Tartalomjegyzék

- 1 Objektumok definiálása
  - 1.1 Tuple, szótár használata
  - 1.2 Python class
    - 1.2.1 Osztály definiálás
    - 1.2.2 Metódusok
    - 1.2.3 Objektum-orientált paradigma
    - 1.2.4 Speciális metódusok

Objektumok definiálása

Tuple, szótár használata

A mai óra folyamán ezzel a példakóddal fogunk foglalkozni. Ez lehetne pl. egy program része amit egy hallgató írt hogy segítsen összeállítani az órarendjét:

link

(Megjegyzés: Az objektumorientált programozás, a kivételekkel való hibakezeléshez hasonlóan, olyan technológia, amelynek az el?nyei igazán csak nagyobb programok írásánál jönnek el?. Ilyen kis példánál el kell túlozni a problémák nagyságát ahhoz hogy megindokoljuk ezen technikák használatát, de attól még hogy itt esetleg a megoldás bonyolultabbnak t?nik mint a probléma, nagyobb programoknál ugyanez a megoldás már jobb arányt produkál.)

Nézzük csak, mi történik, ha szeretném külön számontartani, hogy az óraszámból mennyi olyan, ahol jelenlétet ellen?riznek. Az lenne a logikus, ha a tuple-ben azössz óraszám mellett lenne. Ezért más kódot is módosítani kell, míg végül valami ilyesmire jutunk:

link

A fontos dolog az, hogy megváltoztattam hogy mit tárolok el egy-egy tárgyról, és emiatt meg kellett változtatnom az *osszkredit* függvényt, pedig a krediteket így is, úgy is eltároltam. Ebb?l látható, hogy az ilyen tuple-ös (vagy listás) megoldás nem fenntartható, ha valamit változtatni akarok a tárolási módszeren, akkor annak következtében mindenhol a kódban változtatnom kell, ahol ezeket az adatokat használom.

Egyik alternatíva szótárban tárolni a dolgokat. Ekkor a lista minden eleme egy szótár, ami pontosan ugyanazokat a kulcsokat tartalmazza:

link

Ez nem egy rossz megoldás, így már be lehet rakni új tulajdonságokat nagyobb probléma nélkül. Azért még van vele egy-két probléma:

- Mindegyik tantárgy létrehozásánál le kell írni az adatmez?k neveit (azt hogy "http://wiki.math.bme.hunev"http://wiki.math.bme.hu, "http://wiki.math.bme.huoraszam"http://wiki.math.bme.hu, stb.).
- Ha a kódban több hely is van, ahol létrehozok ilyen tantárgyakat, akkor ha bevezetek ilyen új adatot, akkor mindenhol módosítani kéne, és lehet hogy csak kés?bb derül ki ha valahol elfelejtettük módosítani.
- Minden függvénynél ami ilyen formátumban tárolt tantárgyakat vár, dokumentálni kell, hogy ez pontosan mit jelent.

Ezeket még kb. meg lehet oldani ha bevezetünk egy *ujtargy()* nev? függvényt, és mindig azt használjuk ha tantárgyat akarunk létrehozni a kódban:

link

Ez a dokumentáláson is segít, írhatja azt pl. az *osszora()* függvény dokumentációja hogy "http://wiki.math.bme.huA *minden* paraméter tantárgyak adatait tartalmazó szótárakat tartalmaz, melyek az *ujtargy()* függvénnyel lettek létrehozva. Azt is megoldja, hogy ha a kódban elfelejtjük mindenhol betenni a plusz paramétereket, akkor már egyb?l az objektum létrehozásakor szólni fog a python hogy van hiányzó paraméter, nem csak kés?bb, használat közben derül ki esetleg a hiba.

Így már elég közel járunk egy tényleges osztály koncepcióhoz, csak még a python erre kitalált szintaxisát nem használtuk. Térjünk rá arra:

Python class

Osztály definiálás

Python-ban saját osztályt a *class* és az *object* kulcsszavak használatával tudunk létrehozni. A legegyszer?bb példa így nézne ki:

```
class Targy(object):
    pass
```

(Szokás az osztályok neveit mind nagybet?snek venni, hogy könnyebben megkülönböztethet? legyen. A python dokumentáció is ajánlja ezt mint egy lehet?séget, bár a python beépített osztályok nem követik ezt a módszert.) Az "http://wiki.math.bme.huobject"http://wiki.math.bme.hu helyére kerülhetne más is elvileg, de hogy az mit jelent, oda ebb?l a tárgyból már nem jutunk el, nekünk ott mindig az lesz hogy "http://wiki.math.bme.huobject"http://wiki.math.bme.hu.

Ezen az osztályon belül lehet aztán különböz? függvényeket definiálni, ezek lesznek az osztály metódusai. Néhány függvény név speciális, ilyen az *__init__()*, ami az osztály konstruktora. Minden függvénynek amit az osztályon belül definiálunk, az els? paramétere speciális lesz, ezt általában *"http://wiki.math.bme.huself"http://wiki.math.bme.hu*-nek szokták elnevezni. Nézzük a korábbi példát most már osztállyal megvalósítva.

link

Láthatjuk hogy a fenti két függvényben már hozzáférhetünk a tagváltozóhoz úgy hogy *targy.nev*, és ilyesmi, ami valamivel szebb mint a targy["http://wiki.math.bme.hunev"http://wiki.math.bme.hu], mert jelzi hogy a lehetséges nevek azok az objektum fix tagváltozói lehetnek. Amik a *targy __init__*-jének a paraméterei a self-en kívül, azokat kell megadni létrehozáskor, mást nem fogad el a program. (Legalábbis más darabszám paramétert. Azt elfogadja ha hülyeséget adunk meg.)

Metódusok

Más metódusokat is lehet definiálni az osztályban. Ezeknél is, amikor definiáljuk ?ket, az els? paraméter legyen a *self*. A metóduson belül a *self* változón keresztül férhetünk hozzá az aktuális objektum tagváltozóihoz. Példa:

link

És persze módosíthatjuk is az objektum tartalmát a metódusokból:

```
class Targy(object):
    def __init__(self, nev, oraszam, jelenlet, kredit):
        self.nev = nev
        self.oraszam = oraszam
        self.jelenlet = jelenlet
        self.kredit = kredit

    def valtozott_kreditszam(self, uj_kredit):
        self.kredit = uj_kredit
```

Objektum-orientált paradigma

Ez a megoldás így, eddig, nem tud semmivel se többet mint a szótáras megoldás. (Kés?bbi dolgokra ez nem lesz teljesen igaz.) De most tegyünk egy kis kitér?t az objektum-orientált programozásról.

Az objektum-orientált programozás egy programozási paradigma, tehát egy módszer a nagy programok szervezésére. A paradigma használata független a szintaxistól. Ha, az óra elején mutatott módon, szótárakkal tárolnánk a különböz? objektumok tagváltozóit, akkor is követhetjük az objektum-orientált paradigmát. Csak az kell, hogy szigorúan kövessük a szabályokat: csak az *ujtargy()* függvénnyel hozzunk létre tantárgy "http://wiki.math.bme.huobjektumokat"http://wiki.math.bme.hu, legyenek "http://wiki.math.bme.humetódusaink"http://wiki.math.bme.hu, amik els? paraméternek egy ilyen tantárgy "http://wiki.math.bme.huobjektumot"http://wiki.math.bme.hu (szótárat) kapnak, és azzal dolgoznak. (Ezeknek a függvényeknek a hívásakor így nekünk kell kézzel átadni a célobjektumot.)

Azonban a speciális szintaxis használata kicsit egyszer?bbé teszi a szabályok követését, és jelzi más programozóknak hogy itt objektum-orientáltan programozunk. (Vannak programozási nyelvek, pl. a C, amikben nincs ehhez megfelel? szintaxis, és ezért az el?bb említett trükköket használják amikor objektum-orientált programot írnak.)

Tehát különböz? programozási nyelvek különböz? mértékben segítik, vagy követelik meg az objektum-orientált programozás használatát. Pl. a Java és C# nyelv megköveteli, nem enged olyan kódot írni ami nincs objektumon belül. Azt a kódot is, amit a program indításakor akarunk hogy lefusson, valamelyik osztály *main()* nev? metódusába kell tenni. Ennek megfelel? valahogy így nézne ki pythonban:

link

A python valahol félúton van: van szintaxis az objektum-orientált programozáshoz, bár olyan szigorúságot nehéz elérni mint más objektum-orientált nyelveken. Nem követeli meg az objektum orientált gondolkozást, így lehet akár olyan egyszer? programot is írni hogy:

link

és ez egy teljes és m?köd? python program. Azonban a python-ban is kb. minden egy "http://wiki.math.bme.huobjektum"http://wiki.math.bme.hu, a beépített listák, karakterláncok, szótárak is, a

saját metódusaikkal és tagváltozóikkal. Még a modulok is objektumokként vannak kezelve miután be lettek töltve. Így a python a kódolási stílusban nem feltétlenül objektum-orientált, de megvalósításban az.

Speciális metódusok

Mint láttuk korábban, az *__init__()* metódusnak speciális szerepe van, ez a konstruktor. Vannak még más ilyen speciális nevek, amiknek speciális szerepe van az objektumban. Egyik ilyen az *__str__()*, ami azt mondja meg hogy hogyan lehet az objektumot karakterlánccá konvertálni, hogyan lehet print-elni. Például így lehetne ezt a *Targy* osztályunknál megvalósítani:

link

Általánosságban a két alulvonással kezd?d? nevek le vannak foglalva, hogy a python nyelv beépített dolgaira lehet csak ?ket használni, úgyhogy ti ne nevezzetek el úgy magatoknak dolgokat! (Bár gondolom nem állt szándékotokban.)

A python hivatalos dokumentációban itt megtalálható az összes ilyen speciális név. Számunkra még esetleg érdekes lehet:

- a *__len__()*, ami megmondja, hogy a *len(targy)* mit adjon vissza
- a *__cmp__()*, ami megmondja hogy hogyan kell két ilyen objektumot összehasonlítani, tehát azt, hogy melyik számítson kisebbnek/nagyobbnak sorbarendezéskor
- az *__add__()*, ami megmondja hogy annak hogy *targy1 + targy2* mi legyen az eredménye. (Tantárgyak esetén ennek persze nincs értelme.)