Bevezetés a programozásba

11. Gyakorlat

Ismétlés

- Parancssori argumentumok
  - sys.argv
- Állománykezelés
  - open()
  - read()
  - readline()
  - readlines()
  - write()
  - writelines()

1. Házi Feladat

Lokális maximumhelyek száma

- https://viskillz.inf.unideb.hu/prog/#/?week=P1081&exercise=P108109c&page=sheet

Megoldás – 1

```
import sys

def count_of_local_maximums() -> int:
    numbers = [int(s) for s in sys.argv[1:]]
    count = 0
    for i in range(len(numbers) - 2):
        if numbers[i] < numbers[i + 1] > numbers[i + 2]:
            count += 1
    return count

def main():
    print(count_of_local_maximums())

if __name__ == "__main__":
    main()
```

Megoldás – 2

```
import sys
def count_of_local_maximums(numbers: list[int]) -> int:
    count = 0
    for i in range(len(numbers) - 2):
        if numbers[i] < numbers[i + 1] > numbers[i + 2]:
            count += 1
    return count
def main():
    numbers = []
    argc = len(sys.argv)
    for i in range(1, argc):
        numbers.append(int(sys.argv[i]))
    print(count_of_local_maximums(numbers))
if __name__ == "__main__":
    main()
```

2. Házi Feladat

Körözgetés

- https://progcont.hu/progcont/100132/?pid=200799

Megoldás

```
import sys
pilotak={}

with open(sys.argv[1]) as file:
    for line in file:
        adat = line.strip("\n").split(";")
        # print(adat)
        if adat[0] in pilotak:
            pilotak[adat[0]] += int(adat[2])
        else:
            pilotak[adat[0]] = int(adat[2])
    # print(pilotak)
for kulcs, ertek in sorted(pilotak.items(),
                           key = lambda rend: (-rend[1], rend[0])):
    print(kulcs)
```

3. Házi Feladat

Prímválogatás

- https://progcont.hu/progcont/100317/?pid=201426

Megoldás – 1

```
import sys
import math

def is_prime(n: int) -> bool:
    if n == 2:
        return True
    if n < 2 or n % 2 == 0:
        return False
    for i in range(3, int(math.sqrt(n))+1, 2):
        if n % i == 0:
            return False
    return True

def main():
    with open(sys.argv[1]) as file:
        for line in file:
            numbers = line.strip().split(' ')
            primes=[]
            for number in numbers:
                if is_prime(int(number)) and int(number) not in primes:
                    primes.append(int(number))
            primes.sort()
            if primes:
                for i in range(len(primes)):
                    if i < len(primes)-1:
                        print(primes[i], end=', ')
                    else:
                        print(primes[i])
            else:
                print("NOTHING")
if __name__ == '__main__':
    main()
```

Megoldás – 2

```
import sys
import math

def is_prime(n: int) -> bool:
    if n == 2:
        return True
    if n < 2 or n % 2 == 0:
        return False
    for i in range(3, int(math.sqrt(n))+1, 2):
        if n % i == 0:
            return False
    return True
def main():
    with open(sys.argv[1]) as file:
        for line in file:
            numbers = line.strip().split(' ')
            primes = []
            for number in numbers:
                if is_prime(int(number)) and int(number) not in primes:
                    primes.append(int(number))
            primes.sort()
            if primes:
                print(', '.join(str(p) for p in primes))
            else:
                print("NOTHING")
if __name__ == '__main__':
    main()
```

Tuple (Rendezett n-es)

- Rendezett tároló
- Index segítségével tudunk hivatkozni az egyes elemekre
- Egy elem/érték többször is előfordulhat
- Többféle típust is tárolhat egyszerre
- Az elemeket kerek zárójelben adjuk meg
- Nem lehet megváltoztatni (immutabilis):
  - Elemeket
  - Elemek sorrendjét
  - Elemek számát

Tuple (Rendezett n-es)

- Rögzített számú adatelemeket tároló immutábilis objektum.
- `t = (5, 9)`
- `print(t[0], t[1])   # 5  9`
- `type(t)`
- `<class 'tuple'>`

- A tuple típusú objektumot a kerek zárójelben felsorolt, vesszővel elválasztott értékekkel hozzuk létre.
- Az egyes adatokat pedig indexeléssel érjük el, t[0] az első adatelem (itt 5), t[1] pedig a második (itt 9).
- A tuple típusú objektumok immutábilisak; a bennük tárolt referenciák nem változhatnak, sem a számuk, sem az értékük.

Tuple – Példa

```
a = ()
#Üres tuple létrehozása: ().
print(a)

b = tuple()
#Üres tuple létrehozása: tuple().
print(b)

c = 1,
print(c)
#Egy elemű tuple.
```

Output

```
()
()
(1,)
```

Tuple – Példa

- Tuple elemeinek összegzése:

```
def sum(test_tuple):
    # listává alakítjuk
    test = list(test_tuple)
    count = 0
    for i in test:
        count += i
    return count

test_tuple = (5, 20, 3, 7, 6, 8)
print(sum(test_tuple))
```

Tuple – Példa

- Tuple elemeinek összegzése:

```
def sum2(test_tuple):
    test = [x for x in test_tuple]
    # listává alakítjuk list comprehension-el
    return sum(test)

test_tuple = (5, 20, 3, 7, 6, 8)
print(sum2(test_tuple))
```

Tuple – Példa

- Tuple elemeinek a módosítása:

```
t1 = (10, 20, 30, 40, 50)
print("Eredeti tuple :", t1)
l = list(t1) # list típusúvá konvertáljuk
l[2] = 33
t1 = tuple(l) # tuple típusúvá konvertáljuk
print("Módosítptt tuple :", t1)
```

Tuple (Rendezett n-es)

- Gyakran egy programhelyen több adatunk egymás mellé kerül, sokszor csak ideiglenesen.
- Például szeretnénk egy két visszatérési értékű függvényt csinálni, amelyik megkeresi egy lista minimumát ÉS maximumát is egyszerre.
- Vagy számpárokkal dolgoznánk, amelyek listák tartományait (kezdő index, befejező index) jelölik meg.
- Esetleg egy pont koordinátáit tárolnánk (x, y), de nincs szükségünk sztringgé alakításra, operátorokra, semmi egyébre, ami miatt fontosnak tartanánk egy külön osztályt definiálni.

Tuple (Rendezett n-es)

- Ilyen esetben szükségünk van egy tárolóra. De az nem egy lista, mert az kicsit mást jelent: a listában változhat az elemek száma, és megcserélhetőek.

- A két visszatérési értékű függvényben ez nem igaz: ott az első visszaadott adat a lista minimuma, a másik a maximuma – ezek nem megcserélhetőek (más a jelentésük), és nem változhat a számuk (mindig kettő lesz).

Tuple (Rendezett n-es)

- **Példa**: Írjuk meg a függvényt, amelyik visszaadja egy lista legkisebb és legnagyobb elemét!

```
def minmax(szamok: list[int]) -> tuple:
    min = max = szamok[0]
    for i in range(1, len(szamok)):
        if szamok[i] < min: min = szamok[i]
        if szamok[i] > max: max = szamok[i]
    return (min, max)
```

Tuple (Rendezett n-es)

```
verseny = [
    (4, "Jó Áron"),
    (3, "Break Elek"),
    (2, "Dil Emma"),
    (1, "Kasza Blanka"),
    (4, "Am Erika"),
]

for helyezes, nev in sorted(verseny):
    print(f"{helyezes}. helyezett: {nev}")
```

Tuple (Rendezett n-es)

- Kimenet:

1. helyezett: Kasza Blanka
2. helyezett: Dil Emma
3. helyezett: Break Elek
4. helyezett: Am Erika
4. helyezett: Jó Áron

- A ciklus ezen a tárolón iterál végig, minden *tuple*-t kicsomagolva a *nev* és a *helyezes* nevű változóba.
- A listázásban ráadásul a dobogósok kerültek előre: ezt a sorted() függvény biztosítja. Láthatóan valójában nem az eredeti tárolón, hanem annak rendezett változatán iterálunk.
- A rendezésben a sorted() függvény *tuple* elemeket kellett összehasonlítson. Ezekre értelmezve vannak az összehasonlító, relációs operátorok is. A *tuple*-ök összehasonlítása az adatelem szerint halad egyesével.

Nevesített Tuple – Named Tuple

```
from typing import NamedTuple

# Deklarálás
Student = NamedTuple('Student', [('name', str),
                                 ('age', int),
                                 ('neptun', str)])
# Értékek hozzáadása
S = Student('Mike', 19, 'NDKYCF')
# Index szerinti elérés
print("Student age: ",S[1]) # S.age
# Név szerinti elérés
print("Student name: ", S.name) #S[0]
# Neptun kód szerinti elérés
print("Neptun code: ", S.neptun) #S[2]
```

Feladatgyűjtemény

- https://viskillz.inf.unideb.hu/prog/#/
- Rekordok – rendezés
- Rekordok – kigyűjtés
- Rekordok – megszámlálás
- Rekordok – csoportosítás
- Rekordok – fájlból olvasás, rendezés
- Rekordok – fájlból olvasás, kigyűjtés
- Rekordok – fájlból olvasás, megszámlálás
- Rekordok – fájlból olvasás, csoportosítás

Minyonok

https://viskillz.inf.unideb.hu/prog/#/?week=P1083&exercise=P108301e&page=sheet

- A minyonok története az idők kezdetétől ered. A minyonok sárga egysejtű organizmusként kezdték, és fejlődtek a korokon át, és mindig a leggrúsabb gazdákat szolgálták. Mivel ezeket a gazdákat – a T-Rextől Napóleonig – folytonosan elveszítették, a minyonoknak most nincs kit szolgálniuk, és mély depresszióba zuhantak.

- Ebben a feladatban minyonok adatait kell beolvasnod a standard inputról, rendezni, majd kiírni a standard outputra.

Minyonok – Formátumok

- A standard bemeneten egy minyon a következő formátumban szerepel:
- `minyon_neve;éhség;lelkesedés;nadrág_mérete`

- A standard kimeneten egy minyon a következő formátumban szerepel:
- `minyon_neve éhség (nadrág_mérete)`

- A jelölések:
  - minyon_neve: egy legfeljebb 30 karakter hosszú, csak angol betűket tartalmazó egyedi sztring
  - éhség: egy nemnegatív egész érték
  - lelkesedés: egy nemnegatív egész érték
  - nadrág_mérete: az "S", "L", "XL" és "XXL" sztringek egyike

Minion típus

- Definiáld a Minion nevű **nevesített tuple**-t, mely a következő mezőkkel rendelkezik:

  - name: a minyon neve (típus: <class 'str'>)
  - hunger: a minyon éhsége (típus: <class 'int'>)
  - motivation: a minyon motiváltsága (típus: <class 'int'>)
  - size: a minyon mérete (típus: <class 'str'>)

Nevesített Minion Tuple

```
from typing import NamedTuple

Minion = NamedTuple("Minion", [("name", str),
                               ("hunger", int),
                               ("motivation", int),
                               ("size", str)])
```

A line_to_minion() függvény

- Írj egy **line_to_minion()** nevű függvényt, mely megkapja a bemenet egy sorát, majd visszaadja sorból kinyert adatokkal feltöltött, Minion nevesített tuple-t!

- Paraméterlista
  - a bemenet egy Minion rekordot leíró sora
- Visszaadott érték
  - a line paraméternek megfelelő Minion rekord

A line_to_minion() függvény

```
def line_to_minion(line):
    adat = line.strip().split(";")
    return Minion(adat[0], int(adat[1]),
                  int(adat[2]), adat[3])
```

A minion_to_line() függvény

- Írj egy **minion_to_line()** nevű függvényt, mely megkap egy Minion rekordot, majd visszaadja a neki megfelelő, kimenetre írható sztringet!

- Paraméterlista
  - minion – a(z) Minion típusú rekord
- Visszaadott érték
  - az minion paraméternek megfelelő sztring reprezentáció

A minion_to_line() függvény

```
def minion_to_line(minion):
    return f"{minion.name} {minion.hunger}
           ({minion.size})"
```

A sort_minions() függvény

- Írj egy sort_minions() nevű függvényt, mely megkap egy Minion listát, majd rendezi és visszaadja azt!

- Paraméterlista
  - minions – a(z) Minion elemeket tartalmazó lista
- Visszaadott érték
  - a paraméterként megkapott minions lista a következő szempontok szerint rendezve:
    - a lelkesedés szerint csökkenő sorrend
    - a név szerint növekvő sorrend

A sort_minions() függvény

```
def sort_minions(minions):
    minions.sort(key = lambda minion:
    (-minion.motivation, minion.name))

    return minions
```

A főprogram

- Készíts egy olyan főprogramot, mely az alábbi specifikációk alapján teszi kipróbálhatóvá a megoldást!

- A standard bemenet sorai egy-egy Minion pontosvesszőkkel (;) elválasztott adatait tartalmazzák.
- A tesztesetek végét állományvégjel (EOF) jelzi.
- A beolvasott sorok feldolgozása a következő módon történik:
  - a(z) Minion lista előállítása a(z) line_to_minion() függvény felhasználásával
  - a(z) Minion lista rendezése a(z) sort_minions() függvénnyel
  - a(z) Minion lista kiírása a(z) minion_to_line() függvény felhasználásával

A főprogram – main()

```
def main():
      minions = []

      for line in sys.stdin:
            minions.append(line_to_minion(line))

      for minion in sort_minions(minions):
            print(minion_to_line(minion))
```

Bemenet/Kimenet

Bemenet:

- Bob;87;100;S
- Dave;43;10;L
- Stuart;50;30;L
- Jerry;40;20;XL

Kimenet:

- Bob 87 (S)
- Stuart 50 (L)
- Jerry 40 (XL)
- Dave 43 (L)

```python
import sys
from typing import NamedTuple

Minion = NamedTuple("Minion", [("name", str), ("hunger", int), ("motivation", int),
("size", str)])

def minion_to_line(minion):
    return f"{minion.name} {minion.hunger} ({minion.size})"

def line_to_minion(line):
    adat = line.strip().split(";")
    return Minion(adat[0], int(adat[1]), int(adat[2]), adat[3])

def sort_minions(minions):
    minions.sort(key=lambda minions: (-minions.motivation, minions.name))
    return minions

def main():
    minions = []
    for line in sys.stdin:
        minions.append(line_to_minion(line))

    for minion in sort_minions(minions):
        print(minion_to_line(minion))

if __name__ == '__main__':
    main()
```

Főprogram – Fájlból olvasás

https://viskillz.inf.unideb.hu/prog/#/?week=P1094&exercise=P109401e&page=sheet

Bemenet:
bemenet.txt – Parancssori Argumentumként

- Bob;87;100;S
- Dave;43;10;L
- Stuart;50;30;L
- Jerry;40;20;XL

Kimenet:

- Bob 87 (S)
- Stuart 50 (L)
- Jerry 40 (XL)
- Dave 43 (L)

Főprogram

```
def main():
    with open(sys.argv[1]) as file:
        minions=[]
        for line in file:
            minions.append(line_to_minion(line))

        for minion in sort_minions(minions):
            print(minion_to_line(minion))
if __name__ == '__main__':
    main()
```

Feladat – Hullámvasút

https://viskillz.inf.unideb.hu/prog/#/?week=P1083&exercise=P108302e&page=sheet

- Egy zsúfolt vidámparkban olykor több órát is sorban kell ahhoz állni, hogy felülhessünk egy-egy hullámvasútra. A parkok méretére és a kígyózó sorokra való tekintettel érdemes megtervezni, hogy milyen sorrendben ülünk fel a játékaikra.
- A PortAventura World Európa egyik legnagyobb vidámparkja, mely a Costa Braván, Salou és Tarragona között található. Klasszikus területe (a kontinens egyetlen Ferrari parkján kívül) több tematikus világot tartalmaz, amelyek mindegyikében található legalább egy hullámvasút.
- Ebben a feladatban hullámvasutak adatait kell beolvasnod a standard inputról, rendezni, majd kiírni a standard outputra.

Feladat – Hullámvasút

Formátumok:

- A standard bemeneten egy hullámvasút a következő formátumban szerepel:
- <hullámvasút_neve>;<világ_neve>;<legkisebb_magasság>;<várakozási_idő>
- A standard kimeneten egy hullámvasút a következő formátumban szerepel:
- <hullámvasút_neve> (<világ_nave>): <várkozási_idő>

A jelölések:

- hullámvasút_neve: egy legfeljebb 30 karakter hosszú, csak angol betűket és szóközöket tartalmazó egyedi sztring
- világ_neve: egy legfeljebb 30 karakter hosszú, csak angol betűket és szóközöket tartalmazó egyedi sztring
- legkisebb_magasság: egy nemnegatív egész érték
- várakozási_idő: egy nemnegatív egész érték
- A feladat megoldását több függvényre kell bontanod, ezek leírását egyenként, lentebb találod.

RollerCoaster típus

- Definiáld a **RollerCoaster** nevű nevesített tuple-t, mely a következő mezőkkel rendelkezik:

- name: a hullámvasút neve (típus: <class 'str'>)
- world: a világ neve (típus: <class 'str'>)
- height: a legkisebb magasság (típus: <class 'int'>)
- time: a várakozási idő (típus: <class 'int'>)

RollerCoaster típus

```
from typing import NamedTuple

RollerCoaster = NamedTuple("RollerCoaster",
[("name", str), ("world", str),
("height", int), ("time", int)])
```

A line_to_coaster() függvény

- Írj egy **line_to_coaster()** nevű függvényt, mely megkapja a bemenet egy sorát, majd visszaadja sorból kinyert adatokkal feltöltött, RollerCoaster nevesített tuple-t!

A line_to_coaster() függvény

```
def line_to_coaster(line):
    adat = line.strip().split(";")
    return RollerCoaster(adat[0], adat[1],
           int(adat[2]), int(adat[3]))
```

A coaster_to_line() függvény

- Írj egy **coaster_to_line** nevű függvényt, mely megkap egy RollerCoaster rekordot, majd visszaadja a neki megfelelő, kimenetre írható sztringet!

- Paraméterlista
  - coaster – a(z) RollerCoaster típusú rekord
- Visszaadott érték
  - az coaster paraméternek megfelelő sztring reprezentáció

A coaster_to_line() függvény

```
def coaster_to_line(coaster):

    return f"{coaster.name}
    ({coaster.world}): {coaster.time}"
```

A sort_coasters() függvény

- Írj egy **sort_coasters()** nevű függvényt, mely megkap egy RollerCoaster listát, majd rendezi és visszaadja azt!

- Paraméterlista
  - coasters – a(z) RollerCoaster elemeket tartalmazó lista
- Visszaadott érték
  - a paraméterként megkapott coasters lista a következő szempontok szerint rendezve:
    - a várakozási idő szerint növekvő sorrend
    - a legkisebb magasság szerint csökkenő sorrend
    - a hullámvasút neve szerint növekvő sorrend

A sort_coasters() függvény

```
def sort_coasters(coasters):

    coasters.sort(key=lambda
    coaster: (coaster.time, -coaster.height,
                coaster.name))

    return coasters
```

```python
import sys
from typing import NamedTuple

RollerCoaster = NamedTuple("RollerCoaster", [("name", str), ("world", str),
                           ("height", int), ("time", int)])

def line_to_coaster(line):
    tokens = line.strip().split(";")
    return RollerCoaster(tokens[0], tokens[1], int(tokens[2]), int(tokens[3]))

def coaster_to_line(coaster):
    return f"{coaster.name} ({coaster.world}): {coaster.time}"

def sort_coasters(coasters):
    coasters.sort(key = lambda coaster: (coaster.time, -coaster.height,
                  coaster.name))
    return coasters
def main():
    coasters = []
    for line in sys.stdin:
        coasters.append(line_to_coaster(line))

    for coaster in sort_coasters(coasters):
        print(coaster_to_line(coaster))
if __name__ == '__main__':
    main()
```

1. Házi Feladat

- Rekordok – fájlból olvasás, kigyűjtés
- Repülőterek (állományvégjelig olvasás)
  - https://viskillz.inf.unideb.hu/prog/#/?week=P1095&exercise=P109503e&page=sheet

2. Házi Feladat

- Rekordok – fájlból olvasás, megszámlálás
- Lego készletek #1 (n teszteset)
  - https://viskillz.inf.unideb.hu/prog/#/?week=P1096&exercise=P109604n&page=sheet

3. Házi Feladat

- Rekordok – fájlból csoportosítás
- Kuponok #1 (végjelig olvasás)
  - https://viskillz.inf.unideb.hu/prog/#/?week=P1097&exercise=P109706t&page=sheet