabb pontszám, és az ezeket elérő versenyzők meghatározása,
----------------------------------------------------------------------


Add meg a k értékét (minimum 1 és maximum 6): 3

A(z) 3 legmagasabb pontszám: [80, 77, 72]

80 pontot elért versenyzők:
        123456
        zold
77 pontot elért versenyzők:
        FFF
        fizikus
72 pontot elért versenyzők:
        3puputeve
        c786
        karoly
```

```
Add meg az n értékét (minimum 1 és maximum 16): 10

A(z) 10 legjobb eredményt elérő versenyzők helyezései (ugrásokkal):

 1. helyezett 123456       80 pont
 1. helyezett zold         80 pont
 3. helyezett FFF          77 pont
 3. helyezett fizikus      77 pont
 5. helyezett 3puputeve    72 pont
 5. helyezett c786         72 pont
 5. helyezett karoly       72 pont
 8. helyezett mz/x         67 pont
 8. helyezett oroszlan     67 pont
 8. helyezett zx428        67 pont

Add meg a kiválasztott versenyző azonosítóját: fizikus
0

fizikus pontszáma 77. Az 2. legmagasabb pontszámot érte el,
amivel a(z) 3-4. helyen végzett.


Ellenőrzés, a rendezett verseny lista adatsorai:

123456 80
zold 80
FFF 77
fizikus 77
3puputeve 72
c786 72
karoly 72
mz/x 67
oroszlan 67
zx428 67
b66 66
kishableany 66
aladar 54
asdf 54
iqbetyar 54
matrix77 54


A befejezéshez nyomd meg az ENTER billentyűt!
```

8. A leghosszabb adott tulajdonságú részsorozat keresése: a leghosszabb prímsorozat egy véletlen sorozatban

```
"""
8. A leghosszabb adott tulajdonságú részsorozat keresése:
a leghosszabb prímsorozat egy véletlen sorozatban
@author Klemand66
"""
from random import *


def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

def cimiro2(cim,karakter):
    print(cim)
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")



cimiro("A leghosszabb adott tulajdonságú részsorozat keresése:","*")
cimiro2("a leghosszabb prímsorozat egy véletlen sorozatban","*")

def T(n): # az adott T tulajdonság: prím-e (az általánosság kedvéért jelöljük T-vel)
    return n!=1 and n%2!=0 and n%3!=0 and n%5!=0 and n%7!=0
    # csak a maximális 100 gyökéig kell vizsgálnunk az oszthatóságot

cimiro("500 db 1 és 100 közötti véletlen egész szám előállítása","-")

X=[]
for i in range(500):
    X.append(randrange(1,101))

for i, szam in enumerate(X):
    print(f'{szam:3}', end="")
    if T(szam):
        print("*\t", end="")
    else:
        print(" \t",end="")
    if (i+1)%10==0:
        print()
    if (i+1)%100==0:
        print()
```

```
cimiro("A leghosszabb prímsorozat megkeresése és kiíratása","=")

aktkezdet=None # az aktuális T tulajdonságú sorozat kezdetének indexe
akthossz=0     # az aktuális T tulajdonságú sorozat hossza
maxkezdet=None # a maximális hosszúságú T tulajdonságú sorozat kezdetének indexe
maxhossz=0     # a maximális hosszúságú T tulajdonságú sorozat hossza

for i,szam in enumerate(X):
    if T(szam):                     # a vizsgált elem T tulajdonságú
        if aktkezdet==None:         # ha ő az első az aktuális T tulajdonságú sorozatban,
            aktkezdet=i             # akkor az aktuális kezdőindex az ő indexe lesz
        akthossz+=1                 # ha 0 volt, 1 lesz, különben csak folytatódik a jó sorozat
        if akthossz>maxhossz:       # ha nagyobb lett a jó sorozat hossza az eddigi leghosszabbnál,
            maxkezdet=aktkezdet     # akkor ez lesz az új maximális jó sorozat,
            maxhossz=akthossz       # a maxhossz is ennek a hossza lesz
    else:                           # megszakad a T tulajdonságú sorozat
        aktkezdet=None
        akthossz=0

"""
Amikor egy T tulajdonságú (jó) sorozat hosszabb lesz, mint az addigi maximális hosszúságú,
akkor ameddig csak tart a jó sorozat, akkor minden lépésben eggyel nagyobb lesz az akthossz
az érvényes maxhossznál. Ezért újra beállítja a maxkezdetet a saját aktkezdetének,
(ami nem lenne szükséges, de emiatt nem bonyilítjuk tovább az algoritmust)
és eggyel növeli a maxhosszat is, ami viszont szükséges, hogy ha a végén van
a maximális jó sorozat, akkor is helyes eredményt adjon az algoritmus.
"""

print(f'A leghosszabb prímsorozat kezdetének indexe: {maxkezdet}, a kezdőszám: {X[maxkezdet]}.')
print(f'A sorozat hossza: {maxhossz}, a sorozat: {X[maxkezdet:(maxkezdet+maxhossz)]}')



cimiro("A feladat megoldása Python-függvényekkel","=")

"""
Ha egy sztring kétféle karaktert tartalmaz, pl. "0" és "1" vagy "+" és "-",
akkor könnyen meg tudjuk állapítani, hogy az egyik karakterből milyen hosszú,
és hol kezdődik a leghosszabb részsorozat.
Csak annyit kell tennünk, hogy a másik karakterrel feldaraboljuk a sztringet,
s a kapott listának megkeressük a leghosszabb elemét.
Mivel a max() függvény sztringeket is össze tud hasonlítani az ASCII-kód alapján,
azonos karakterek esetén a leghosszabb lesz a legnagyobb értékű. Pl. "+++">"++".
Ha megvan a leghosszabb karaktersorozat, az index() függvénnyel azt is meg tudjuk mondani,
hogy hol kezdődik az eredeti sztringben.

Az általános esetet pedig könnyen visszavezethetjük erre a speciális helyzetre.
A map() függvénnyel készítsük el azt az igazságlistát,
amely az eredeti lista T tulajdonságú elemeinél az "i", a többinél a "h" karaktert tartalmazza.
Ebből a "".join(igazságlista) pedig elkészíti a megfelelő sztringet.
"""

def igazsag(elem):
    if T(elem): # a korábban tetszőlegesen definiált T tulajdonságra
        return"i"
    else:
        return "h"

igazsaglista=list(map(lambda elem: igazsag(elem) ,X))
igazsagstr="".join(igazsaglista)
darstr=igazsagstr.split("h")
maxstr=max(darstr)
maxhossz=len(maxstr)
maxkezdet=igazsagstr.index(maxstr)

print(f'A leghosszabb prímsorozat kezdetének indexe: {maxkezdet}, a kezdőszám: {X[maxkezdet]}.')
print(f'A sorozat hossza: {maxhossz}, a sorozat: {X[maxkezdet:(maxkezdet+maxhossz)]}')


input("\nA befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak08\BevGyak08.py ==========================

A leghosszabb adott tulajdonságú részsorozat keresése:
******************************************************

a leghosszabb prímsorozat egy véletlen sorozatban
*************************************************

500 db 1 és 100 közötti véletlen egész szám előállítása
-------------------------------------------------------

 69      56      78      90      86      23*     68       1       4     100
 43*     37*     58      70      77      22      49      44      79*     5
 72      66      70      30      99      21      15       8      42      23*
 96      64      56      43*     92      92      37*     74      26      55
 63      70       5      47*     17*     22      19*     65      44     100
 95      91      18      97*     11*     84      17*     97*      6       7
 73*     75      46       8      72      56      82      62      15      39
 29*     29*     99      84      39      72      86      10      35      92
 62      23*     37*     91      11*     48      37*     80      87      69
 91       3      51      62      23*     81      49     100      25      57

 14      68      26      19*     36      61*    100      37*     52      64
  8      65      19*     79*     80      88       3      63      93      65
 75      53*     80      96      66      70      36      64      11*     27
 14      23*     41*     93      86      52      71*     31*     88       3
 62      41*     26      25      35      38      65      96      16      75
 41*     36      78      34       9      39      91      52       6      82
 65      17*     29*     42      21      32      21       5      49      41*
 17*     32      60      16      98      77      91      42      44      31*
 17*     46      32      92      69      83*     25      54      20      72
 26      82      37*     56      39      95      52      39      30      21

 24      99      58      78      93      37*     86      18      76      67*
 46      67*     12       8      34      10      52      72      20      24
 83*     71*     12       5      98       6      93      60       2      11*
 33      67*     80      95      38      51      81      31*     57       5
 72      17*      6      73*      7      55      33      49      47*     66
 38      83*     67*     24      50      45       1      83*     12      55
 81      16      38      21       8      37*     35      79*      8      40
 72      46      10      84      46      24      40      58      71*      9
  5      67*     94      28      52      90      67*     21      30       1
 98      92      98      94      11*     10      82      68      24      20

 52      81       2      49      18       4       1      19*     99      94
 82      54      77      65      34      44      34      80     100      33
 95      33      23*     35      33      55      11*     78      84       6
 37*     89*     79*     70      85      82      54      73*     34      45
 47*     24      97*     36      69      53*     54      18      38      11*
 42      88      60       3      86      91      80      36      59*     46
 57      27      92       7      30      28      88      54      31*     78
  5       4      62      94      53*      6      82      97*     14      73*
 25      48      39      31*     77      55      69      93      30      85
 63      96      51      61*      3       1      26      87      80       6

 19*     90      24      64      84      79*     41*     90      70      47*
 41*     66      66      51      87      80      26      97*     88      91
 13*     33      53*     62      87      11*     63      48      76      32
 49      91      68      67*     99      58      30      47*     82      69
 42       8      32      59*     84      89*     62       4      69      40
 30      60      93      71*     26       8      81      15      79*     60
 74       8     100      37*     11*     72      75      79*     63      83*
 85      74      30       8      51      17*     64      16      12      55
 52      71*     77      13*     93      64      91       1      70      24
 60      83*     58      15      29*     59*     56      90      46      79*
```

```
A leghosszabb prímsorozat megkeresése és kiíratása
==================================================

A leghosszabb prímsorozat kezdetének indexe: 330, a kezdőszám: 37.
A sorozat hossza: 3, a sorozat: [37, 89, 79]


A feladat megoldása Python-függvényekkel
========================================

A leghosszabb prímsorozat kezdetének indexe: 330, a kezdőszám: 37.
A sorozat hossza: 3, a sorozat: [37, 89, 79]

A befejezéshez nyomd meg az ENTER billentyűt!
```

9. Ismétlődés nélküli véletlen sorozat előállítása: lottóhúzás

```
"""
9. Ismétlődés nélküli véletlen sorozat előállítása: lottóhúzás
@author Klemand66
"""

from random import *
from datetime import *


def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")



cimiro("Ismétlődés nélküli véletlen sorozat előállítása: lottóhúzás ","*")

cimiro("1. Ötöslottó","=")

def veletlenegesz(alsohatar,felsohatar):
    szam=randrange(alsohatar,felsohatar+1)
    return szam


cimiro("Figyelmeztetés: 14 év alattiak nem lottózhatnak!","-")

szul=input("Kérem a születési dátumodat 1982.8.12 alakban: ")

print(f'A szul valtozó típusa az első értékadás után: {type(szul)}')
szul=szul.strip().split(".")
print(f'A szul valtozó típusa a második értékadás után: {type(szul)}')
ev,ho,nap=int(szul[0]),int(szul[1]),int(szul[2])
szul=date(ev,ho,nap)
print(f'A szul valtozó típusa a harmadik értékadás után: {type(szul)}')

ma=date.today()
napok=(ma-szul).days
evek=napok//365

print(f'\nTehát {evek} éves vagy.')
if evek<14:
    print(f'Találkozzunk {14*365-napok} nap múlva!')
elif evek>80:
    print("Ilyen korban már kerülni kell a túlzott izgalmakat!")
else:
    print("Sok szerencsét a játékhoz!\n")

    cimiro("Tippelés","-")

    tippek=[]
    while len(tippek)<5:
        tipp=int(input(f'Kérem a(z) {len(tippek)+1}. tippet: '))
        if tipp not in tippek:
            tippek.append(tipp)

    tippek.sort()

    print(f'\nA tippjeid nagyság szerint rendezve: ',end="")
    for n in tippek:
        print(n," ",end="")
```

```
    cimiro("Számhúzás","-")


    kihuzottak=[]
    while len(kihuzottak)<5:
        kihuzott=veletlenegesz(1,90)
        if kihuzott not in kihuzottak:
            kihuzottak.append(kihuzott)

    kihuzottak.sort()

    print(f'A kihúzott számok nagyság szerint rendezve: ',end="")
    for n in kihuzottak:
        print(n," ",end="")

    print()



    cimiro("Értékelés","-")

    nyertesek=[]

    for n in tippek:
        if n in kihuzottak:
            nyertesek.append(n)

    print(f'A találataid száma: {len(nyertesek)}, a nyertes számaid: ',end="")
    for n in nyertesek:
        print(n," ",end="")

    print()

    if len(nyertesek)>3:
        print("Tudsz kölcsönadni? :-)")



cimiro("2. A lottóhúzás általánosítása","=")

cimiro("k db a és f közötti különböző véletlen egész szám előállítása","=");

a=int(input("Kérem az alsó határ értékét: "))
f=int(input("Kérem a felső határ értékét: "))
k=int(input("Kérem a darabszámot: "))

if k>f-a+1:
      print(f'\nSajnos a(z) [{a},{f}] intervallumba csak {f-a+1} egész szám fér bele. :-(')
else:
    veletlenek=[]
    while len(veletlenek)<k:
        veletlen=veletlenegesz(a,f)
        if veletlen not in veletlenek:
            veletlenek.append(veletlen)

    veletlenek.sort()

    leghosszabb=max(abs(min(veletlenek)),abs(max(veletlenek)))
    d=len(str(leghosszabb))+3


    print("\nAz előállított számok nagyság szerint rendezve:\n")
    for i,n in enumerate(veletlenek):
        print(f'{n:{d}}' ,end="")
        if (i+1)%10==0:
            print()
    print()

print("\n")
input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:/Python/KlInfoPy/BevGyak09/BevGyak09.py ==========================


Ismétlődés nélküli véletlen sorozat előállítása: lottóhúzás
************************************************************



1. Ötöslottó
============



Figyelmeztetés: 14 év alattiak nem lottózhatnak!
------------------------------------------------

Kérem a születési dátumodat 1982.8.12 alakban: 2012.12.21
A szul valtozó típusa az első értékadás után: <class 'str'>
A szul valtozó típusa a második értékadás után: <class 'list'>
A szul valtozó típusa a harmadik értékadás után: <class 'datetime.date'>

Tehát 9 éves vagy.
Találkozzunk 1537 nap múlva!


2. A lottóhúzás általánosítása
==============================



k db a és f közötti különböző véletlen egész szám előállítása
=============================================================

Kérem az alsó határ értékét: -4
Kérem a felső határ értékét: 6
Kérem a darabszámot: 20

Sajnos a(z) [-4,6] intervallumba csak 11 egész szám fér bele. :-(


A befejezéshez nyomd meg az ENTER billentyűt!
```

```
========================== RESTART: C:/Python/KlInfoPy/BevGyak09/BevGyak09.py ==========================


Ismétlődés nélküli véletlen sorozat előállítása: lottóhúzás
************************************************************


1. Ötöslottó
============

Figyelmeztetés: 14 év alattiak nem lottózhatnak!
------------------------------------------------

Kérem a születési dátumodat 1982.8.12 alakban: 2004.4.22
A szul valtozó típusa az első értékadás után: <class 'str'>
A szul valtozó típusa a második értékadás után: <class 'list'>
A szul valtozó típusa a harmadik értékadás után: <class 'datetime.date'>

Tehát 18 éves vagy.
Sok szerencsét a játékhoz!


Tippelés
--------

Kérem a(z) 1. tippet: 21
Kérem a(z) 2. tippet: 13
Kérem a(z) 3. tippet: 21
Kérem a(z) 3. tippet: 28
Kérem a(z) 4. tippet: 28
Kérem a(z) 4. tippet: 66
Kérem a(z) 5. tippet: 90

A tippjeid nagyság szerint rendezve: 13  21  28  66  90

Számhúzás
---------

A kihúzott számok nagyság szerint rendezve: 18  32  47  79  81


Értékelés
---------

A találataid száma: 0, a nyertes számaid:


2. A lottóhúzás általánosítása
==============================

k db a és f közötti különböző véletlen egész szám előállítása
=============================================================

Kérem az alsó határ értékét: -100
Kérem a felső határ értékét: 100
Kérem a darabszámot: 30

Az előállított számok nagyság szerint rendezve:

  -89  -75  -65  -39  -36  -31  -27  -23  -22  -20
  -18  -16   -7    6   10   16   19   20   22   23
   26   33   35   36   47   60   62   75   78   96

A befejezéshez nyomd meg az ENTER billentyűt!
```

10. Statisztikai minta gyakorisági táblázata: érték szerinti indexelés és általános módszer

```
"""
10. Statisztikai minta gyakorisági táblázata:
érték szerinti indexelés és általános módszer
@author Klemand66
"""

def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

def cimiro2(cim,karakter):
    print(cim)
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")



cimiro("Statisztikai minta gyakorisági táblázata:","*")
cimiro2("érték szerinti indexelés és általános módszer","*")



cimiro("1. Kvantitatív minta gyakorisági táblázata érték szerinti indexeléssel","=")

print("A januar.txt fájl sorai a januári napi átlaghőmérsékleteket tartalmazzák.")
print("Az értékekről tudjuk, hogy -10 és 10 Celsius fok közötti egész értékek.")
print("Készítsük el a minta gyakorisági táblázatát!\n")

napdb=31 #tudom, hogy a jabuár 31 napos

januar=[]

betxt = open("januar.txt")

for i in range(napdb):
    sor=betxt.readline()
    januar.append(int(sor))

betxt.close()

"""
Arra vagyunk kíváncsiak, hogy a -10 és 10 közötti értékek hányszor fordulnak elő a listában.

Az érték szerinti indexelést úgy képzelhetjük el, mint amikor a dolgozatoknál minden bemondott
jegy után húzunk egy vonalat a megfelelő osztályzathoz, csak itt nem vonalakat húzunk, és nem
a végén számoljuk össze őket, hanem rögtön növeljük a számhoz tartozó gyakoriság értéket.

Először tehát felírjuk a lehetséges értékeket és mindegyikhez 0-t írunk.
Ez a kezdeti gyakoriságértékük.
Azután végignézzük a konkrét példánkban a januári hőmérséklet értékeket,
és ha pl. látunk egy 8-ast, a 8-as szám gyakorisági értékét megnöveljük 1-gyel.

Ha az értékek nem 0-tól indulnak, át szoktuk transzformálni az [m,n] intervallumot
a [0,n-m] intervallumra. Ez negatív értékeknél is működik.
Bár a Pythonban van negatív indexelés, de -1 a lista végét jelenti,
és ez úgysem segítene júliusban, amikor 20 ás 40 fok között vannak az értékek.
"""

# A gyakorisági táblázat előkészítése, nullázása:

gyaktabla=[]
m=-10 # ha nem tudnánk a min. és max. értékét, lekérdezhetnénk a min() és a max() függvényekkel
n=10
for i in range(m,n+1):
    gyaktabla.append(0)
```

```
# A gyakorisági táblázat feltöltése az érték szerinti indexeléssel:

for homerseklet in januar: # végigmegyünk a januári hőmérsékleteken
    gyaktabla[homerseklet-m]+=1 # Pl. -8 fok esetén gyaktabla[2] értékét növeljük

print("A januári hőmérsékletek gyakorisági táblázata:\n");

for i,gyak in enumerate(gyaktabla):
    if gyak>0: # amelyik hőmérséklet nem fordult elő, azt ki sem íratjuk
        print(f'{(i+m):3} Celsius fokos hőmérséklet {gyak:2} napon volt.\n')



cimiro("2. Általános módszer tetszőleges minta esetén","=")

print("Az osztaly.txt fájl sorai szóközökkel elválasztva egy osztály tanulóinak")
print("vezetéknevét, keresztnevét és lakhelyét tartalmazza.")
print("Készítsük el a lakhelyek gyakorisági táblázatát!\n")

lakhelyek=[] # ebben a listában egy település annyiszor szerepel, ahány tanuló ott lakik

betxt = open("osztaly.txt")

"""
Megnéztem, hogy bár a fájl ékezetes, de ANSI kódolású, ezért nem kellett az encoding="utf-8".

Nem ismerjük az osztálylétszámot, a sorok azonos szerkezetűek,
így a legegyszerűbben for ciklussal járhatjuk be a fájl sorait.
Vegyük észre, hogy a nevekre nem is lesz szükségünk.
"""

for sor in betxt:
    darsor=sor.strip().split()
    lakhelyek.append(darsor[2])

betxt.close()

telepulesek=[] # ide gyűjtjük a településeket, mindegyiket csak egyszer
telgyak=[]     # ide pedig azt, hogy hányszor fordulnak elő

"""
Felvetődhet a kérdés: miért nem kétdimenziós listában tároljuk
ezeket a szorosan összetartozó adatokat.
Azért, mert így tudjuk könnyebben megnézni,
hogy a soron következő település szerepelt-e már a listában.
Rendezni nem fogunk, a kapcsot pedig az index biztosítja közöttük.
"""

for telepules in lakhelyek:  # végignézzük a lakhelyek lista településeit
    if telepules in telepulesek:  # ha a vizsgált település már szerepel a telepulesek listában,
        telindex=telepulesek.index(telepules)  # akkor megkeressük az indexét,
        telgyak[telindex]+=1 # és az ehhez az indexhez tartozó településgyakoriságot megnöveljük
    else:  # ha viszont a vizsgált település nem szerepel még a telepulesek listában,
        telepulesek.append(telepules) # akkor hozzáfűzzük
        telgyak.append(1)  # és ezzel együtt beállítjuk a hozzá tartozó gyakoriságot is 1-re

print("A települések gyakorisági táblázata:\n");

# Mekkora szélesség kell a leghosszabb nevű településnek?
szelessegek=list(map(lambda t : len(t), telepulesek))
d=max(szelessegek)

for i,tel in enumerate(telepulesek):
    print(f'{tel:{d}} az osztály {telgyak[i]:2} tanulójának a lakhelye.\n')


print("\n")
input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:/Python/KlInfoPy/BevGyak10/BevGyak10.py ==========================


Statisztikai minta gyakorisági táblázata:
*****************************************

érték szerinti indexelés és általános módszer
**********************************************



1. Kvantitatív minta gyakorisági táblázata érték szerinti indexeléssel
======================================================================

A januar.txt fájl sorai a januári napi átlaghőmérsékleteket tartalmazzák.
Az értékekről tudjuk, hogy -10 és 10 Celsius fok közötti egész értékek.
Készítsük el a minta gyakorisági táblázatát!

A januári hőmérsékletek gyakorisági táblázata:

 -4 Celsius fokos hőmérséklet  2 napon volt.

 -3 Celsius fokos hőmérséklet  3 napon volt.

 -2 Celsius fokos hőmérséklet  5 napon volt.

  0 Celsius fokos hőmérséklet  4 napon volt.

  1 Celsius fokos hőmérséklet  8 napon volt.

  2 Celsius fokos hőmérséklet  4 napon volt.

  3 Celsius fokos hőmérséklet  4 napon volt.

  4 Celsius fokos hőmérséklet  1 napon volt.



2. Általános módszer tetszőleges minta esetén
=============================================

Az osztaly.txt fájl sorai szóközökkel elválasztva egy osztály tanulóinak
vezetéknevét, keresztnevét és lakhelyét tartalmazza.
Készítsük el a lakhelyek gyakorisági táblázatát!

A települések gyakorisági táblázata:

Körmend         az osztály 11 tanulójának a lakhelye.

Ják             az osztály  4 tanulójának a lakhelye.

Csákánydoroszló az osztály  3 tanulójának a lakhelye.

Egyházasrádóc   az osztály  2 tanulójának a lakhelye.

Nádasd          az osztály  3 tanulójának a lakhelye.

Magyarszecsőd   az osztály  1 tanulójának a lakhelye.

Szentgotthárd   az osztály  1 tanulójának a lakhelye.



A befejezéshez nyomd meg az ENTER billentyűt!
```

11. Számelméleti feladatok: euklidészi algoritmus, osztók száma és összege, prímek, prímfelbontás

```
"""
11. Számelméleti feladatok:
euklidészi algoritmus, osztók száma és összege, prímek, prímfelbontás
@author Klemand66
"""

from math import *
from datetime import *

def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

def cimiro2(cim,karakter):
    print(cim)
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")



cimiro("Számelméleti feladatok:","*")
cimiro2("euklidészi algoritmus, osztók száma és összege, prímek, prímfelbontás","*")



cimiro("1. Euklidészi algoritmus","=")

print("Két megadott pozitív egész szám")
print("legnagyobb közös osztójának")
print("és legkisebb közös többszörösének meghatározása")
print("az euklidészi algoritmus segítségével\n")


a=int(input("Kérek egy pozitív egész számot: "))
b=int(input("Kérek egy másik pozitív egész számot: "))

n,k=max(a,b),min(a,b) #többszörös értékadás

"""
Az euklidészi algoritmus lényege:
A nagyobb számot (osztandó) maradékosan elosztjuk a kisebbel (osztó).
Ezután az osztandó az előbbi osztó lesz, az osztó pedig a kapott maradék.
Az eljárást addig folytatjuk, míg a maradék 0 nem lesz.
Az utolsó el nem tűnő maradék lesz a legnagyobb közös osztó.
"""

while True:
    r=n%k
    if r==0:
        break # kiugráskor az előző maradék k-ban van
    n,k=k,r

lnko=k
lkkt=a*b//lnko
"""
a és b értékét szerencsére meghagytuk.
Sima osztásnál at eredmény float lenne, viszont tudjuk, hogy az eredmény biztosan egész lesz.
"""

print(f'\n{a:,} és {b:,} legnagyobb közös osztója: {lnko:,},\n\
a legkisebb közös többszörösük pedig {lkkt:,}.\n')
```

```
cimiro("2. Egy megadott intervallumba eső számok osztóinak elemzése,","=")
cimiro2("a legösszetettebb és az n-szeresen tökéletes számok meghatározása","=")

print("n-szeresen tökéletes szám: osztóinak összege egyenlő a szám n-szeresével.\n")


a=int(input("Kérem az intervallum alsó határát, egy 1-nél nagyobb pozitív egész számot: "))
f=int(input("Kérem az intervallum felső határát, egy az alsó határnál nem kisebb pozitív egész számot: "))
print()

szel=int(1.5*len(str(f)))

osztolistak=[]
osztokszamai=[]
osztokosszegei=[]
ntokeletesek=[]

for n in range(a,f+1):
    osztok=[]
    osztokszama=0
    osztokosszege=0
    v=int(sqrt(n))
    for i in range(1,v+1):
        if n % i == 0:
            osztokszama+=1
            osztok.append(i)
            osztokosszege+=i
            if n//i != i: # négyzetszám esetén a gyököt csak egyszer vesszük
                osztokszama+=1
                osztok.append(n//i)
                osztokosszege+=n//i
    osztok.sort()
    osztolistak.append(osztok)
    osztokszamai.append(osztokszama)
    osztokosszegei.append(osztokosszege)
    ntokeletesek.append(osztokosszege%n==0)

print(f'Az [{a},{f}] intervallumba eső számoknak, az osztóik számának, osztóik összegének és az osztóik listájának')
print("kiíratása a fenti sorrendben a monitorra és/vagy a osztok.txt fájlba\n")


if f-a<=100: # a monitorra csak akkor íratjuk ki az eredménylistát, ha nem túl hosszú
    for n in range(a,f+1):
        print(f'{n:{szel}}: {osztokszamai[n-a]:4}, {osztokosszegei[n-a]:{szel}},  {osztolistak[n-a]}')
        # transzformálás: a 0 index tartozik az a alsó határhoz
else:
    print(f'\nA megadott [{a},{f}] intervallum túl nagy, a teljes lista az osztok.txt fájlban található.\n')

kitxt = open("osztok.txt","w")

kitxt.write(f'A(z) [{a},{f}] intervallumba eső számok, az osztóik száma, az osztóik összege és az osztóik listája\n')

for n in range(a,f+1):
    kitxt.write(f'{n:{szel}}: {osztokszamai[n-a]:4}, {osztokosszegei[n-a]:{szel}},  {osztolistak[n-a]}\n')

kitxt.close()
print("\nA kiíratás befejeződött.\n")
```

```
cimiro("A megadott intervallum legösszetettebb számai","-")

maxosztoszam=max(osztokszamai)

print(f'Az osztók számának maximuma a(z) [{a},{f}] intervallumban: {maxosztoszam}.')
print("Ennyi osztóval rendelkeznek a következő számok: ", end="")

for i,elem in enumerate(osztokszamai):
    if elem==maxosztoszam:
        print(f'{i+a} ',end="")
print("\n")



cimiro("n-szeresen tökéletes számok a megadott intervallumban","-")


print(f'\nA(z) [{a},{f}] intervallum {ntokeletesek.count(True)} db n-szeresen tökéletes számot tartalmaz: ')

for n,elem in enumerate(ntokeletesek):
    if elem:
        print(f'{n+a:,}: {osztokosszegei[n]//(n+a)}x tökéletes')
print("\n")



cimiro("3. Prímszámok","=")

print("Prímszámnak nevezünk egy természetes számot, ha pontosan két osztója van, az 1 és önmaga.\n")



cimiro("Prímszámvizsgálat","-")

print("Egy megadott pozitív egész számról eldöntjük, hogy prímszám-e.")

def prim_e(n):      # prímvizsgáló függvény
    if n==1:
        return False
    elif n==2:
        return True
    elif n%2==0:
        return False
    else:
        for i in range(3,int(sqrt(n))+1,2): # csak a páratlan számokat vizsgáljuk n gyökéig
            if n%i==0:
                return False # ha talált egy osztót, akkor nem prím
        return True # ha végigment a cikluson és nem talált osztót, akkor prím


a=int(input("Kérek egy pozitív egész számot: "))

if prim_e(a):
   print(f'\nA megadott {a:,} prímszám.\n')
else:
   print(f'\nA megadott {a:,} nem prímszám.\n')



cimiro("Egy megadott intervallum prímszámainak meghatározása","-")

print("A megadott határok közötti prímszámok")
print("kiíratása a monitorra és/vagy a primszamok.txt fájlba\n")

"""
Megtehetnénk, hogy az intervallum minden elemét megvizsgáljuk a prim_e függvénnyel.
Ha viszont sok nagy számról kell eldöntenünk, hogy prímek-e, akkor először érdemes készíteni egy
(páratlan) prímvektort a felső határ gyökét meghaladva. Azzal is gyorsítjuk a folyamatot, hogy
a prímvektor felépítéséhez a prímvizsgálatoknál magát az éppen készülő prímvektort használjuk fel.

A megadott intervallumnak (az esetleg előforduló 2-n kívül) elég a páratlan elemeit vizsgálni.
A prímvizsgálat algoritmusában pedig az összes páratlan szám helyett csak a primvektor elemeit
kell vizsgálni az oszthatóság szempontjából. Ha pedig eljutunk a prímvektorban a vizsgált szám
gyökénél nagyobb prímig úgy, hogy egyik prím sem volt osztója, akkor az biztosan prím lesz.
"""
```

```
a=int(input("Kérem az intervallum alsó határát, egy pozitív egész számot: "))
f=int(input("Kérem az intervallum felső határát, egy az alsó határnál nem kisebb egész számot: "))
print("\n")


ah=a
primek=[]                # a megadott intervallum prímszámait a primek vektorba tesszük

if ah<3:
    if f>1:
        primek.append(2)
    ah=3

if ah%2==0:
    ah=ah+1

szel=int(1.5*len(str(f)))

print(f'A prímvektorkészítés kezdésének időpontja:    {datetime.now()}')

primvektor=[3]            # csak a páratlan prímeket fogja tartalmazni

def prim_e2(n):           # módosított prímvizsgáló függvény
    gyokn=int(sqrt(n))
    for i in primvektor:  # a prímvizsgálathoz magát a készülő prímvektort használjuk fel
        if i>gyokn:
            return True
        if n%i==0:
            return False

n=3                       # a prímvektor elkészítése a prímvizsgálathoz
gyokf=int(sqrt(f))
while True:
    if primvektor[-1]>gyokf:  # az utolsó prím legyen nagyobb a felső határ gyökénél
        break
    n=n+2
    if prim_e2(n):
        primvektor.append(n)

print(f'A prímvektorkészítés befejezésének időpontja: {datetime.now()}')

print(f'A prímvektor {len(primvektor)} darab prímszámot tartalmaz, az utolsó: {primvektor[-1]}.\n')

print(f'A megadott intervallum prímszámai meghatározásának kezdési időpontja:    {datetime.now()}')

for n in range(ah,f+1,2):  # a megadott intervallumnak már csak a páratlan elemeit vizsgáljuk
    if prim_e2(n):
        primek.append(n)

print(f'A megadott intervallum prímszámai meghatározásának befejezési időpontja: {datetime.now()}')

print(f'Összesen {len(primek)} darab prímszám van a(z) [{a:,};{f:,}] intervallumban.\n')

odb=100//szel

if f-a<=1000:
    for i,p in enumerate(primek):
        print(f'{p:{szel},} ', end="")
        if (i+1)%odb==0:
            print()
    print()

v=input("Szeretné kiíratni a teljes listát a primszamok.txt fájlba? i/n: ")
if v=="i":
    print(f'A megadott intervallum prímszámai kiíratásának kezdési időpontja:    {datetime.now()}')
    kitxt = open("primszamok.txt","w")

    kitxt.write(f'Összesen {len(primek)} darab prímszám van a(z) [{a:,};{f:,}] intervallumban:\n\n')
    for i,p in enumerate(primek):
        kitxt.write(f'{p:{szel},} ')
        if (i+1)%odb==0:
            kitxt.write("\n")

    kitxt.close()
    print(f'A megadott intervallum prímszámai kiíratásának befejezési időpontja: {datetime.now()}\n')
```

```
cimiro("A megadott intervallum prímsűrűsödéseinek meghatározása","-")

print("A megadott határok közötti prímsűrűsödések prímszámainak")
print("kiíratása a monitorra és/vagy az primsurusodesek.txt fájlba.\n")
print("Értelmezzük a prímsűrűsödést: mekkora intervallumban min. hány prím legyen.\n")

sh=int(input("Kérem az intervallum hosszát: "))
db=int(input("Kérem a prímek minimális darabszámát az intervallumban: "))

primsurusodesek=[]

for i in range(len(primek)-db+1):
    if primek[i+db-1]-primek[i]<sh:
        primsurusodes=[]
        for j in range(i,i+db):
            primsurusodes.append(primek[j])
        primsurusodesek.append(primsurusodes)

print(f'\nÖsszesen {len(primsurusodesek)} darab prím{db}-s van a(z) [{a:,};{f:,}] intervallumban.\n')

if len(primsurusodesek)<=100:
    for lista in primsurusodesek:
        for p in lista:
            print(f'{p:{szel},}',end=" ")
        print()

v=input("\nSzeretné kiíratni a teljes listát a primsurusodesek.txt fájlba? i/n: ")
if v=="i":
    kitxt = open("primsurusodesek.txt","w")

    kitxt.write(f'Összesen {len(primsurusodesek)} darab prím{db}-s van a(z) [{a:,};{f:,}]
intervallumban:\n\n')
    for lista in primsurusodesek:
        for p in lista:
            kitxt.write(f'{p:{szel},} ')
        kitxt.write("\n")

    kitxt.close()
    print("\nA kiíratás befejeződött.\n")



cimiro("A megadott intervallum prímhiányainak meghatározása","-")

print("A megadott határok között található legalább {hh} egység hosszúságú részintervallumok")
print("kiíratása a monitorra és/vagy a primhianyok.txt fájlba, melyekben egyetlen prímszám sincs. ")
print("A leghosszabb prímhiányos részintervallum meghatározása.\n")
print("Értelmezzük a prímhiányt: mekkora intervallumban ne legyen egyetlen prím sem.\n")

hh=int(input("Kérem az intervallum hosszát: "))

primhianyok=[]
hmax=0

for i in range(len(primek)-1):
    if primek[i+1]-primek[i]>hh:
            primhianyok.append([primek[i]+1,primek[i+1]-1])
    if primek[i+1]-primek[i]-1>hmax:
        hmax=primek[i+1]-primek[i]-1

print(f'\n\nÖsszesen {len(primhianyok)} darab prímhiány van a(z) [{a:,};{f:,}] intervallumban.')
print(f'\nA leghosszabb prímhiányos részintervallum hossza: {hmax}.\n')

if len(primhianyok)<=100:
    for par in primhianyok:
        print(f'[{par[0]:,};{par[1]:,}]  hossza  {par[1]-par[0]+1:3}.')

v=input("\nSzeretné kiíratni a teljes listát a primsurusodesek.txt fájlba? i/n: ")
if v=="i":
    kitxt = open("primhianyok.txt","w")

    kitxt.write(f'Összesen {len(primhianyok)} darab prímhiány van a(z) [{a:,};{f:,}] intervallumban:\n\n')
    for par in primhianyok:
        kitxt.write(f'[{par[0]:,};{par[1]:,}]  hossza  {par[1]-par[0]+1:3}\n')
    kitxt.write(f'\nA leghosszabb prímhiányos részintervallum hossza: {hmax}.\n')

    kitxt.close()
    print("\nA kiíratás befejeződött.\n")
```

```
cimiro("4. Egy megadott intervallumba eső számok prímtényezős felbontása","_")

print("A megadott határok közötti pozitív egész számok")
print("prímtényezős felbontásának kiíratása a monitorra vagy a primfelbontas.txt fájlba\n")


a=int(input("Kérem az intervallum alsó határát, egy 1-nél nagyobb pozitív egész számot: "))
f=int(input("Kérem az intervallum felső határát, egy az alsó határnál nem kisebb pozitív egész számot: "))
print()

szel=int(1.5*len(str(f)))

primfelbontasok=[]

"""
Lényegében az eratoszthenészi szita alapgondolatát alkalmazzuk a prímfelbontásra.
A 2 prím, ha a szám osztható 2-vel, akkor prímosztója.
Addig osztogatjuk 2-vel, amíg páratlan számot nem kapunk.
Közben azt is tudtuk számolni, hogy 2-nek hányadik hatványával osztható.
Ha már páratlan az aktuális számunk, megnézzük a következő számmal, a 3-mal való oszthatóságát.
Ha osztható vele, akkor a 3 biztosan prímosztó is egyben.
Most is addig osztogatunk 3-mmal, amíg lehet.
Ha már nem osztója, következik a 4. De ez biztos nem lesz osztója,
hiszen a kettő hatványaival már végigosztottuk.
Jön az 5. Ha osztó, prímosztó is egyben.
Addig osztogatunk 5-tel is, amíg lehet.
Azután jön a 6. Biztosan nem lesz osztó, hiszen 2-vel és 3-mal már végigosztogattunk.
Tehát látható, hogy ha egy következő szám osztó lesz, az biztosan prímosztó is egyben.
"""

for n in range(a,f+1):
    akt=n
    i=2
    primfelbontas=[]
    while akt>1:
        if akt%i==0:
            primfelbontas.append(i)
            akt=akt//i
        else:
            i+=1
    primfelbontasok.append(primfelbontas)

print(f'\nA(z) [{a},{f}] intervallumba eső egész számok prímtényezős felbontása\n')

if f-a<=100:
    for i,felbontas in enumerate(primfelbontasok):
        print(f'{i+a:{szel},} = ',end="")
        alap=None
        for j,t in enumerate(felbontas):
            if t==alap:
                continue
            else: # azaz ha egy új alapot találunk
                alap=t # elsőként a 2 jöhet számításba
                kitevo=felbontas.count(alap) # csak megszámoltatjuk, hányszor volt osztható az alappal
                if kitevo==1:
                    print(f'{alap}',end="")
                else:
                    print(f'{alap}^{kitevo}',end="")
                if j+kitevo<len(felbontas):
                    print("*",end="")
        print("")
else:
    print(f'\nA megadott [{a},{f}] intervallum túl nagy, a teljes lista a primhatvanyok.txt fájlban található.\n')
```

```
kitxt = open("primhatvanyok.txt","w")

kitxt.write(f'A(z) [{a},{f}] intervallumba eső egész számok prímtényezős felbontása\n')

for i,felbontas in enumerate(primfelbontasok):
    kitxt.write(f'{i+a:{szel},} = ')
    alap=None
    for j,t in enumerate(felbontas):
        if t==alap:
            continue
        else:
            alap=t
            kitevo=felbontas.count(alap)
            if kitevo==1:
                kitxt.write(f'{alap}')
            else:
                kitxt.write(f'{alap}^{kitevo}')
            if j+kitevo<len(felbontas):
                kitxt.write("*")
    kitxt.write("\n")

kitxt.close()
print("\nA kiíratás befejeződött.\n")


print("\n")
input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak11\BevGyak11.py ==========================


Számelméleti feladatok:
***********************

euklidészi algoritmus, osztók száma és összege, prímek, prímfelbontás
**********************************************************************



1. Euklidészi algoritmus
========================

Két megadott pozitív egész szám
legnagyobb közös osztójának
és legkisebb közös többszörösének meghatározása
az euklidészi algoritmus segítségével

Kérek egy pozitív egész számot: 1956
Kérek egy másik pozitív egész számot: 2028

1,956 és 2,028 legnagyobb közös osztója: 12,
a legkisebb közös többszörösük pedig 330,564.



2. Egy megadott intervallumba eső számok osztóinak elemzése,
============================================================

a legösszetettebb és az n-szeresen tökéletes számok meghatározása
==================================================================

n-szeresen tökéletes szám: osztóinak összege egyenlő a szám n-szeresével.

Kérem az intervallum alsó határát, egy 1-nél nagyobb pozitív egész számot: 460
Kérem az intervallum felső határát, egy az alsó határnál nem kisebb pozitív egész számot: 500

Az [460,500] intervallumba eső számoknak, az osztóik számának, osztóik összegének és az osztóik listájának
kiíratása a fenti sorrendben a monitorra és/vagy a osztok.txt fájlba

 460:   12, 1008,  [1, 2, 4, 5, 10, 20, 23, 46, 92, 115, 230, 460]
 461:    2,  462,  [1, 461]
 462:   16, 1152,  [1, 2, 3, 6, 7, 11, 14, 21, 22, 33, 42, 66, 77, 154, 231, 462]
 463:    2,  464,  [1, 463]
 464:   10,  930,  [1, 2, 4, 8, 16, 29, 58, 116, 232, 464]
 465:    8,  768,  [1, 3, 5, 15, 31, 93, 155, 465]
 466:    4,  702,  [1, 2, 233, 466]
 467:    2,  468,  [1, 467]
 468:   18, 1274,  [1, 2, 3, 4, 6, 9, 12, 13, 18, 26, 36, 39, 52, 78, 117, 156, 234, 468]
 469:    4,  544,  [1, 7, 67, 469]
 470:    8,  864,  [1, 2, 5, 10, 47, 94, 235, 470]
 471:    4,  632,  [1, 3, 157, 471]
 472:    8,  900,  [1, 2, 4, 8, 59, 118, 236, 472]
 473:    4,  528,  [1, 11, 43, 473]
 474:    8,  960,  [1, 2, 3, 6, 79, 158, 237, 474]
 475:    6,  620,  [1, 5, 19, 25, 95, 475]
 476:   12, 1008,  [1, 2, 4, 7, 14, 17, 28, 34, 68, 119, 238, 476]
 477:    6,  702,  [1, 3, 9, 53, 159, 477]
 478:    4,  720,  [1, 2, 239, 478]
 479:    2,  480,  [1, 479]
 480:   24, 1512,  [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 32, 40, 48, 60, 80, 96, 120, 160, 240, 480]
```

```
 481:    4,  532,  [1, 13, 37, 481]
 482:    4,  726,  [1, 2, 241, 482]
 483:    8,  768,  [1, 3, 7, 21, 23, 69, 161, 483]
 484:    9,  931,  [1, 2, 4, 11, 22, 44, 121, 242, 484]
 485:    4,  588,  [1, 5, 97, 485]
 486:   12, 1092,  [1, 2, 3, 6, 9, 18, 27, 54, 81, 162, 243, 486]
 487:    2,  488,  [1, 487]
 488:    8,  930,  [1, 2, 4, 8, 61, 122, 244, 488]
 489:    4,  656,  [1, 3, 163, 489]
 490:   12, 1026,  [1, 2, 5, 7, 10, 14, 35, 49, 70, 98, 245, 490]
 491:    2,  492,  [1, 491]
 492:   12, 1176,  [1, 2, 3, 4, 6, 12, 41, 82, 123, 164, 246, 492]
 493:    4,  540,  [1, 17, 29, 493]
 494:    8,  840,  [1, 2, 13, 19, 26, 38, 247, 494]
 495:   12,  936,  [1, 3, 5, 9, 11, 15, 33, 45, 55, 99, 165, 495]
 496:   10,  992,  [1, 2, 4, 8, 16, 31, 62, 124, 248, 496]
 497:    4,  576,  [1, 7, 71, 497]
 498:    8, 1008,  [1, 2, 3, 6, 83, 166, 249, 498]
 499:    2,  500,  [1, 499]
 500:   12, 1092,  [1, 2, 4, 5, 10, 20, 25, 50, 100, 125, 250, 500]

A kiíratás befejeződött.



A megadott intervallum legösszetettebb számai
----------------------------------------------

Az osztók számának maximuma a(z) [460,500] intervallumban: 24.
Ennyi osztóval rendelkeznek a következő számok: 480



n-szeresen tökéletes számok a megadott intervallumban
------------------------------------------------------


A(z) [460,500] intervallum 1 db n-szeresen tökéletes számot tartalmaz:
496: 2x tökéletes




3. Prímszámok
=============

Prímszámnak nevezünk egy természetes számot, ha pontosan két osztója van, az 1 és önmaga.



Prímszámvizsgálat
-----------------

Egy megadott pozitív egész számról eldöntjük, hogy prímszám-e.
Kérek egy pozitív egész számot: 2027

A megadott 2,027 prímszám.
```

```
Egy megadott intervallum prímszámainak meghatározása
----------------------------------------------------

A megadott határok közötti prímszámok
kiíratása a monitorra és/vagy a primszamok.txt fájlba

Kérem az intervallum alsó határát, egy pozitív egész számot: 99999000000
Kérem az intervallum felső határát, egy az alsó határnál nem kisebb egész számot: 100000000000


A prímvektorkészítés kezdésének időpontja:    2023-05-14 22:06:02.913623
A prímvektorkészítés befejezésének időpontja: 2023-05-14 22:06:03.167580
A prímvektor 27293 darab prímszámot tartalmaz, az utolsó: 316241.

A megadott intervallum prímszámai meghatározásának kezdési időpontja:    2023-05-14 22:06:03.183206
A megadott intervallum prímszámai meghatározásának befejezési időpontja: 2023-05-14 22:08:20.193532
Összesen 39331 darab prímszám van a(z) [99,999,000,000;100,000,000,000] intervallumban.

Szeretné kiíratni a teljes listát a primszamok.txt fájlba? i/n: i
A megadott intervallum prímszámai kiíratásának kezdési időpontja:    2023-05-14 22:08:23.620152
A megadott intervallum prímszámai kiíratásának befejezési időpontja: 2023-05-14 22:08:23.682640



A megadott intervallum prímsűrűsödéseinek meghatározása
--------------------------------------------------------

A megadott határok közötti prímsűrűsödések prímszámainak
kiíratása a monitorra és/vagy az primsurusodesek.txt fájlba.

Értelmezzük a prímsűrűsödést: mekkora intervallumban min. hány prím legyen.

Kérem az intervallum hosszát: 30
Kérem a prímek minimális darabszámát az intervallumban: 6

Összesen 18 darab prím6-s van a(z) [99,999,000,000;100,000,000,000] intervallumban.

    99,999,100,699     99,999,100,703     99,999,100,709     99,999,100,711     99,999,100,717     99,999,100,723
    99,999,145,093     99,999,145,097     99,999,145,109     99,999,145,111     99,999,145,117     99,999,145,121
    99,999,145,097     99,999,145,109     99,999,145,111     99,999,145,117     99,999,145,121     99,999,145,123
    99,999,230,159     99,999,230,167     99,999,230,171     99,999,230,173     99,999,230,179     99,999,230,183
    99,999,347,671     99,999,347,681     99,999,347,683     99,999,347,687     99,999,347,689     99,999,347,699
    99,999,385,253     99,999,385,259     99,999,385,261     99,999,385,267     99,999,385,271     99,999,385,273
    99,999,438,967     99,999,438,971     99,999,438,977     99,999,438,983     99,999,438,989     99,999,438,991
    99,999,475,789     99,999,475,799     99,999,475,801     99,999,475,811     99,999,475,813     99,999,475,817
    99,999,607,151     99,999,607,157     99,999,607,159     99,999,607,163     99,999,607,171     99,999,607,177
    99,999,712,507     99,999,712,511     99,999,712,517     99,999,712,523     99,999,712,529     99,999,712,531
    99,999,741,911     99,999,741,913     99,999,741,917     99,999,741,923     99,999,741,929     99,999,741,931
    99,999,745,243     99,999,745,247     99,999,745,249     99,999,745,259     99,999,745,267     99,999,745,271
    99,999,833,281     99,999,833,287     99,999,833,291     99,999,833,293     99,999,833,303     99,999,833,309
    99,999,929,657     99,999,929,669     99,999,929,671     99,999,929,677     99,999,929,681     99,999,929,683
    99,999,951,197     99,999,951,199     99,999,951,203     99,999,951,209     99,999,951,211     99,999,951,217
    99,999,958,039     99,999,958,043     99,999,958,057     99,999,958,061     99,999,958,063     99,999,958,067
    99,999,962,653     99,999,962,659     99,999,962,663     99,999,962,669     99,999,962,677     99,999,962,681
    99,999,962,659     99,999,962,663     99,999,962,669     99,999,962,677     99,999,962,681     99,999,962,683

Szeretné kiíratni a teljes listát a primsurusodesek.txt fájlba? i/n: i

A kiíratás befejeződött.
```

```
A megadott intervallum prímhiányainak meghatározása
---------------------------------------------------

A megadott határok között található legalább {hh} egység hosszúságú részintervallumok
kiíratása a monitorra és/vagy a primhianyok.txt fájlba, melyekben egyetlen prímszám sincs.
A leghosszabb prímhiányos részintervallum meghatározása.

Értelmezzük a prímhiányt: mekkora intervallumban ne legyen egyetlen prím sem.

Kérem az intervallum hosszát: 150


Összesen 43 darab prímhiány van a(z) [99,999,000,000;100,000,000,000] intervallumban.

A leghosszabb prímhiányos részintervallum hossza: 245.

[99,999,040,334;99,999,040,488]  hossza  155.
[99,999,075,032;99,999,075,192]  hossza  161.
[99,999,076,268;99,999,076,422]  hossza  155.
[99,999,123,912;99,999,124,080]  hossza  169.
[99,999,140,882;99,999,141,036]  hossza  155.
[99,999,168,740;99,999,168,910]  hossza  171.
[99,999,180,168;99,999,180,330]  hossza  163.
[99,999,182,364;99,999,182,518]  hossza  155.
[99,999,184,824;99,999,184,992]  hossza  169.
[99,999,212,738;99,999,212,920]  hossza  183.
[99,999,224,202;99,999,224,358]  hossza  157.
[99,999,243,798;99,999,244,042]  hossza  245.
[99,999,252,638;99,999,252,790]  hossza  153.
[99,999,253,658;99,999,253,816]  hossza  159.
[99,999,264,752;99,999,264,922]  hossza  171.
[99,999,321,762;99,999,321,936]  hossza  175.
[99,999,379,724;99,999,379,900]  hossza  177.
[99,999,408,920;99,999,409,116]  hossza  197.
[99,999,418,848;99,999,419,016]  hossza  169.
[99,999,452,178;99,999,452,338]  hossza  161.
[99,999,479,978;99,999,480,130]  hossza  153.
[99,999,493,148;99,999,493,342]  hossza  195.
[99,999,511,688;99,999,511,866]  hossza  179.
[99,999,517,068;99,999,517,222]  hossza  155.
[99,999,542,934;99,999,543,100]  hossza  167.
[99,999,554,330;99,999,554,488]  hossza  159.
[99,999,567,932;99,999,568,086]  hossza  155.
[99,999,595,880;99,999,596,070]  hossza  191.
[99,999,596,972;99,999,597,148]  hossza  177.
[99,999,616,968;99,999,617,128]  hossza  161.
[99,999,624,368;99,999,624,550]  hossza  183.
[99,999,704,192;99,999,704,352]  hossza  161.
[99,999,708,332;99,999,708,502]  hossza  171.
[99,999,709,764;99,999,709,918]  hossza  155.
[99,999,770,672;99,999,770,850]  hossza  179.
[99,999,792,948;99,999,793,176]  hossza  229.
[99,999,812,268;99,999,812,448]  hossza  181.
[99,999,888,720;99,999,888,928]  hossza  209.
[99,999,890,912;99,999,891,066]  hossza  155.
[99,999,900,824;99,999,900,982]  hossza  159.
[99,999,933,558;99,999,933,712]  hossza  155.
[99,999,955,062;99,999,955,260]  hossza  199.
[99,999,959,048;99,999,959,200]  hossza  153.

Szeretné kiíratni a teljes listát a primhianyok.txt fájlba? i/n: i

A kiíratás befejeződött.
```

```
4. Egy megadott intervallumba eső számok prímtényezős felbontása
_____________________________________________________________

A megadott határok közötti pozitív egész számok
prímtényezős felbontásának kiíratása a monitorra vagy a primfelbontas.txt fájlba

Kérem az intervallum alsó határát, egy 1-nél nagyobb pozitív egész számot: 2000
Kérem az intervallum felső határát, egy az alsó határnál nem kisebb pozitív egész számot: 2080


A(z) [2000,2080] intervallumba eső egész számok prímtényezős felbontása

 2,000 = 2^4*5^3                                    2,041 = 13*157
 2,001 = 3*23*29                                    2,042 = 2*1021
 2,002 = 2*7*11*13                                  2,043 = 3^2*227
 2,003 = 2003                                       2,044 = 2^2*7*73
 2,004 = 2^2*3*167                                  2,045 = 5*409
 2,005 = 5*401                                      2,046 = 2*3*11*31
 2,006 = 2*17*59                                    2,047 = 23*89
 2,007 = 3^2*223                                    2,048 = 2^11
 2,008 = 2^3*251                                    2,049 = 3*683
 2,009 = 7^2*41                                     2,050 = 2*5^2*41
 2,010 = 2*3*5*67                                   2,051 = 7*293
 2,011 = 2011                                       2,052 = 2^2*3^3*19
 2,012 = 2^2*503                                    2,053 = 2053
 2,013 = 3*11*61                                    2,054 = 2*13*79
 2,014 = 2*19*53                                    2,055 = 3*5*137
 2,015 = 5*13*31                                    2,056 = 2^3*257
 2,016 = 2^5*3^2*7                                  2,057 = 11^2*17
 2,017 = 2017                                       2,058 = 2*3*7^3
 2,018 = 2*1009                                     2,059 = 29*71
 2,019 = 3*673                                      2,060 = 2^2*5*103
 2,020 = 2^2*5*101                                  2,061 = 3^2*229
 2,021 = 43*47                                      2,062 = 2*1031
 2,022 = 2*3*337                                    2,063 = 2063
 2,023 = 7*17^2                                     2,064 = 2^4*3*43
 2,024 = 2^3*11*23                                  2,065 = 5*7*59
 2,025 = 3^4*5^2                                    2,066 = 2*1033
 2,026 = 2*1013                                     2,067 = 3*13*53
 2,027 = 2027                                       2,068 = 2^2*11*47
 2,028 = 2^2*3*13^2                                 2,069 = 2069
 2,029 = 2029                                       2,070 = 2*3^2*5*23
 2,030 = 2*5*7*29                                   2,071 = 19*109
 2,031 = 3*677                                      2,072 = 2^3*7*37
 2,032 = 2^4*127                                    2,073 = 3*691
 2,033 = 19*107                                     2,074 = 2*17*61
 2,034 = 2*3^2*113                                  2,075 = 5^2*83
 2,035 = 5*11*37                                    2,076 = 2^2*3*173
 2,036 = 2^2*509                                    2,077 = 31*67
 2,037 = 3*7*97                                     2,078 = 2*1039
 2,038 = 2*1019                                     2,079 = 3^3*7*11
 2,039 = 2039                                       2,080 = 2^5*5*13
 2,040 = 2^3*3*5*17

A kiíratás befejeződött.


A befejezéshez nyomd meg az ENTER billentyűt!
```

Az osztok.txt szövegfájl:

```
Az [460,500] intervallumba eső számok, az osztóik száma, az osztóik összege és az osztóik listája
 460: 12, 1008,  [1, 2, 4, 5, 10, 20, 23, 46, 92, 115, 230, 460]
 461:  2, 462,  [1, 461]
 462: 16, 1152,  [1, 2, 3, 6, 7, 11, 14, 21, 22, 33, 42, 66, 77, 154, 231, 462]
 463:  2, 464,  [1, 463]
 464: 10, 930,  [1, 2, 4, 8, 16, 29, 58, 116, 232, 464]
 465:  8, 768,  [1, 3, 5, 15, 31, 93, 155, 465]
 466:  4, 702,  [1, 2, 233, 466]
 467:  2, 468,  [1, 467]
 468: 18, 1274,  [1, 2, 3, 4, 6, 9, 12, 13, 18, 26, 36, 39, 52, 78, 117, 156, 234, 468]
 469:  4, 544,  [1, 7, 67, 469]
 470:  8, 864,  [1, 2, 5, 10, 47, 94, 235, 470]
 471:  4, 632,  [1, 3, 157, 471]
 472:  8, 900,  [1, 2, 4, 8, 59, 118, 236, 472]
 473:  4, 528,  [1, 11, 43, 473]
 474:  8, 960,  [1, 2, 3, 6, 79, 158, 237, 474]
 475:  6, 620,  [1, 5, 19, 25, 95, 475]
 476: 12, 1008,  [1, 2, 4, 7, 14, 17, 28, 34, 68, 119, 238, 476]
 477:  6, 702,  [1, 3, 9, 53, 159, 477]
 478:  4, 720,  [1, 2, 239, 478]
 479:  2, 480,  [1, 479]
 480: 24, 1512,  [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 32, 40, 48, 60, 80, 96, 120, 160, 240, 480]
 481:  4, 532,  [1, 13, 37, 481]
 482:  4, 726,  [1, 2, 241, 482]
 483:  8, 768,  [1, 3, 7, 21, 23, 69, 161, 483]
 484:  9, 931,  [1, 2, 4, 11, 22, 44, 121, 242, 484]
 485:  4, 588,  [1, 5, 97, 485]
 486: 12, 1092,  [1, 2, 3, 6, 9, 18, 27, 54, 81, 162, 243, 486]
 487:  2, 488,  [1, 487]
 488:  8, 930,  [1, 2, 4, 8, 61, 122, 244, 488]
 489:  4, 656,  [1, 3, 163, 489]
 490: 12, 1026,  [1, 2, 5, 7, 10, 14, 35, 49, 70, 98, 245, 490]
 491:  2, 492,  [1, 491]
 492: 12, 1176,  [1, 2, 3, 4, 6, 12, 41, 82, 123, 164, 246, 492]
 493:  4, 540,  [1, 17, 29, 493]
 494:  8, 840,  [1, 2, 13, 19, 26, 38, 247, 494]
 495: 12, 936,  [1, 3, 5, 9, 11, 15, 33, 45, 55, 99, 165, 495]
 496: 10, 992,  [1, 2, 4, 8, 16, 31, 62, 124, 248, 496]
 497:  4, 576,  [1, 7, 71, 497]
 498:  8, 1008,  [1, 2, 3, 6, 83, 166, 249, 498]
 499:  2, 500,  [1, 499]
 500: 12, 1092,  [1, 2, 4, 5, 10, 20, 25, 50, 100, 125, 250, 500]
```

A primszamok.txt szövegfájl:

Összesen 39331 darab prímszám van a(z) [99,999,000,000;100,000,000,000] intervallumban:

99,999,000,001	99,999,000,037	99,999,000,089	99,999,000,181	99,999,000,229
99,999,000,239	99,999,000,247	99,999,000,259	99,999,000,319	99,999,000,337
99,999,000,389	99,999,000,469	99,999,000,503	99,999,000,589	99,999,000,707
99,999,000,709	99,999,000,769	99,999,000,793	99,999,000,799	99,999,000,811
99,999,000,869	99,999,000,883	99,999,000,901	99,999,000,917	99,999,000,929
99,999,000,953	99,999,000,979	99,999,001,007	99,999,001,051	99,999,001,073
99,999,001,079	99,999,001,097	99,999,001,103	99,999,001,121	99,999,001,133
99,999,001,147	99,999,001,177	99,999,001,217	99,999,001,237	99,999,001,259
99,999,001,261	99,999,001,301	99,999,001,303	99,999,001,331	99,999,001,339
99,999,001,343	99,999,001,357	99,999,001,429	99,999,001,447	99,999,001,481
99,999,001,483	99,999,001,549	99,999,001,559	99,999,001,561	99,999,001,577
99,999,001,589	99,999,001,591	99,999,001,643	99,999,001,649	99,999,001,679
99,999,001,691	99,999,001,693	99,999,001,703	99,999,001,727	99,999,001,787
99,999,001,793	99,999,001,819	99,999,001,903	99,999,001,969	99,999,001,987
99,999,001,997	99,999,002,003	99,999,002,053	99,999,002,093	99,999,002,111

…

A primsurusodesek.txt szövegfájl:

Összesen 18 darab prím6-s van a(z) [99,999,000,000;100,000,000,000] intervallumban:

99,999,100,699	99,999,100,703	99,999,100,709	99,999,100,711	99,999,100,717	99,999,100,723
99,999,145,093	99,999,145,097	99,999,145,109	99,999,145,111	99,999,145,117	99,999,145,121
99,999,145,097	99,999,145,109	99,999,145,111	99,999,145,117	99,999,145,121	99,999,145,123
99,999,230,159	99,999,230,167	99,999,230,171	99,999,230,173	99,999,230,179	99,999,230,183
99,999,347,671	99,999,347,681	99,999,347,683	99,999,347,687	99,999,347,689	99,999,347,699
99,999,385,253	99,999,385,259	99,999,385,261	99,999,385,267	99,999,385,271	99,999,385,273
99,999,438,967	99,999,438,971	99,999,438,977	99,999,438,983	99,999,438,989	99,999,438,991
99,999,475,789	99,999,475,799	99,999,475,801	99,999,475,811	99,999,475,813	99,999,475,817
99,999,607,151	99,999,607,157	99,999,607,159	99,999,607,163	99,999,607,171	99,999,607,177
99,999,712,507	99,999,712,511	99,999,712,517	99,999,712,523	99,999,712,529	99,999,712,531
99,999,741,911	99,999,741,913	99,999,741,917	99,999,741,923	99,999,741,929	99,999,741,931
99,999,745,243	99,999,745,247	99,999,745,249	99,999,745,259	99,999,745,267	99,999,745,271
99,999,833,281	99,999,833,287	99,999,833,291	99,999,833,293	99,999,833,303	99,999,833,309
99,999,929,657	99,999,929,669	99,999,929,671	99,999,929,677	99,999,929,681	99,999,929,683
99,999,951,197	99,999,951,199	99,999,951,203	99,999,951,209	99,999,951,211	99,999,951,217
99,999,958,039	99,999,958,043	99,999,958,057	99,999,958,061	99,999,958,063	99,999,958,067
99,999,962,653	99,999,962,659	99,999,962,663	99,999,962,669	99,999,962,677	99,999,962,681
99,999,962,659	99,999,962,663	99,999,962,669	99,999,962,677	99,999,962,681	99,999,962,683

A primhianyok.txt szövegfájl:

Összesen 43 darab prímhiány van a(z) [99,999,000,000;100,000,000,000] intervallumban:

[99,999,040,334;99,999,040,488]	hossza	155
[99,999,075,032;99,999,075,192]	hossza	161
[99,999,076,268;99,999,076,422]	hossza	155
[99,999,123,912;99,999,124,080]	hossza	169
[99,999,140,882;99,999,141,036]	hossza	155
[99,999,168,740;99,999,168,910]	hossza	171
[99,999,180,168;99,999,180,330]	hossza	163
[99,999,182,364;99,999,182,518]	hossza	155
[99,999,184,824;99,999,184,992]	hossza	169
[99,999,212,738;99,999,212,920]	hossza	183
[99,999,224,202;99,999,224,358]	hossza	157
[99,999,243,798;99,999,244,042]	hossza	245
[99,999,252,638;99,999,252,790]	hossza	153
[99,999,253,658;99,999,253,816]	hossza	159
[99,999,264,752;99,999,264,922]	hossza	171
[99,999,321,762;99,999,321,936]	hossza	175
[99,999,379,724;99,999,379,900]	hossza	177
[99,999,408,920;99,999,409,116]	hossza	197
[99,999,418,848;99,999,419,016]	hossza	169
[99,999,452,178;99,999,452,338]	hossza	161
[99,999,479,978;99,999,480,130]	hossza	153
[99,999,493,148;99,999,493,342]	hossza	195
[99,999,511,688;99,999,511,866]	hossza	179
[99,999,517,068;99,999,517,222]	hossza	155
[99,999,542,934;99,999,543,100]	hossza	167
[99,999,554,330;99,999,554,488]	hossza	159
[99,999,567,932;99,999,568,086]	hossza	155
[99,999,595,880;99,999,596,070]	hossza	191
[99,999,596,972;99,999,597,148]	hossza	177
[99,999,616,968;99,999,617,128]	hossza	161
[99,999,624,368;99,999,624,550]	hossza	183
[99,999,704,192;99,999,704,352]	hossza	161
[99,999,708,332;99,999,708,502]	hossza	171
[99,999,709,764;99,999,709,918]	hossza	155

```
[99,999,770,672;99,999,770,850]  hossza  179
[99,999,792,948;99,999,793,176]  hossza  229
[99,999,812,268;99,999,812,448]  hossza  181
[99,999,888,720;99,999,888,928]  hossza  209
[99,999,890,912;99,999,891,066]  hossza  155
[99,999,900,824;99,999,900,982]  hossza  159
[99,999,933,558;99,999,933,712]  hossza  155
[99,999,955,062;99,999,955,260]  hossza  199
[99,999,959,048;99,999,959,200]  hossza  153

A leghosszabb prímhiányos részintervallum hossza: 245.


A primhatvanyok.txt szövegfájl:

A(z) [2000,2080] intervallumba eső egész számok prímtényezős felbontása

 2,000 = 2^4*5^3                                    2,041 = 13*157
 2,001 = 3*23*29                                    2,042 = 2*1021
 2,002 = 2*7*11*13                                  2,043 = 3^2*227
 2,003 = 2003                                       2,044 = 2^2*7*73
 2,004 = 2^2*3*167                                  2,045 = 5*409
 2,005 = 5*401                                      2,046 = 2*3*11*31
 2,006 = 2*17*59                                    2,047 = 23*89
 2,007 = 3^2*223                                    2,048 = 2^11
 2,008 = 2^3*251                                    2,049 = 3*683
 2,009 = 7^2*41                                     2,050 = 2*5^2*41
 2,010 = 2*3*5*67                                   2,051 = 7*293
 2,011 = 2011                                       2,052 = 2^2*3^3*19
 2,012 = 2^2*503                                    2,053 = 2053
 2,013 = 3*11*61                                    2,054 = 2*13*79
 2,014 = 2*19*53                                    2,055 = 3*5*137
 2,015 = 5*13*31                                    2,056 = 2^3*257
 2,016 = 2^5*3^2*7                                  2,057 = 11^2*17
 2,017 = 2017                                       2,058 = 2*3*7^3
 2,018 = 2*1009                                     2,059 = 29*71
 2,019 = 3*673                                      2,060 = 2^2*5*103
 2,020 = 2^2*5*101                                  2,061 = 3^2*229
 2,021 = 43*47                                      2,062 = 2*1031
 2,022 = 2*3*337                                    2,063 = 2063
 2,023 = 7*17^2                                     2,064 = 2^4*3*43
 2,024 = 2^3*11*23                                  2,065 = 5*7*59
 2,025 = 3^4*5^2                                    2,066 = 2*1033
 2,026 = 2*1013                                     2,067 = 3*13*53
 2,027 = 2027                                       2,068 = 2^2*11*47
 2,028 = 2^2*3*13^2                                 2,069 = 2069
 2,029 = 2029                                       2,070 = 2*3^2*5*23
 2,030 = 2*5*7*29                                   2,071 = 19*109
 2,031 = 3*677                                      2,072 = 2^3*7*37
 2,032 = 2^4*127                                    2,073 = 3*691
 2,033 = 19*107                                     2,074 = 2*17*61
 2,034 = 2*3^2*113                                  2,075 = 5^2*83
 2,035 = 5*11*37                                    2,076 = 2^2*3*173
 2,036 = 2^2*509                                    2,077 = 31*67
 2,037 = 3*7*97                                     2,078 = 2*1039
 2,038 = 2*1019                                     2,079 = 3^3*7*11
 2,039 = 2039                                       2,080 = 2^5*5*13
 2,040 = 2^3*3*5*17
```

12. Rekurzió: Fibonacci-sorozat, gyorsrendezés, zárójelezés érvényessége

```
"""
12. Rekurzió: Fibonacci-sorozat, gyorsrendezés, zárójelezés érvényessége
@author Klemand66
"""

from math import *

def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")


cimiro("Rekurzió: Fibonacci-sorozat, gxorsrendezés, zárójelezés érvényessége","*")


cimiro("1. A Fibonacci-sorozat tagjainak meghatározása ciklussal","=")

n=int(input("Kérem a sorozat tagjának indexét (1000-re is kapásból válaszol): "))

Fibsor=[1,1]

for i in range(2,n):
    Fibsor.append(Fibsor[i-2]+Fibsor[i-1])

print(f'\nA Fibonacci-sorozat {n}. tagja: {Fibsor[n-1]:,}\n')


print(f'A Fibonacci-sorozat első {n} tagjának kiíratása a fibonacci.txt fájlba\n')

kitxt = open("fibonacci.txt","w")

for i,elem in enumerate(Fibsor):
    kitxt.write(f'{i+1}. tag: {elem:,}\n')

kitxt.close()

print("A Fájl kiíratása befejeződött.")



cimiro("2. A Fibonacci-sorozat tagjainak meghatározása rekurzióval","=")


def Fib(n):
    if n==1 or n==2:
        return 1
    else:
        return Fib(n-2) + Fib(n-1)


n=int(input("Kérem a sorozat tagjának indexét (35-nél már gondolkodik): "))


print(f'\nA Fibonacci-sorozat {n}. tagja: {Fib(n):,}')
```

```
cimiro("3. Gyorsrendezés (quicksort)","=")

"""
A gyorsrendezés függvény az "oszd meg és uralkodj" elven működik:
a rendezendő számok listáját egy szétválasztó szám,
pl. a legkisebb és a legnagyobb elem számtani közepe segítségével
két részre bontja, a nála kisebb-egyenlőkre és nagyobbakra,
majd ezeket a részeket rekurzívan, önmaga meghívásával rendezi.
"""

def quicksort(lista):
    if len(lista)<=1:
        print(lista)
        return lista
    else:
        szetvalaszto=(min(lista)+max(lista))/2
        K,N=[],[]
        for elem in lista:
            if elem<=szetvalaszto:
                K.append(elem)
            else:
                N.append(elem)
        print(f'   {K}   << {szetvalaszto:.1f} >>   {N}')
        return quicksort(K) + quicksort(N)
        # A + művelet a listák összefűzését végzi

X = [84, 48, -64, 56, 76, 38, -16, 44, 68, 28, 74, 18, -52, 0, -96, 8]

print(f'Az eredeti lista:  {X}\n')
"""
20 számmal kezdtem, de kiakadt tőle, még a 18-tól is.
Úgy látszik, igazából csak dísznek van a Pythonban a rekurzió.
"""
print(f'\nA rendezett lista: {quicksort(X)}')



cimiro("4. Egy megadott zárójelezés érvényességének eldöntése","=")

def ervenyes_e(z):
    if z=="":
        return True
    else:
        par=""
        if "()" in z:
            par="()"
        elif "[]" in z:
            par="[]"
        elif "{}" in z:
            par="{}"

        if par!="":
            i=z.index(par)
            return ervenyes_e(z[:i]+z[i+2:]) # kihagyjuk a zárójelpárokat és visszaküldjük


print("Helyes pl. {{[]()[]}}[()][] vagy {[([]{((()))})]}")
print("Hibás pl. {[)(]}, {([])}, {{[()[]}} vagy {{{()}}}}\n")

zj=input("Kérek egy zárójelsorozatot: ")

if ervenyes_e(zj):
    print("A megadott zárójelezés érvényes.")
else:
    print("A megadott zárójelezés nem érvényes.")


input("\nA befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak12\BevGyak12.py ==========================


Rekurzió: Fibonacci-sorozat, gxorsrendezés, zárójelezés érvényessége
*********************************************************************


1. A Fibonacci-sorozat tagjainak meghatározása ciklussal
========================================================

Kérem a sorozat tagjának indexét (1000-re is kapásból válaszol): 40

A Fibonacci-sorozat 40. tagja: 102,334,155

A Fibonacci-sorozat első 40 tagjának kiíratása a fibonacci.txt fájlba

A Fájl kiíratása befejeződött.


2. A Fibonacci-sorozat tagjainak meghatározása rekurzióval
==========================================================

Kérem a sorozat tagjának indexét (35-nél már gondolkodik): 40

A Fibonacci-sorozat 40. tagja: 102,334,155


3. Gyorsrendezés (quicksort)
============================

Az eredeti lista:  [84, 48, -64, 56, 76, 38, -16, 44, 68, 28, 74, 18, -52, 0, -96, 8]

   [-64, -16, -52, -96]   << -6.0 >>   [84, 48, 56, 76, 38, 44, 68, 28, 74, 18, 0, 8]
   [-64, -96]   << -56.0 >>   [-16, -52]
   [-96]   << -80.0 >>   [-64]
[-96]
[-64]
   [-52]   << -34.0 >>   [-16]
[-52]
[-16]
   [38, 28, 18, 0, 8]   << 42.0 >>   [84, 48, 56, 76, 44, 68, 74]
   [18, 0, 8]   << 19.0 >>   [38, 28]
   [0, 8]   << 9.0 >>   [18]
   [0]   << 4.0 >>   [8]
[0]
[8]
[18]
   [28]   << 33.0 >>   [38]
[28]
[38]
   [48, 56, 44]   << 64.0 >>   [84, 76, 68, 74]
   [48, 44]   << 50.0 >>   [56]
   [44]   << 46.0 >>   [48]
[44]
[48]
[56]
   [76, 68, 74]   << 76.0 >>   [84]
   [68]   << 72.0 >>   [76, 74]
[68]
   [74]   << 75.0 >>   [76]
[74]
[76]
[84]

A rendezett lista: [-96, -64, -52, -16, 0, 8, 18, 28, 38, 44, 48, 56, 68, 74, 76, 84]
```

```
4. Egy megadott zárójelezés érvényességének eldöntése
=====================================================

Helyes pl. {{[]()[]}}[()][] vagy {[([]{((())})])]}
Hibás pl. {[)(]}, {([])}, {{[()[]}} vagy {{{()}}}}

Kérek egy zárójelsorozatot: {[{{{[(())]}}]}}[[[()]]]
A megadott zárójelezés érvényes.

A befejezéshez nyomd meg az ENTER billentyűt!



A fibonacci.txt szöveges fájl:

1. tag: 1
2. tag: 1
3. tag: 2
4. tag: 3
5. tag: 5
6. tag: 8
7. tag: 13
8. tag: 21
9. tag: 34
10. tag: 55
11. tag: 89
12. tag: 144
13. tag: 233
14. tag: 377
15. tag: 610
16. tag: 987
17. tag: 1,597
18. tag: 2,584
19. tag: 4,181
20. tag: 6,765
21. tag: 10,946
22. tag: 17,711
23. tag: 28,657
24. tag: 46,368
25. tag: 75,025
26. tag: 121,393
27. tag: 196,418
28. tag: 317,811
29. tag: 514,229
30. tag: 832,040
31. tag: 1,346,269
32. tag: 2,178,309
33. tag: 3,524,578
34. tag: 5,702,887
35. tag: 9,227,465
36. tag: 14,930,352
37. tag: 24,157,817
38. tag: 39,088,169
39. tag: 63,245,986
40. tag: 102,334,155
```

13. Verselemzés: Üllői-úti fák

```
"""
13. Verselemzés: Üllői-úti fák
@author Klemand66
"""

from random import *

def cimiro(cim,karakter):
    print("\n\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")


cimiro("Verselemzés: Üllői-úti fák","*")


cimiro("1. feladat","=")

print("A vers költőjének, címének és szavainak beolvasása a vers.txt fájlból")
print("A szerző és a cím legyenenek nagybetűsek, a szavak kisbetűsek!\n")


betxt = open("vers.txt")
# Megnéztem, hogy bár a fájl ékezetes, de ANSI kódolású, ezért nem kellett az encoding="utf-8".

# Tudom, hogy az 1. sor tartalmazza a költő nevét, a 2. a címet.

kolto= betxt.readline().strip().upper()
cim= betxt.readline().strip().upper()

"""
Azt viszont nem tudom, hogy a vers hány soros és hol vannak közben a "fehér" sorok,
de a for ciklus a fehér sorok átugrásával képes beolvasni a fájlt.

Jól látszik a különbség az üres és a fehér sor között: az első sor lényegében annak felel meg,
hogy while (sor:=betxt.readline()) != "":, a fehér sor viszont csak a megtisztítás után lesz üres.
"""

szavak=[]

for sor in betxt:
    sor=sor.strip()
    if sor == "":
        continue
    darsor=sor.split()
    for elem in darsor:
        szavak.append(elem.strip(" ,.!?").lower())

"""
A strip(" ,.!?") az elemek elejéről, ill végéről távolítja el az összes megadott karaktert,
ameddig más karaktert nem talál. Ha beljebb is lennének még ilyen karakterek, azokat már nem.
"""

betxt.close()

print(f'A beolvasás megtörtént. A vers {len(szavak)} szóból áll.\n')
```

```
cimiro("2. feladat","=")

print("Írjuk ki a képernyőre a vers szerzőjét és címét,")
print("a betűket ciklikusan hárommal eltolva (a szóközöket, kötőjeleket meghagyva)!\n")

def eltol(k):
    abc = "AÁBCDEÉFGHIÍJKLMNOÓÖŐPQRSTUÚÜŰVWXYZAÁB"
    if k in abc:
        return abc[abc.index(k)+3]
    else:
        return k # a szóköz, kötőjel marad

print("A költő: ",end="")

for k in kolto:
    print(eltol(k),end="")
print("\n")

print("A cím: ",end="")

for k in cim:
    print(eltol(k),end="")
print("\n")



cimiro("3. feladat","=")

print("A verselemzes.txt fájlban listázzuk ki a versben szereplő szavakat,")
print("mindegyiket csak egyszer! Adjuk meg a szavak előfordulásainak számát,")
print("a karaktereik számát és az ékezetes betűik számát!\n")

def ekdb(szo):
    db=0
    for k in szo:
        if k in "áéíóöőúüű":
            db+=1
    return db

"""
Most külön készítjük el a szólistát, a megfelelő gyakoriságlistát
és az ékezetes betűk számának listáját, mert látni fogjuk,
hogy így mennyivel egyszerűbb dolgozni velük.
Viszont rendezés előtt készítünk majd belőlük egyetlen kétdimenziós listát,
mert az viszont úgy lesz sokkal egyszerűbb.
"""

szolista=[]      # a vers szavainak ismétlődés nélküli listája
szogyaklista=[]  # a szólista szavainak előfordulási száma a versben
ekdblista=[]     # a szólista szavai ékezetes karaktereinek száma

for szo in szavak:
    if szo in szolista:
        szoindex=szolista.index(szo)
        szogyaklista[szoindex]+=1
    else:
        szolista.append(szo)
        szogyaklista.append(1)
        ekdblista.append(ekdb(szo))


kitxt = open("verselemzes.txt","w")

kitxt.write("A versben szereplő szavak, gyakoriságuk, karaktereik, ill. ékezetes betűik száma\n")
kitxt.write("az első előfordulásuk sorrendjében:\n\n")

szohosszlista=list(map(lambda x: len(x),szolista))
szel=max(szohosszlista)+1

for i,szo in enumerate(szolista):
    kitxt.write(f'{i+1:2}. {szo:{szel}}: {szogyaklista[i]}x, {len(szo):2}, {ekdblista[i]}\n')

kitxt.close()

print("A kiíratás befejeződött.\n")
```

```
cimiro("4. feladat","=")

print("Adjuk meg a leghosszabb szót, ill. szavakat,")
print("és készítsük el a szavak hossza szerinti statisztikát!\n")

print("A leghosszabb ",end="")

leghosszabbak=list(filter(lambda x:len(x)==max(szohosszlista),szolista))

if len(leghosszabbak)==1:
    print("szó ", end="")
else:
        print("szavak ", end="")

maxhossz=len(leghosszabbak[0])
print(f'({maxhossz} karakter): ', end="")

for elem in leghosszabbak:
    print(elem, end=" ")

print("\n\nA szavak hossza szerinti statisztika:\n")

for i in range(1,maxhossz+1):
    print(f'{i:2} karakter: {szohosszlista.count(i):2} szó')

print("\n")



cimiro("5. feladat","=")

print("Határozzuk meg a szólistában az ékezetes szavak százalékos arányát")
print("két tizedes pontossággal!\n")

db=len(ekdblista)
ekdb=db-ekdblista.count(0)

print(f'Az ékezetes szavak aránya a szólistában: {100*ekdb/db:.2f}%\n')



cimiro("6. feladat","=")

print("Írjuk ki a képernyőre a szólista szavainak egy véletlen permutációját!")
print("Minden sor 1 és 5 közötti véletlen számú szót tartalmazzon!")
print("Négy sor után legyen egy üres sor is!\n")

permszolista=[]

while len(permszolista)<len(szolista):
    velindex=randrange(len(szolista))
    if szolista[velindex] not in permszolista:
        permszolista.append(szolista[velindex])

print("A szólista szavainak véletlen permutációja:\n")


szodb=randrange(1,6)
sordb=0
for szo in permszolista:
    print(szo,end=" ")
    szodb-=1
    if szodb==0:
        szodb=randrange(1,6)
        print()
        sordb+=1
        if sordb==4:
            sordb=0
            print()

print("\n")
```

```
cimiro("7. feladat","=")

print("Írjuk ki a képernyőre a vers szavainak egy véletlen permutációját,")
print("azaz a szólista szavainak egy olyan ismétléses permutációját,")
print("melyben minden szó annyiszor szerepel, mint a versben!")
print("Minden sor 1 és 5 közötti véletlen számú szót tartalmazzon!")
print("Négy sor után legyen egy üres sor is!\n")

permindex=[]
permvers=[]

while len(permvers)<len(szavak):
    velindex=randrange(len(szavak))
    if velindex not in permindex:
        permindex.append(velindex)
        permvers.append(szavak[velindex])

print("A vers szavainak véletlen permutációja:\n")

szodb=randrange(1,6)
sordb=0
for szo in permvers:
    print(szo,end=" ")
    szodb-=1
    if szodb==0:
        szodb=randrange(1,6)
        print()
        sordb+=1
        if sordb==4:
            sordb=0
            print()

print("\n")



cimiro("8. feladat","=")

print("Rendezzük ábécé-rendbe (nem ASCII-kód szerint) a szólistát!")
print("Az eredményt fűzzük hozzá a verselemzes.txt fájlhoz!\n")



def abc_sorrendben(szo1,szo2):
    abc = "-aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz"
    i=0
    while i<len(szo1) and i<len(szo2) and szo1[i]==szo2[i]:
        i+=1

    if i<len(szo1) and i<len(szo2):
        return abc.index(szo1[i])< abc.index(szo2[i])
    else:
        return len(szo1)<len(szo2)

"""
Muszáj nekünk megírni a rendezési algoritmust, mert a beépített rendezések
ASCII-kód szerint rendeznének. Az egyszerű cserés rendezést választjuk,
de előbb a külön listákat egyesíteni kell.
"""
```

```
szolista2d=[]

for i in range(len(szolista)):
    sor=[szolista[i], szogyaklista[i], ekdblista[i]]
    szolista2d.append(sor)

# És már jöhet is a rendezés:

n=len(szolista2d)

for i in range(n-1):
    for j in range(i+1,n):
        if not abc_sorrendben(szolista2d[i][0],szolista2d[j][0]):
            szolista2d[i],szolista2d[j]=szolista2d[j],szolista2d[i]


kitxt = open("verselemzes.txt","a") # hozzáírásra nyitjuk meg a fáljt

kitxt.write("\nA versben szereplő szavak, gyakoriságuk, karaktereik, ill. ékezetes betűik száma\n")
kitxt.write("ábécé-rendben:\n\n")

for i in range(len(szolista2d)):
    kitxt.write(f'{i+1:2}. {szolista2d[i][0]:{szel}}: \
{szolista2d[i][1]}x, {len(szolista2d[i][0]):2}, {szolista2d[i][2]}\n')

kitxt.close()

print("A kiíratás befejeződött.\n")



cimiro("9. feladat","=")

print("Határozzuk meg az ábécébe rendezett szólista leghosszabb")
print("ékezetes szakaszának kezdőszavát és hosszát!")
print("Írassuk ki a teljes szakaszt is!\n")


aktkezdet=None
akthossz=0
maxkezdet=None
maxhossz=0

for i in range(n): # len(szolista2d)
    if szolista2d[i][2]>0: # T tulajdonság: a vizsgált szóban van ékezetes betű
        if aktkezdet==None:
            aktkezdet=i
        akthossz+=1
        if akthossz>maxhossz:
            maxkezdet=aktkezdet
            maxhossz=akthossz
    else:
        aktkezdet=None
        akthossz=0

if maxkezdet!=None:
    print(f'\nA kezdőszó: {szolista2d[maxkezdet][0]},\
a leghosszabb ékezetes szakasz hossza: {maxhossz}')
    print(f'Az ékezetes szakasz: ',end="")
    for i in range(maxkezdet,maxkezdet+maxhossz):
        print(szolista2d[i][0],end=" ")
else:
    print("A szólistában nincs ékezetes szó.")

print()
```

```
cimiro("10. feladat","=")

print("Csoportosítsuk a szólistát a szavak hossza szerint,")
print("csoportonként megtartva az ábécé rendbe sorolást!")
print("Az eredményt írassuk hozzá a verselemzes.txt fájlhoz!\n")

"""
A szolista2d a mellékszempont szerint már rendezett,
szerencsére a főszempont szerinti rendezést már
a beépített sort() rendezéssel is elvégezhetjük.
"""

szolista2d.sort(key=lambda sor: len(sor[0]))

kitxt = open("verselemzes.txt","a")

kitxt.write("\nA versben szereplő szavak, gyakoriságuk, ékezetes betűik száma\n")
kitxt.write("a szavak hossza szerint és azon belül ábécé rendben:\n\n")

db=0
for i in range(len(szolista2d)):
    if len(szolista2d[i][0])>db:
        db=len(szolista2d[i][0])
        kitxt.write(f'{db} karakteres szavak:\n')
    kitxt.write(f'\t{szolista2d[i][0]:{szel}}: {szolista2d[i][1]}x, {szolista2d[i][2]}\n')

kitxt.close()

print("A kiíratás befejeződött.\n")


print()
input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak13\BevGyak13.py ==========================


Verselemzés: Üllői-úti fák
**************************



1. feladat
==========

A vers költőjének, címének és szavainak beolvasása a vers.txt fájlból
A szerző és a cím legyenenek nagybetűsek, a szavak kisbetűsek!

A beolvasás megtörtént. A vers 84 szóból áll.



2. feladat
==========

Írjuk ki a képernyőre a vers szerzőjét és címét,
a betűket ciklikusan hárommal eltolva (a szóközöket, kötőjeleket meghagyva)!

A költő: NŐÚBÜŐODÖÁK FGBÚR

A cím: WOORK-VÜK IDN



3. feladat
==========

A verselemzes.txt fájlban listázzuk ki a versben szereplő szavakat,
mindegyiket csak egyszer! Adjuk meg a szavak előfordulásainak számát,
a karaktereik számát és az ékezetes betűik számát!

A kiíratás befejeződött.



4. feladat
==========

Adjuk meg a leghosszabb szót, ill. szavakat,
és készítsük el a szavak hossza szerinti statisztikát!

A leghosszabb szavak (10 karakter): balzsamost feleljetek

A szavak hossza szerinti statisztika:

 1 karakter:  2 szó
 2 karakter:  7 szó
 3 karakter:  6 szó
 4 karakter:  5 szó
 5 karakter: 10 szó
 6 karakter:  8 szó
 7 karakter: 11 szó
 8 karakter:  5 szó
 9 karakter:  4 szó
10 karakter:  2 szó
```

```
5. feladat
==========

Határozzuk meg a szólistában az ékezetes szavak százalékos arányát
két tizedes pontossággal!

Az ékezetes szavak aránya a szólistában: 58.33%



6. feladat
==========

Írjuk ki a képernyőre a szólista szavainak egy véletlen permutációját!
Minden sor 1 és 5 közötti véletlen számú szót tartalmazzon!
Négy sor után legyen egy üres sor is!

A szólista szavainak véletlen permutációja:

is búsan fejetek
hova lássák higgyék a
kedvet ne lombú
bús tusát fehér ezer

bú sárgult tivéletek lombos feleljetek
már
megöl
adtatok

üllői-úti csírát virágos az
szagos kedvem fergeteg ti legyen
ifjúság
balzsamost másoknak óráin határ

ciprusát szél haldoklik illatot szívják
fák
s
nyugszik borítsa

minden voltatok
napja
est így virág nyíljatok
jár örök dúdolva át

ég édes
repül
altatót
```

```
7. feladat
==========

Írjuk ki a képernyőre a vers szavainak egy véletlen permutációját,
azaz a szólista szavainak egy olyan ismétléses permutációját,
melyben minden szó annyiszor szerepel, mint a versben!
Minden sor 1 és 5 közötti véletlen számú szót tartalmazzon!
Négy sor után legyen egy üres sor is!

A vers szavainak véletlen permutációja:

voltatok napja fejetek üllői-úti
a ne át feleljetek
határ ég üllői-úti bú
szívják

búsan est ti
sárgult nyugszik
illatot is fák adtatok üllői-úti
ezer tusát

higgyék
az ifjúság az fergeteg
repül legyen fák óráin másoknak
az

szagos haldoklik kedvem nyíljatok megöl
már
az minden s így a
az lássák

hova a a fák virágos
üllői-úti ti lombú virág
ciprusát fehér tivéletek
az fák üllői-úti

az a lombos bús borítsa
kedvet ifjúság jár édes fák
balzsamost altatót örök ifjúság
szél üllői-úti fák

csírát
fák dúdolva



8. feladat
==========

Rendezzük ábécé-rendbe (nem ASCII-kód szerint) a szólistát!
Az eredményt fűzzük hozzá a verselemzes.txt fájlhoz!

A kiíratás befejeződött.
```

```
9. feladat
==========

Határozzuk meg az ábécébe rendezett szólista leghosszabb
ékezetes szakaszának kezdőszavát és hosszát!
Írassuk ki a teljes szakaszt is!


A kezdőszó: borítsa,a leghosszabb ékezetes szakasz hossza: 7
Az ékezetes szakasz: borítsa bú bús búsan ciprusát csírát dúdolva


10. feladat
===========

Csoportosítsuk a szólistát a szavak hossza szerint,
csoportonként megtartva az ábécé rendbe sorolást!
Az eredményt írassuk hozzá a verselemzes.txt fájlhoz!

A kiíratás befejeződött.


A befejezéshez nyomd meg az ENTER billentyűt!
```

A verselemzes.txt szövegfájl:

A versben szereplő szavak, gyakoriságuk, karaktereik, ill. ékezetes betűik száma
az első előfordulásuk sorrendjében:

1. az : 7x, 2, 0	21. ifjúság : 3x, 7, 2	41. haldoklik : 1x, 9, 0
2. ég : 1x, 2, 1	22. másoknak : 1x, 8, 1	42. sárgult : 1x, 7, 1
3. legyen : 1x, 6, 0	23. is : 1x, 2, 0	43. határ : 1x, 5, 1
4. tivéletek : 1x, 9, 1	24. így : 1x, 3, 1	44. nyugszik : 1x, 8, 0
5. üllői-úti : 6x, 9, 3	25. nyíljatok : 1x, 9, 1	45. kedvem : 1x, 6, 0
6. fák : 7x, 3, 1	26. szívják : 1x, 7, 2	46. napja : 1x, 5, 0
7. borítsa : 1x, 7, 1	27. édes : 1x, 4, 1	47. már : 1x, 3, 1
8. lombos : 1x, 6, 0	28. illatot : 1x, 7, 0	48. szél : 1x, 4, 1
9. fejetek : 1x, 7, 0	29. a : 5x, 1, 0	49. búsan : 1x, 5, 1
10. szagos : 1x, 6, 0	30. balzsamost : 1x, 10, 0	50. dúdolva : 1x, 7, 1
11. virágos : 1x, 7, 1	31. altatót : 1x, 7, 1	51. jár : 1x, 3, 1
12. fergeteg : 1x, 8, 0	32. est : 1x, 3, 0	52. s : 1x, 1, 0
13. ezer : 1x, 4, 0	33. óráin : 1x, 5, 2	53. megöl : 1x, 5, 1
14. fehér : 1x, 5, 1	34. át : 1x, 2, 1	54. minden : 1x, 6, 0
15. virág : 1x, 5, 1	35. ne : 1x, 2, 0	55. csírát : 1x, 6, 2
16. ti : 2x, 2, 0	36. lássák : 1x, 6, 2	56. hova : 1x, 4, 0
17. adtatok : 1x, 7, 0	37. bú : 1x, 2, 1	57. repül : 1x, 5, 1
18. kedvet : 1x, 6, 0	38. ciprusát : 1x, 8, 1	58. feleljetek : 1x, 10, 0
19. tusát : 1x, 5, 1	39. higgyék : 1x, 7, 1	59. bús : 1x, 3, 1
20. voltatok : 1x, 8, 0	40. örök : 1x, 4, 2	60. lombú : 1x, 5, 1

A versben szereplő szavak, gyakoriságuk, karaktereik, ill. ékezetes betűik száma
ábécé-rendben:

1. a : 5x, 1, 0	21. feleljetek : 1x, 10, 0	41. minden : 1x, 6, 0
2. adtatok : 1x, 7, 0	22. fergeteg : 1x, 8, 0	42. napja : 1x, 5, 0
3. altatót : 1x, 7, 1	23. haldoklik : 1x, 9, 0	43. ne : 1x, 2, 0
4. az : 7x, 2, 0	24. határ : 1x, 5, 1	44. nyíljatok : 1x, 9, 1
5. át : 1x, 2, 1	25. higgyék : 1x, 7, 1	45. nyugszik : 1x, 8, 0
6. balzsamost : 1x, 10, 0	26. hova : 1x, 4, 0	46. óráin : 1x, 5, 2
7. borítsa : 1x, 7, 1	27. ifjúság : 3x, 7, 2	47. örök : 1x, 4, 2
8. bú : 1x, 2, 1	28. illatot : 1x, 7, 0	48. repül : 1x, 5, 1
9. bús : 1x, 3, 1	29. is : 1x, 2, 0	49. s : 1x, 1, 0
10. búsan : 1x, 5, 1	30. így : 1x, 3, 1	50. sárgult : 1x, 7, 1
11. ciprusát : 1x, 8, 1	31. jár : 1x, 3, 1	51. szagos : 1x, 6, 0
12. csírát : 1x, 6, 2	32. kedvem : 1x, 6, 0	52. szél : 1x, 4, 1
13. dúdolva : 1x, 7, 1	33. kedvet : 1x, 6, 0	53. szívják : 1x, 7, 2
14. est : 1x, 3, 0	34. lássák : 1x, 6, 2	54. ti : 2x, 2, 0
15. ezer : 1x, 4, 0	35. legyen : 1x, 6, 0	55. tivéletek : 1x, 9, 1
16. édes : 1x, 4, 1	36. lombos : 1x, 6, 0	56. tusát : 1x, 5, 1
17. ég : 1x, 2, 1	37. lombú : 1x, 5, 1	57. üllői-úti : 6x, 9, 3
18. fák : 7x, 3, 1	38. már : 1x, 3, 1	58. virág : 1x, 5, 1
19. fehér : 1x, 5, 1	39. másoknak : 1x, 8, 1	59. virágos : 1x, 7, 1
20. fejetek : 1x, 7, 0	40. megöl : 1x, 5, 1	60. voltatok : 1x, 8, 0

A versben szereplő szavak, gyakoriságuk, ékezetes betűik száma
a szavak hossza szerint és azon belül ábécé rendben:

1 karakteres szavak:
- a : 5x, 0
- s : 1x, 0

2 karakteres szavak:
- az : 7x, 0
- át : 1x, 1
- bú : 1x, 1
- ég : 1x, 1
- is : 1x, 0
- ne : 1x, 0
- ti : 2x, 0

3 karakteres szavak:
- bús : 1x, 1
- est : 1x, 0
- fák : 7x, 1
- így : 1x, 1
- jár : 1x, 1
- már : 1x, 1

4 karakteres szavak:
- ezer : 1x, 0
- édes : 1x, 1
- hova : 1x, 0
- örök : 1x, 2
- szél : 1x, 1

5 karakteres szavak:
- búsan : 1x, 1
- fehér : 1x, 1
- határ : 1x, 1
- lombú : 1x, 1
- megöl : 1x, 1
- napja : 1x, 0
- óráin : 1x, 2
- repül : 1x, 1
- tusát : 1x, 1
- virág : 1x, 1

6 karakteres szavak:
- csírát : 1x, 2
- kedvem : 1x, 0
- kedvet : 1x, 0
- lássák : 1x, 2
- legyen : 1x, 0
- lombos : 1x, 0
- minden : 1x, 0
- szagos : 1x, 0

7 karakteres szavak:
- adtatok : 1x, 0
- altatót : 1x, 1
- borítsa : 1x, 1
- dúdolva : 1x, 1
- fejetek : 1x, 0
- higgyék : 1x, 1
- ifjúság : 3x, 2
- illatot : 1x, 0
- sárgult : 1x, 1
- szívják : 1x, 2
- virágos : 1x, 1

8 karakteres szavak:
- ciprusát : 1x, 1
- fergeteg : 1x, 0
- másoknak : 1x, 1
- nyugszik : 1x, 0
- voltatok : 1x, 0

9 karakteres szavak:
- haldoklik : 1x, 0
- nyíljatok : 1x, 1
- tivéletek : 1x, 1
- üllői-úti : 6x, 3

10 karakteres szavak:
- balzsamost : 1x, 0
- feleljetek : 1x, 0

14. Érettségi mintafeladat: Autók

Az *autok.txt* szövegfájl egy autókereskedés éves beszállításait tartalmazza időrendben. A fájl első sora az év, azután egy sorban ** és szóköz után található a hónap sorszáma, másik sorban * és szóköz után a napé. A hónapokat csak a hónap első szállítási napja előtt adjuk meg. A nap sorszáma után következő sorokban az aznapi szállítások találhatók. Először a szállító kétszámjegyű kódja, utána szóközzel elválasztva az autó márkája, majd újabb szóközzel elválasztva a típusa. Újabb szóköz után a darabszám, végül az egységár millió Ft-ban 1 tizedesjeggyel, tizedesvesszővel. Max. 1000 szállítás és max. 40 szállító van.

A fájl első néhány sora:

```
2008
** 02
* 11
77 Skoda Fabia 16 3,8
77 Skoda Octavia 8 5,4
```

1. Olvasd be a szövegfájt alkalmasan megválasztott adatszerkezetbe!
2. Írasd ki a képernyőre, hogy melyik volt az első és az utolsó szállítási nap!
3. Melyik szállító szállította a legtöbb autót? Összesen hány darabot?
4. Mennyi volt az autók összértéke? Ha a tizedesvesszős alakot a program nem tudja kezelni, megfelelő módon alakítsd át! Az eredményt ezres tagolással, tizedesjegyek nélkül add meg millió Ft-ban!
5. Készíts függvényt, mely megadja az adott évben, hogy egy adott hónap adott napja az év hányadik napja! A számításba jöhető évek között minden néggyel osztható szökőév! Feltételezzük, hogy ismered, hogy hány naposak a hónapok. Példaként alkalmazd a függvényt a mai napra!
6. Határozd meg, hány napig tartott és mikor kezdődött a leghosszabb időszak az évben, amikor nem volt szállítás!
7. Határod meg, melyik szállító hány napon szállított! Ha egy napon több típust is szállított, azt akkor is csak egy napnak kell tekinteni!
8. Ha volt olyan szállító, amelyik legalább két napon is szállított, akkor melyik szállító két szállítási napja között telt el a legkevesebb nap?
9. Mikor kezdődött és hány napos volt az a leghosszabb időszak, amikor minden nap volt szállítás?
10. Állítsd elő egy véletlen hónap 15 különböző véletlen napját! Add meg 2 tizedes pontossággal, hogy ezeknek a napoknak hány százalékában volt szállítás!
11. Kérd be egy szállító kódját! Alkalmazott-e az év során áremelést valamely típusnál? Ha igen, add meg, melyik típusnál, és hány %-os volt a legnagyobb arányú áremelése! Sorold fel a többi szállító ennél magasabb áremeléseit a típus és az áremelés mértékének %-os arányával együtt!
12. Kérj be egy márkanevet (pl. skoda, kisbetűvel is fogadja el)! Írasd ki a **megadott_márkanév***.txt* fájlba időrendben a megadott márka beszállításait! Minden beszállítás külön sorba kerüljön! Táblázatos elrendezésben elöl a szállító kódja, majd a típus (márkanév nélkül), a dátum, a darabszám és az ár szerepeljen!
13. Írasd ki az **év**_*evi_szallitasok.txt* fájlba a beszállítások listáját márkák és típusok szerint rendezve. A márkanév és a típus szerepeljen a sor elején, utána táblázatosan a szállító kódja és a többi adat!
14. Készíts statisztikát arról, hogy az egyes szállítók hány autótípust szállítottak! Az eredményt típusszám szerint csökkenő sorrendben írasd ki a *szallitok.txt* fájlba, úgy, hogy minden szállító külön sorba kerüljön! Elöl a szállító kódja, tabulátorral elválasztva a szállított típusok száma, majd újabb tabulátor után vesszővel és szóközökkel elválasztva az egyes márkák és típusok, de minden típus csak egyszer szerepeljen egy sorban! Az utolsó típus után ne legyen vessző!

```
"""
14. Érettségi mintafeladat: Autók
@author Klemand66
"""

from random import *
from math import *
from datetime import *

def cimiro(cim,karakter):
    print("\n",cim,sep="")
    hossz=len(cim)
    for i in range(hossz):
        print(karakter,end="")
    print("\n")

cimiro("Érettségi mintafeladat: Autók","*")



cimiro("1. feladat","=")

betxt = open("autok.txt")

ev=int(betxt.readline().strip())

szallitasok=[]

for sor in betxt:
    darsor=sor.strip().split()
    if darsor[0]=="**":
        aktho=int(darsor[1])
    elif darsor[0]=="*":
        aktnap=int(darsor[1])
    else:
        ho=aktho
        nap=aktnap
        szallito=int(darsor[0])
        tipus=darsor[1]+" "+darsor[2]
        db=int(darsor[3])
        daradat=darsor[4].split(",")
        ar=(float(daradat[0]+"."+daradat[1])) # a tizedesvesszőt pontra cseréltük
        szallitas=[ho,nap,szallito,tipus,db,ar]
        szallitasok.append(szallitas)
        # indexek: 0: ho,  1: nap,  2: szallito,  3: márka és tipus,  4: db,  5: ar

betxt.close()
print("A beolvasás megtörtént.\n")



cimiro("2. feladat","=")

print(f'Az első szállítási nap: {ev}.{szallitasok[0][0]}.{szallitasok[0][1]}.,')
print(f'az utolsó pedig: {ev}.{szallitasok[-1][0]}.{szallitasok[-1][1]}. volt.\n')



cimiro("3. feladat","=")

szdb=[]
# szdb[szallitokod] tartalmazza az adott indexű szállító darabszámait

szdb=[]
for i in range(100): # tudjuk, hogy a szállítókód kétjegyű szám
    szdb.append(0)

for i,elem in enumerate(szallitasok):
    szdb[elem[2]]+=elem[4]

maxdb=max(szdb)
maxindex=szdb.index(maxdb)
print(f'A(z) {maxindex} szállító szállította a legtöbb autót, összesen {maxdb} darabot.\n');
```

```
cimiro("4. feladat","=")

osszertek = 0
for elem in szallitasok:
    osszertek +=elem[4]*elem[5]

print(f'A szállított autók összértéke {osszertek:,.0f} millió Ft volt.\n');



cimiro("5. feladat","=")

def evnapja(ev, ho, nap):
    athozatok = [ 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30 ] # áthozatok az előző hónapról
    if ev%4==0:
        athozatok[2]+=1
    osszathozat=sum(athozatok[:ho])
    return osszathozat+nap

maidatum=date.today()
ideiev=maidatum.year
aktho=maidatum.month
mainap=maidatum.day

print(f'A függvény elkészült.\nMa {ideiev}.{aktho}.{mainap}. van, az év \
{evnapja(ideiev,aktho,mainap)}. napja.\n')



cimiro("6. feladat","=")

def datumozo(ev, evnapja):
    honapok = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 ]
    if ev%4==0:
        honapok[1]+=1
    for i,n in enumerate(honapok):
        if sum(honapok[:i+1])>=evnapja:
            ho=i+1
            nap=evnapja-sum(honapok[:i])
            datum=[ho,nap]
            break
    return datum # ha több adatot akarunk visszaadni, egyszerűen listába tehetjük őket

for i,elem in enumerate(szallitasok):
    if i==0: # év eleje
        elozo=evnapja(ev,elem[0],elem[1])
        maxmentesnap=elozo-1
        kezdonap=1
        continue # az első szállításnál még nem viszonyítunk korábbihoz, csak az év kezdetéhez
    akt=evnapja(ev,elem[0],elem[1])
    mentesnap=akt-elozo-1
    if mentesnap>maxmentesnap:
        maxmentesnap=mentesnap
        kezdonap=elozo+1 # az előző szállítás a hónap végére is eshetett, ezért kell a dátumozó
    elozo=akt

mentesnap=evnapja(ev,12,31)-elozo # a ciklus végéről
if mentesnap>maxmentesnap:
    maxmentesnap=vege
    kezdonap=elozo+1

kezdodatum=datumozo(ev,kezdonap)

print(f'A leghosszabb szállításmentes időszak {maxmentesnap} napig tartott,')
print(f'a kezdőnapja {ev}.{kezdodatum[0]}.{kezdodatum[1]} volt.\n')
```

```
cimiro("7. feladat","=")

sznapok=[]
for i in range(100):
    sznapok.append([]) # sznapok[i] tartalmazza majd az i kódú szállító szállítási napjait

for i,elem in enumerate(szallitasok):
    if [elem[0],elem[1]] not in sznapok[elem[2]]:
        sznapok[elem[2]].append([elem[0],elem[1]])

for i in range(100):
    if len(sznapok[i])>0:
        print(f'A(z) {i:2} kódú szállító {len(sznapok[i])} napon végzett beszállítást.')

print()


cimiro("8. feladat","=")

"""
Tudjuk, hogy a szállítók beszállítási napjai időrendben szerepelnek az sznapok listában.
"""

minkul=1000 # irreálisan nagy érték

for i,elem in enumerate(sznapok):
    if len(elem)>1:
        for (j,datum) in enumerate(elem):
            if j==0:
                elozo=evnapja(ev,datum[0],datum[1])
                continue
            akt=evnapja(ev,datum[0],datum[1])
            kul=akt-elozo
            if kul<minkul:
                minkul=kul
                szkod=i
            elozo=akt

print(f'A(z) {szkod} kódú szállító két szállítási napja között telt el a legkevesebb nap: \
{minkul}\n')



cimiro("9. feladat","=")

ossznap=evnapja(ev,12,31)
volt=[]
for i in range(ossznap+1):
    volt.append(False)

for i,elem in enumerate(szallitasok):
    volt[evnapja(ev, elem[0],elem[1])]=True

aktkezdet=None
akthossz=0
maxkezdet=None
maxhossz=0

for i,elem in enumerate(volt):
    if elem:
        if aktkezdet==None:
            aktkezdet=i
        akthossz+=1
        if akthossz>maxhossz:
            maxkezdet=aktkezdet
            maxhossz=akthossz
    else:
        aktkezdet=None
        akthossz=0

kezdodatum=datumozo(ev,maxkezdet)
print(f'A leghosszabb mindennapos szállítás kezdőnapja \
{ev}.{kezdodatum[0]}.{kezdodatum[1]}. volt és {maxhossz} napig tartott.\n')
```

```
cimiro("10. feladat","=")

def hohossz(ev, ho):
    honapok = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 ]
    if ev%4==0:
        honapok[1]+=1
    return honapok[ho-1]

velho=randrange(1,13)
hossz=hohossz(ev,velho)
velnapok=[]

while len(velnapok)<15:
    velnap=randrange(1,hossz+1)
    if velnap not in velnapok:
        velnapok.append(velnap)

velnapok.sort()
print(f'A kiválasztott hónap: {velho}, hossza {hossz}.')
print(f'A kiválasztott napok listája:\n{velnapok}.\n')

szvelnapok=[]
for nap in velnapok:
    n=evnapja(ev,velho,nap)
    if volt[n]:
        szvelnapok.append(nap)

print(f'A(z) {velho}. hónapban a kiválasztott napok közül a következőkben volt beszállítás:\n\
{szvelnapok}. Ez a kiválasztott napok {100*len(szvelnapok)/15:.2f}%-a. \n')



cimiro("11. feladat","=")

"""
Rendezzük a szállítások listáját szallitasok1 listába
a szállítók kódja, azon belül márka,ill. típus, végül ár szerint!
A rendezésnél fordított sorrendben járunk el, a főszempont szerinti rendezés marad a végére.
"""

szallitasok1=sorted(szallitasok,key=lambda elem: elem[5]) # először ár szerint rendezzük új listába
szallitasok1.sort(key=lambda elem: elem[3]) # másodszor már helyben rendezzük márka és típus szerint
szallitasok1.sort(key=lambda elem: elem[2]) # végül szállító kódja szerint rendezzük (fő szempont)
# az idő szerinti rendezés eleve adott volt, megmarad

"""
Minden szállító és típus esetén listákba tesszük,
hogy mennyi volt a kezdőár, a végár, ill. az emelés.
"""
```

```
i=0 # kezdőértékek
szkod=szallitasok1[0][2]
szkodok=[szkod]
tipus=szallitasok1[0][3]
tipusok=[tipus]
kezdoar=szallitasok1[0][5]
kezdoarak=[kezdoar]
vegar=szallitasok1[0][5]
vegarak=[vegar]
emeles=vegar/kezdoar
emelesek=[emeles]

for elem in szallitasok1:
    if elem[2]==szkod:
        if elem[3]==tipus:
            if elem[5]>vegar: # amikor csak az ár nő
                vegar=elem[5]
                vegarak[i]=vegar
                emeles=vegar/kezdoar
                emelesek[i]=emeles
        else: # amikor típusváltozás van
            i+=1
            szkodok.append(szkod)
            tipus=elem[3]
            tipusok.append(tipus)
            kezdoar=elem[5]
            kezdoarak.append(kezdoar)
            vegar=elem[5]
            vegarak.append(vegar)
            emeles=vegar/kezdoar
            emelesek.append(emeles)
    else: # amikor a szállító is változik
        i+=1
        szkod=elem[2]
        szkodok.append(szkod)
        tipus=elem[3]
        tipusok.append(tipus)
        kezdoar=elem[5]
        kezdoarak.append(kezdoar)
        vegar=elem[5]
        vegarak.append(vegar)
        emeles=vegar/kezdoar
        emelesek.append(emeles)

aktkod=int(input("Kérem egy szállító kétjegyű kódját: "))

if len(sznapok[aktkod])==0:
    print(f'\nA megadott {aktkod} kódú szállító nem végzett beszállítást.')
else:
    maxemeles=1
    for i,emeles in enumerate(emelesek):
        if szkodok[i]==aktkod and emeles>maxemeles:
            maxemeles=emeles
            maxtipus=tipusok[i]
    if maxemeles>1:
        print(f'\nA megadott {aktkod} kódú szállító legnagyobb áremelése {100*(maxemeles-1):.2f}% volt \
a(z) {maxtipus} esetén.\n')
    else:
        print(f'\nA megadott {aktkod} kódú szállító nem emelt árat az év folyamán.\n')

    print("Ennél nagyobb áremelések:")
    for i,emeles in enumerate(emelesek):
        if emeles>maxemeles:
            print(f'{szkodok[i]:2}\t{tipusok[i]:14}\t{100*(emeles-1):5.2f}%')
    print("\n")
```

```
cimiro("12. feladat","=")

aktmarka=input("Kérek egy márkanevet (pl. skoda): ")

aktmarka=aktmarka.strip().lower()
fajlnev=(f'{aktmarka}.txt') # az f-stringgel könnyen tehetünk változókat a fájlnevekbe is

kitxt = open(fajlnev,"w")

for elem in szallitasok:
    dartipus=elem[3].split()
    marka=dartipus[0]
    tipus=dartipus[1]
    if marka.lower()==aktmarka:
        ho=str(elem[0])
        if len(ho)<2:
            ho="0"+ho
        nap=str(elem[1])
        if len(nap)<2:
            nap="0"+nap
        kitxt.write(f'{elem[2]:3} {tipus:8} {ev}.{ho}.{nap}. {elem[4]:4} db {elem[5]:6} MFt \n')

kitxt.close()
print("\nA kiíratás és a szövegfájl lezárása sikeresen befejeződött.\n")



cimiro("13. feladat","=")

fajlnev=(f'{ev}_evi_szallitasok.txt')
kitxt = open(fajlnev,"w")

szallitasok.sort(key=lambda elem: elem[3]) # márka és típus

szelessegek=list(map(lambda t: len(t),tipusok))
szel=max(szelessegek)

for elem in szallitasok:
    ho=str(elem[0])
    if len(ho)<2:
        ho="0"+ho
    nap=str(elem[1])
    if len(nap)<2:
        nap="0"+nap
    kitxt.write(f' {elem[3]:{szel}}  {elem[2]:3}  {ev}.{ho}.{nap}.  {elem[4]:4} db  {elem[5]:6} MFt \n')

kitxt.close()
print("\nA kiíratás és a szövegfájl lezárása sikeresen befejeződött.\n")
```

```
cimiro("14. feladat","=")

"""""
A 11. feladatban már ennek a megoldását is előkészítettük:
Az szkodok listában szerepelnek a szállítók kódjai, mégpedig éppen annyiszor,
ahány típust szállítottak. A neki megfelelő tipusok listában pedig
párhuzamosan szerepel az a típus, amit szállítottak.
Ezekből elkészítjük most a szállítók listáját, ahol már minden szállító csak egyszer szerepel,
és párhuzamosan minden szállítóhoz a típusaik listáját, amikben több típus is szerepelhet.
A rendezés miatt egy kétdimenziós listát kell készítenünk.
"""

szallitok=[]
tipusaik=[]
for i,kod in enumerate(szkodok):
    if kod in szallitok:
        szindex=szallitok.index(kod)
        tipusaik[szindex].append(tipusok[i])
    else:
        szallitok.append(kod)
        tipusaik.append([tipusok[i]]) # a tipusaik lista listák listája (kétdimenziós lista)

statlista=[]
for i,elem in enumerate(szallitok):
    statlista.append([szallitok[i],tipusaik[i]])

statlista.sort(key=lambda elem: len(elem[1]),reverse=True)

kitxt = open("szallitok.txt","w")

for elem in statlista:
   kitxt.write(f' {elem[0]}\t{len(elem[1])}\t')
   for j,tipus in enumerate(elem[1]):
       kitxt.write(f'{tipus}')
       if j<len(elem[1])-1:
          kitxt.write(", ")
   kitxt.write("\n")

kitxt.close()

print("\nA kiíratás és a szövegfájl lezárása sikeresen befejeződött.\n")


input("A befejezéshez nyomd meg az ENTER billentyűt!")
```

```
========================== RESTART: C:\Python\KlInfoPy\BevGyak14\BevGyak14.py ==========================

Érettségi mintafeladat: Autók
*****************************


1. feladat
==========

A beolvasás megtörtént.


2. feladat
==========

Az első szállítási nap: 2008.2.11.,
az utolsó pedig: 2008.11.30. volt.


3. feladat
==========

A(z) 99 szállító szállította a legtöbb autót, összesen 140 darabot.


4. feladat
==========

A szállított autók összértéke 1,706 millió Ft volt.


5. feladat
==========

A függvény elkészült.
Ma 2022.10.17. van, az év 290. napja.


6. feladat
==========

A leghosszabb szállításmentes időszak 83 napig tartott,
a kezdőnapja 2008.7.1 volt.


7. feladat
==========

A(z) 25 kódú szállító 1 napon végzett beszállítást.
A(z) 32 kódú szállító 2 napon végzett beszállítást.
A(z) 64 kódú szállító 2 napon végzett beszállítást.
A(z) 77 kódú szállító 5 napon végzett beszállítást.
A(z) 88 kódú szállító 2 napon végzett beszállítást.
A(z) 99 kódú szállító 2 napon végzett beszállítást.


8. feladat
==========

A(z) 77 kódú szállító két szállítási napja között telt el a legkevesebb nap: 12
```

```
9. feladat
==========

A leghosszabb mindennapos szállítás kezdőnapja 2008.9.22. volt és 3 napig tartott.


10. feladat
===========

A kiválasztott hónap: 5, hossza 31.
A kiválasztott napok listája:
[1, 2, 4, 7, 8, 11, 12, 14, 15, 20, 21, 23, 27, 29, 30].

A(z) 5. hónapban a kiválasztott napok közül a következőkben volt beszállítás:
[]. Ez a kiválasztott napok 0.00%-a.


11. feladat
===========

Kérem egy szállító kétjegyű kódját: 77

A megadott 77 kódú szállító legnagyobb áremelése 10.53% volt a(z) Skoda Fabia esetén.

Ennél nagyobb áremelések:
99      Opel Astra      15.56%
99      Opel Corsa      11.11%



12. feladat
===========

Kérek egy márkanevet (pl. skoda): skoda

A kiíratás és a szövegfájl lezárása sikeresen befejeződött.


13. feladat
===========


A kiíratás és a szövegfájl lezárása sikeresen befejeződött.


14. feladat
===========


A kiíratás és a szövegfájl lezárása sikeresen befejeződött.

A befejezéshez nyomd meg az ENTER billentyűt!
```

A skoda.txt szövegfájl:

```
 77 Fabia    2008.02.11.   16 db    3.8 MFt
 77 Octavia  2008.02.11.    8 db    5.4 MFt
 77 Fabia    2008.04.08.   12 db    3.9 MFt
 77 Fabia    2008.11.30.   45 db    4.2 MFt
 77 Octavia  2008.11.30.    4 db    5.8 MFt
```

A 2008_evi_szallitasok.txt szövegfájl:

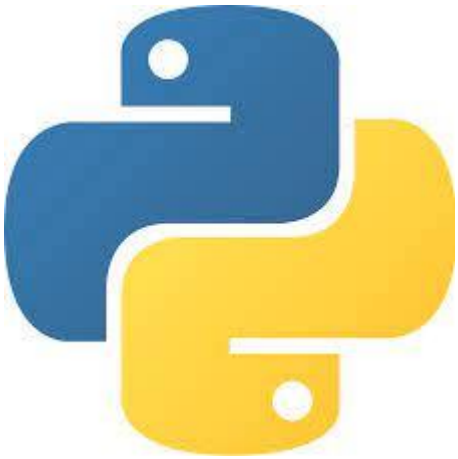
```
 Audi A6         64  2008.02.12.    12 db     8.8 MFt
 Audi A8         64  2008.06.30.     4 db     9.9 MFt
 BMW 1           25  2008.02.23.    14 db     6.2 MFt
 Ford Fiesta     88  2008.04.08.    60 db     3.2 MFt
 Ford KA         88  2008.02.12.    18 db     2.6 MFt
 Mercedes A      32  2008.09.24.     8 db     5.2 MFt
 Mercedes C      32  2008.04.01.     6 db     8.2 MFt
 Opel Astra      99  2008.09.22.    42 db     4.5 MFt
 Opel Astra      99  2008.11.30.    42 db     5.2 MFt
 Opel Corsa      99  2008.09.22.    28 db     3.6 MFt
 Opel Corsa      99  2008.11.30.    28 db     4.0 MFt
 Skoda Fabia     77  2008.02.11.    16 db     3.8 MFt
 Skoda Fabia     77  2008.04.08.    12 db     3.9 MFt
 Skoda Fabia     77  2008.11.30.    45 db     4.2 MFt
 Skoda Octavia   77  2008.02.11.     8 db     5.4 MFt
 Skoda Octavia   77  2008.11.30.     4 db     5.8 MFt
 VW Golf         77  2008.02.23.    30 db     4.2 MFt
 VW Golf         77  2008.09.23.     8 db     4.4 MFt
```

A szallitok.txt szövegfájl:

```
 77     3       Skoda Fabia, Skoda Octavia, VW Golf
 32     2       Mercedes A, Mercedes C
 64     2       Audi A6, Audi A8
 88     2       Ford Fiesta, Ford KA
 99     2       Opel Astra, Opel Corsa
 25     1       BMW 1
```

Folytatás: Az emelt szintű informatika érettségi programozási feladatainak megoldása Python nyelven

Az emelt szintű informatika érettségi programozási feladatainak megoldása Python nyelven



Klement András

2022-23