el?z? fel következ?

Tartalomjegyzék

- 1 Feladatok
  - 1.1 Alakzatok a vásznon
    - 1.1.1 Alakzatok
    - 1.1.2 Örökl?dés és konstruktorok
    - 1.1.3 Vászon
  - 1.2 Overload
  - 1.3 Iterálható

Feladatok

Nyissuk meg a Sypdert.

Alakzatok a vásznon

Alakzatok

Írjunk egy **Shape** osztályt.

- Legyen **x** és **y** változója, ezek tárolják az alakzat pozícióját a síkon.
- Legyen egy **move** metódusa, aminek egyetlen **v** paramétere van, egy kételem? lista, a vektor, amivel el kell mozgatni az alakzatot.

Definiáljuk a **Shape** osztály leszármazottaiként az

- **Ellipse** ellipszis, legyen meg a kis- és nagytengelye (**a,b**)
- **Rectangle** téglalap, legyen meg az oldalak hossza (**a,b**)

osztályokat. Mindkét esetben a pozíciójuk a súlypontjukat jelentse.

Írjunk mindkét osztályhoz egy **area** függvényt, ami kiszámítja az alakzat területét!

Definiáljuk az **Ellipse** osztály **equation** metódusát, ami kiírja az adott ellipszis egyenletét!

Örökl?dés és konstruktorok

Ha a *leszármazott osztályban* (például Ellipse) az *?sosztály konstruktorát* (jelen esetben Shape) akarjuk hívni, akkor erre két módszer is lehetséges:

```
class B(A):
    def __init__(self, x, y, a, b):
         A.__init__(self, x, y)
         # vagy
         super(B, self).__init__(x, y)
```

Az els? változatban

```
A.__init__(self, x, y)
```

megmondjuk, hogy az **A** osztály konstruktorát akarjuk meghívni (amit már korábban megírtunk).
A második változatban

```
super(B, self).__init__(x, y)
```

azt mondjuk, hogy a **B** mindenkori ?sosztályát hívjuk meg, ami most éppen **A**.

Vászon

Definiáljuk a **Canvas** (vászon) osztályt.

- Egyetlen tagváltozója legyen a **shapes**, ami alakzatok listáját tárolja.
- Definiáljuk egy **add** metódust, amivel újabb **Shape**-et adunk a vászonhoz!
- Oldjuk meg, hogy az osztályunk iterálható legyen! Ehhez definiáljuk az **__iter__(self)** metódust, valamit a **next(self)** metódust, ahogy az el?adáson láttuk.
- Definiáljuk a **crop** metódust a következ?képp: a bemeneti paraméter két pont koordinátája, ezek egy téglalap bal fels? és jobb alsó pontjai. A függvény térjen vissza azon alakzatok listájával, amelyek a vásznon vannak és teljesen belefférnek az így definiált téglalapba. Ehhez a feladathoz az **Ellipse** és a **Rectangle** osztálynak is szüksége lesz egy **box()** metódusra, ami a legkisebb tartalmazó doboz bal fels? és jobb alsó sarkait adja vissza.

Overload

1. Írjunk egy függvényt, aminek az els? argumentuma **n**, egy **int** típusú változó. A függvény térjen vissza **True**-val, ha annyi extra paraméterrel hívták meg, mint az els? bemeneti paraméter értéke, egyébként térjen vissza **False**-szal.
2. Definiáljunk egy **szumma** függvényt, ami tetsz?legesen sok bemeneti paraméterének összegével tér vissza!
    1. Kezeljük le a kivételt, ha a paraméterek típusa nem azonos!
3. Definiáljunk egy **print_words** függvényt, úgy, hogy a megadott (akármennyi) szavakat annyiszor írja ki, amennyit megadunk bemenetnként (szavanként)!
    1. Kezejük le kivételként, ha a bemeneten nem egész számot adtak meg a szó gyakoriságára!

Iterálható

Íjunk olyan iterálható osztályt, mint a **range**, de ne egy listát járjon be, hanem csak az aktuális elemet tárolja.

```
class Range(object):
    def __init__( ... ):
        ...
    def __iter__( ... ):
        ...
    def next( ... ):
        ...
```

- konstruktora egy számot vagy sztringet kapjon. Addig a számig lehessen iterálni rajta, nullától, egyesével.
- Ha a szám nem pozitív, akkor 0 hosszan lehessen iterálni rajta.
- Ha sztringet kap a konstruktor és az nem értelmezhet? egészként, akkor emeljünk **ValueError**

kivételt.
        - ♦ Ha értelmezhet? egészként, akkor alakítsuk át egésszé és számoljunk azzal.
- Ha **"http://wiki.math.bme.huinf"http://wiki.math.bme.hu** sztringet kap a konstruktor, akkor végtelen sokáig lehessen rajta iterálni!

el?z? fel következ?