Ismétlés

- Osztályok:

```
class Test:
    def fv(self, n): # self mindig az elsõ paramétere a metódusoknak
        self.v = n   # ha itt csak azt írnám, hogy v = n az nem jó, mert a v egy lokális változó l

t = Test()           # Test típusú objektum létrehozása
t.fv(4)              # meghívjuk a fv metódusát t-nek
print t.v            # ekkor a v adattagja 4 lesz
print t.__dict__     # megnézhetjük az adattagokat
```

- Hibakezelés:

```
l = [1, 2, 3]
try:
    print l[1]
except:
    print "ide nem jutunk, mer nem lesz hiba"


try:
    print l[5]
except:
    print "ide jutottunk, de nem lett futas kozben hiba"
```

- Mátrix osztály:

Elkezdtünk írni egy mátrix osztályt, jelenleg elég kezdetleges, ezt fogjuk folytatni a mai gyakorlaton.

Feladat

Mátrix folytatása

Innnen letölthetitek az eddigi osztályt, a következõ sorokban leírt metódusok leírásának fejeivel kiegészítve.

- **__init__(self, n = 5)**: az __init__ metódus akkor fut le amikor létrehozunk egy objektumot (tehát pl az ismétlésben amikor azt írjuk t = Test()), ezzel tudjuk egy kezdõállapotba állítani az objektumunkat. Itt most az lenne a feladat, hogy a kapott **n** számnak megfelelõen egy n x n-es csupa 0 mátrix legyen a létrehozott mátrixunk. Mint korábban most is az **A** adattagjába tároljuk a mátrixot (listák listája). Most még pluszban vegyünk fel egy **n** adattagot is, amiben a mátrix dimenzióját tároljuk. A függvénydefiniálásban az **n = 5** azt jelenti, hogy ha nem adjuk meg ezt a paramétert, akkor automatikusan 5-nek veszi, tehát írhatjuk ezeket:

```
m1 = Matrix(2)   # 2 x 2-es csupa 0 mátrixot készít
m2 = Matrix()    # 5 x 5-ös csupa 0 mátrixot készít
```

- **__add__(self, jobboldal)**: az __add__ függvény akkor hívódik meg, ha egy ilyen objektummal összeadunk, tehát ha **m1** és **m2** is Matrix típusú, akkor **m1 + m2** utasításnál fut le. Méghozzá úgy, hogy **m1**-et veszi **self**-nek, az **m2**-t pedig a második paraméternek, szóval **jobboldal**-nak ebben az esetben. A függvénynek tehát egy Matrix típusú objektumot kell visszaadnia. (Ezt a függvényt megírtam, hogy példaként szolgáljon a továbbiakhoz.)

- **__sub__(self, jobboldal)**: az összeadás párja, akkor hívódik meg ha kivonást végzünk Matrix-al, nagyon hasonló az elõzõhöz, egy dolgot kell csak változtatni benne.

- **identity(self, n)**: ha meghívjuk egy már létezõ Matrix objektumon, akkor csináljon belõle n x n-es identitás mátrixot (fõátlóban egyesek, minden más 0).

- **__mul__(self, jobboldal)**: az __add__-hoz hasonló, ez a szorzáskor hívódik, mátrixszorzást kellene csinálnunk, kicsit (1 ciklussal) bonyolultabb mint az összeadás, de annak a mintájára lehet csinálni.

- **setMatrix(self, M)**: a bemenete **M** listák listája, írja át a mátrixot úgy, hogy ez a bemeneti **M** legyen.

- **__repr__(self)**: ez a függvény akkor hívódik meg, ha kiiratjuk az objektumot, szóval ha **print m1**-et írunk. A függvénynek egy str (string)-et kell visszaadnia, tehát az üres "http://wiki.math.bme.hu"http://wiki.math.bme.hu string-bõl kellene felépíteni a mátrix string-jét. Ehhez egy hasznos tudnivaló:

```
a = 15
print str(a) + str(24) # 1524-et ír ki
```

- **__len__(self)**: akkor hívódik meg, ha a **len** beépített függvényt használjuk egy objektumra, tehát pl: **len(m1)**-kor. Adja vissza a mátrix dimenzióját.

- **transpose(self)**: transzponálja a mátrixot.
- **__pow__(self, jobboldal)**: akkor hívódik meg amikor a hatvány operátort használjuk (**). Adja vissza a mátrix **jobboldal**-adik hatványát (önmagával vett szorzatát).

- **random(self, n, m)**: randomizálja a mátrix elemeit, méghozzá a **random.randint(min, max)** függvény segítségével, mely egy véletlen egész számot ad a [min, max] intervallumon. Legye n a min és m a max.

- **subMatrix(self, kihagyott)**: Visszaadja a mátrix azon részmátrixát amit az elsõ sor és a **kihagyott**-adik oszlop elhagyásával kapunk. Tehát egy mátrixot kell visszaadni, fontos hogy ne azt a mátrixot módosítsa amin meghívtuk, hanem egy új mátrixot adjon vissza. (A determináns számításnál lesz jelentõsége.)

- **det(self)**: adja vissza a mátrix determinánsát, bármilyen módszer használható, szerintem a legkönnyebben megvalósítható módszer (signum függvényt utálom) ez. Azaz vegyük a mátrix determinánsának 1. sora szerinti kifejtését és ezt alkalmazzuk rekurzívan. Ha az 1 x 1-es mátrixnál vagyunk értelemszerûen adjuk vissza az egyetlen elemet ami benne van. Egy példa a teszteléshez:

```
m3 = Matrix(3)
m3.setMatrix([[-2, 2, -3],[-1, 1, 3],[2, 0, -1]])
print m3
print m3.det()  # 18-at kell adnia
```