Bevezetés a programozásba

9. Gyakorlat

Ismétlés

- Rekurzív függvények

1. Házi Feladat

https://viskillz.inf.unideb.hu/prog/#/?week=P1032

- Pitagorasz tétel (állományvégjelig olvasás)
- Pitagorasz tétel (n teszteset)
- Pitagorasz tétel (végjelig olvasás)

Pitagorasz() függvény

```
import math

def Pitagorasz(a: float, b: float, d: int) -> float:
    return round(math.sqrt(a**2 + b**2), d)
```

Pitagorasz() függvény – EOF-ig

```
import math
import sys
def Pitagorasz(a: float, b: float, d: int) -> float:
    return round(math.sqrt(a**2 + b**2),d)

def main():
    for line in sys.stdin:
        numbers = line.split()
        a = float(numbers[0])
        b = float(numbers[1])
        d = int(numbers[2])
        print(Pitagorasz(a,b,d))

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

Pitagorasz() függvény – N teszteset

```
import math

def Pitagorasz(a: float, b: float, d: int) -> float:
    return round(math.sqrt(a ** 2 + b ** 2), d)

def main():
    n = int(input())
    for i in range(n):
        numbers = input().split()
        a = float(numbers[0])
        b = float(numbers[1])
        d = int(numbers[2])
        print(Pitagorasz(a, b, d))

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

Pitagorasz() függvény – Üres sorig

```
import math
import sys
def Pitagorasz(a: float, b: float, d: int) -> float:
    return round(math.sqrt(a ** 2 + b ** 2), d)

def main():
    while True:
        line = input()
        if line== "":
            break
        numbers = line.split()
        a = float(numbers[0])
        b = float(numbers[1])
        d = int(numbers[2])
        print(Pitagorasz(a, b, d))
if __name__ == '__main__':
    main()
```

2. Házi Feladat

https://viskillz.inf.unideb.hu/prog/#/?week=P1043

- Lista valós osztása (állományvégjelig olvasás)
- Lista valós osztása (n teszteset)
- Lista valós osztása (végjelig olvasás)

Megoldás – Lista valós osztása

```
def apply_float_division(numbers: list[int],
divisor: int) -> list[float]:
    result = []
    for number in numbers:
        result.append(round(number / divisor, 3))
    return result
```

3. Házi Feladat

https://viskillz.inf.unideb.hu/prog/#/?week=P1053

- Kisbetűk inkrementálása (állományvégjelig olvasás)
- Kisbetűk inkrementálása (n teszteset)
- Kisbetűk inkrementálása (végjelig olvasás)

3. Házi Feladat – Segítség

- chr() függvény
- ord() függvény

- Karakterkódok kezelése:

- ord("A") # 65 a kódja
- chr(65) # "A" betű

- chr(ord("A") + 3) # "D", mert A → B → C → D

Megoldás – Kisbetűk inkrementálása

```
def increment_lowers(original: str) -> str:
    return "".join([chr((ord(c) - ord("a") + 1) % 26 + ord("a"))
                    if c.islower() else c for c in original])
```

Halmazok

- A halmaz típus: **class set**
- A Python nyelv beépítve tartalmaz egy halmaz típust, a **set** osztályt.
- Ez támogatja a szokásos halmazműveleteket:
  - betehetünk egy elemet,
  - kivehetünk egy elemet,
  - képezhetjük a halmazok metszetét,
  - unióját,
  - különbségét.

Halmazok

```
s = set(["magyar", "angol", "német"])  # 3 elemű halmaz
s = {"magyar", "angol", "német"}
s = set()  # üres halmaz, vigyázat: {} mást jelent->szótár!

# lekérdezések: eleme-e, nincs benne
print("angol" in s)
print("japán" not in s)

# módosítók: betesz, kivesz
s.add("olasz")
s.remove("német")
```

Halmaz műveletek

- s1 & s2 metszet
- s1 | s2 unió
- s1 - s2 különbség
- s1 ^ s2 delta/szimmentrikus különbség
- Az & ÉS operátor a metszetet: olyan elemeket keres, amelyek benne vannak az első ÉS a második halmazban benne vannak.
- A | VAGY operátor az uniót, olyan elemeket keresve, amelyek az első VAGY a második halmazban szerepelnek.
- A - különbség, az első halmazból kiveszi a második halmazban megtalálható elemeket.
- A ^ (kalap, angolul caret) operátorral jelölt delta műveletet. Ez azokat az elemeket adja, amelyek csak az egyik, vagy csak a másik halmazban szerepelnek (mint az XOR művelet).

Példa

```
s1 = {"magyar", "angol", "német"}
s2 = set()
s2.add("olasz")
s2.add("német")
print(s1)
print(s2)
print("-----------------------------------")
# halmazműveletek: metszet, unió, különbség,
szimmentikus különbség (delta, xor)
print(s1 & s2)
print(s1 | s2)
print(s1 - s2)
print(s1 ^ s2)
```

Halmaz műveletek

Set műveletek	Matematikai megfelelője
Union (|)	Egyesítés, unió
Intersection (&)	Metszet
Difference (-)	Különbség
SymmetricDifference (^)	Szimmetrikus különbség

```
s1 = {"magyar", "angol", "német"}
s2 = set()
s2.add("olasz")
s2.add("német")
print(s1)
print(s2)
print("----------------------------------------")
# halmazműveletek: metszet, unió, különbség,
szimmentikus különbség (delta, xor)
print(s1.intersection(s2))
print(s1 & s2)
print(s1.union(s2))
print(s1 | s2)
print(s1.difference(s2))
print(s1 - s2)
print(s1.symmetric_difference(s2))
print(s1 ^ s2)
```

Halmaz műveletek

- **len(s)**: s halmaz hosszát adja vissza
- x **in** s: igazzal tér vissza, ha x eleme s halmaznak
- x **not in** s: igazzal tér vissza, ha x nem eleme s halmaznak
- **s1.isdisjoint(s2)**: igazat ad, ha s1 és s2 metszete üres halmaz (diszjunkt halmazok)
- **s1.issubset(s2)**:
  - s1 <= s2: igazzal ad, ha s1 részhalmaza az s2 halmaznak
  - s1 < s2: igazat ad, ha s1 valódi részhalmaza s2-nek

Halmaz műveletek

- **s.add(elem)**: elem-et beteszi s halmazba
- **s.remove(elem)**: elem-et kiveszi a halmazból. KeyError kivétel keletkezik, ha elem nem található s-ben
- **s.discard(elem)**: elem-et kiveszi a halmazból, ha az benne van
- **s.pop()**: kivesz és visszaad egy tetszőleges elemet a halmazból
- **s.clear()**: minden elemet töröl a halmazból
- **s.copy()**: Lemásolja s-et, és visszatér az új objektummal

Feladatok

- Töltsünk fel két halmazt 1-10 közötti random számokkal.
- A halmazok méretét olvassuk be a standard inputról.
- Írassuk ki a két halmaz unióját, metszetét, különbségét és szimmetrikus különbségét.

```
import random
s1 = set()
s2 = set()
n=int(input("n = "))
m=int(input("m = "))
while len(s1)<n:
    s1.add(random.randint(1,10))
while len(s2)<m:
    s2.add(random.randint(1,10))
print(s1)
print(s2)
print("Uniója =", s1.union(s2))
print("Metszete =", s1.intersection(s2))
print("Különbsége =", s1.difference(s2))
print("Szimmetrikus különbsége =",
s1.symmetric_difference(s2))
```

Feladatok

- Töltsünk fel két halmazt 1-20 közötti véletlenszerűen generált páros illetve páratlan számokkal, majd a képezzük a halmazok unióját.
- A halmazok méretét olvassuk be a standard inputról.

Megoldás

```
import random
paros = set()
paratlan = set()
n = int(input("n = "))
m = int(input("m = "))
while len(paros) < n:
    szam=random.randint(1, 20)
    if szam%2==0:
        paros.add(szam)
while len(paratlan) < m:
    szam = random.randint(1, 20)
    if szam % 2 == 1:
        paratlan.add(szam)
print(paros)
print(paratlan)
print("Uniója =", paros.union(paratlan))
```

Szótár

- Pythonban egy szótár, azaz **dict** típusú tárolót legegyszerűbben úgy hozhatunk létre, hogy kapcsos zárójelek között felsoroljuk a kulcs–érték párokat, közöttük kettősponttal. Ez a leggyakoribb forma.
- A zárójelpár akár lehet üres is, és olyankor üres szótár jön létre, amelybe később tehetünk majd elemeket.
- Megjegyzés: Azért nem lehet üres halmazt, azaz set típusú tárolót kapcsos zárójelekkel létrehozni, mivel mikor már van legalább egy elem, egyértelművé válik, melyikről van szó: ha kettőspontok is vannak, akkor szótárról, ha nincsenek, akkor halmazról.

Szótár

```
bevasarlolista = {
    "alma": 2,
    "körte": 1,
    "barack": 5,
}
print(bevasarlolista)
```

Eredmény:

- {'alma': 2, 'körte': 1, 'barack': 5}

Szótár

- Üres szótárat a **dict()** konstruktorhívással, vagy egyszerűen egy üres kapcsos zárójelpárral tudunk létrehozni: {}.
- Az egyes elemeknek a hozzáadása és elérése az indexelő operátorral történik.

```
bevasarlolista = dict()      # vagy röviden: {}
bevasarlolista["alma"] = 2
bevasarlolista["körte"] = 1
bevasarlolista["barack"] = 5
print(bevasarlolista)
```

Szótár

- Harmadik lehetőségünk, hogy megadunk egy listát, amelynek elemei a szótár tartalmát adják. Ebben az esetben a lista minden eleme egy ***tuple*** kell legyen, amelyek a kulcs-érték párokat tartalmazzák.

```
bevasarlolista = dict([
    ("alma", 2),
    ("körte", 1),
    ("barack", 5),
])
print(bevasarlolista)
```

A szótár elemeinek elérése

```
print(bevasarlolista)
 #{'alma': 2, 'körte': 1, 'barack': 5}
print(bevasarlolista["alma"])  # 2
print("alma" in bevasarlolista)  # True
del bevasarlolista["alma"]
bevasarlolista["szilva"] = 7
print(bevasarlolista)
#{'körte': 1, 'barack': 5, 'szilva': 7}
try:
    print(bevasarlolista["papaja"])
except KeyError as e:
    print("Nincs benne")  # Nincs benne
```

Szótár elemeinek bejárása

```
for kulcs in bevasarlolista:  #kulcsok
    ertek = bevasarlolista[kulcs]
    print(kulcs, ertek)

for ertek in bevasarlolista.values():  #értékek
    print(ertek)
print("Összesen:", sum(bevasarlolista.values()))

for kulcs, ertek in bevasarlolista.items():
    #tuple-k
    print(f"{kulcs}: {ertek} kg")
```

Szótár elemeinek bejárása

- Ha csak az értékekre vagyunk kíváncsiak, a kulcsok lényegtelenek, akkor a **.values()** függvényhívást használhatjuk. Ez egy listához hasonlóan viselkedő tárolóban visszaadja csak az értékeket.

- A középső példa **sum()** függvényhívása így összegzi, hány kg gyümölcsöt vásárolunk összesen.

- Legkényelmesebb a dict bejárásakor úgy dolgozni, hogy a kulcsokat és az értékeket is látjuk a ciklusban. Ez az **.items()** függvényhívással érhető el.
- Ekkor az iterálás során ***tuple*** típusú elemeket kapunk, amelyek (kulcs, érték) formában megadják a szótár elemeit.

Feladat

- Írjunk programot, mely szavakat olvas az üres sztringig, és készítsünk a beolvasott szavakról egy előfordulási statisztikát, tehát melyik szó hányszor szerepelt a beolvasás során.

Megoldás

```
elofordulas = {}

szo = input()
while szo != "":
    if szo in elofordulas:
        elofordulas[szo] += 1
    else:
        elofordulas[szo] = 1
    szo = input()

for szo, db in elofordulas.items():
    print(f"{szo}: {db} db")
```

Majomszokás

- https://progcont.hu/progcont/100356/?pid=201536

Megoldás

```
import sys
majmok={}
for line in sys.stdin:
    adat = line.split(";")
    if adat[0] in majmok:
        majmok[adat[0]] += int(adat[1])
    else:
        majmok[adat[0]]=int(adat[1])

for i in sorted(majmok):
    print ("{0}: {1}".format(i, majmok[i]))
```

Fesztiválszezon

- https://progcont.hu/progcont/100357/?pid=201540

Megoldás

```
import sys
egyuttesek = {}
for line in sys.stdin:
    adat = line.split(":")
    nevek = adat[1].strip().split(",")
    for i in range(0,len(nevek)):
        if nevek[i] in egyuttesek:
            egyuttesek[nevek[i]] += 1
        else:
            egyuttesek[nevek[i]] = 1
for i in sorted(egyuttesek):
    print ("{0}: {1}".format(i, egyuttesek[i]))
```

Funkcionális programozás

- A funkcionális programozásban a program csak olyan függvényeket tartalmaz, amelyek nem módosítják a hívó környezetüket.
- Tehát a függvények kimenete csak a bemenetüktől függ, és látható módon nem befolyásolják azt a helyet, ahonnan meghívták őket.

- Vannak olyan programozási nyelvek, amelyek tisztán a funkcionális programozással dolgoznak, például a Haskell, az Erlang vagy a Lisp.
- A Python támogatja a funkcionális programozást, de tartalmaz elemeket egyéb programozási paradigmákból is.

Funkcionális programozás

- A Pythonban lehetséges, hogy egy függvénynek függvény bemeneti paramétere legyen, sőt függvény kimenetet is képes produkálni.

```
def fuggveny(a, b):
    return a + b

def masik_fuggveny(valamilyen_fuggveny_parameter, x, y):
    return valamilyen_fuggveny_parameter(x, y)

print(masik_fuggveny(fuggveny, 1, 2))
3
```

Funkcionális programozás

- Átadhatunk tehát egy függvénynek egy másik függvényt bemenetként, amely módosítja a hatását. Erre talán az egyik legjobb példa a sort függvény, amellyel sorrendbe tudjuk rakni egy lista elemeit.

```
print(lista)
print(sorted(lista))  # alapértelmezetten ABC sorrendbe rakjuk az elemeket
print(sorted(lista, key=len))  # a sorrendbe rakás kulcsa a len függvény
lesz
# Ez azt jelenti, hogy a lista minden elemén meghívjuk a len függvényt, és
az eredmény alapján rakjuk sorrendbe az elemeket.
print(sorted(lista, key=len, reverse=True))  #ugyanaz fordított sorrendben

# Készítsünk egy saját függvényt, a len helyett, amely kimenete a szavak
negatív hossza lesz.
def forditott_sorrend(elem):
    return -len(elem)
print(sorted(lista, key=forditott_sorrend))
```

Lambda (névtelen) függvények

- Vannak olyan esetek, amikor szükségtelen megadnunk egy nevet a létrehozott függvényünknek, mert például csak egy rendezéshez használjuk, mint a fentiekben.
- Emellett, ha egyszerűbb függvényekről van szó, akkor akár lambda függvényekkel is létre tudjuk hozni őket.

- A lambda függvények egyszerűsített függvény definíciók. Szerkezetük a következő:
- **lambda <paraméterlista>: <képlet, kifejezés>**

```
print((lambda x, y, z: (x + y + z) * 10)(1, 2, 3))
60

lista = ["Jabba", "Han", "Luke"]
print(sorted(lista, key=lambda elem: -len(elem)))
['Jabba', 'Luke', 'Han']
```

Klubhűség

- https://progcont.hu/progcont/100360/?pid=201553

```
import sys
n = int(input())
sportolok = {}
for line in sys.stdin:
    adat = line.split(":")
    nevek = adat[1].strip().split(",")
    for i in range(len(nevek)):
        if nevek[i] in sportolok:
            sportolok[nevek[i]] += 1
        else:
            sportolok[nevek[i]] = 1

for kulcs, ertek in sorted(sportolok.items(),
            key=lambda rend: (-rend[1], rend[0])):
    if ertek > n:
        print("{0}: {1}".format(kulcs, ertek))
```

1. Házi Feladat

Mindennap egy alma az orvost távol tartja

- https://progcont.hu/progcont/100358/?pid=201544

2. Házi Feladat

Új mintatanterv

- https://progcont.hu/progcont/100278/?pid=201288

3. Házi Feladat

Erdei lemmingek

- https://progcont.hu/progcont/100337/?pid=201476