Tartalomjegyzék

Referenciák, objektumok

Alapvet? fogalmak

Emlékezzünk vissza két el?adással korábban amit a mutable típusokról mondtunk. Hogy a listáknál/szótáraknál el?fordulhat pl. hogy két változó ugyanarra a listára vonatkozik, ha módosítom az egyiket, a másik is módosul.

link

Ezt értelmezhetjük úgy, hogy van egy lista, ami létezik valahol a számítógép memóriájában, és több változó is ugyanara a listára hivatkozik. Bevezetünk néhány szakkifejezést, ami segít ezeket a fogalmakat kezelni.

Egy ilyen dolog, ami a memóriában valahol létezik, egy **objektum**. Az objektumnak van egy **típusa**, vagy más szóval **osztálya**, ami megmondja, hogy az az objektum éppen lista, szótár, vagy valami más. A konkrét változók amik erre az objektumra hivatkoznak, azok **referenciák**, és úgy mondhatjuk, hogy erre az objektumra **mutatnak**.

Az objektumnak a legfontosabb, alapvet?, tulajdonsága az osztálya. Ez mondja meg, hogy milyen dolgokat tudunk vele tenni. Például, hogy a listának ki tudjuk venni a valahányadik elemét a szögletes zárójellel, vagy hogy a szótárnak van egy *has_key()* nev? metódusa, amivel meg lehet nézni hogy egy bizonyos kulcs szerepel-e benne. Hasonlóan egy modulhoz, egy osztály is alapvet?en kétfajta elemet tartalmazhat: változókat, ezeket hívhatjuk az osztály **tagváltozóinak**, és függvényeket, ezeket hívhatjuk az osztály **metódusainak**.

Nézzünk egy példát:

link

Ebben az esetben létrehoztam a *datetime.date* osztály egy példányát, egy objektumot, ami el tud tárolni egy dátumot, ebben az esetben a mai dátumot. A *ma* nev? változó egy referencia, ami erre az objektumra mutat. Megtudható a *datetime.date* osztály dokumentációjából, hogy ennek az osztálynak van pl. egy *year* nev? tagváltozója, és egy *weekday()* nev? metódusa, így ezek elérhet?ek a referencián keresztül.

Az osztályoknak egy speciális metódusa a **konstruktor**a. Amikor létrehozom az osztály egy példányát, akkor ezt a metódust hívom meg, tehát az számít, hogy ennek a metódusnak milyen paraméterei vannak. Ennek ellenére csak az osztály nevét kell leírni, és aztán a paramétereket a **példányosítás**hoz. Pl. a *datetime.date* osztály konstruktorának 3 paramétere az év, hónap és nap, ilyen sorrendben. (Kivéve azt a néhány osztályt aminek speciális szintakszisa van a létrehozáshoz, mint a lista meg a szótár.)

Itt mindjárt láthatunk is egy fontos különbséget a modulokhoz képest. A modulnál a bele tartozó függvények meghívásához csak a modulra volt szükségem, azt mondhattam a *math* modulnál hogy *math.sin(1)*. Azonban a *datetime.date* osztályra nem mondhatom azt hogy *datetime.date.weekday()*, szükségem van az objektum egy példányára, hogy a metódust azon tudjam meghívni.

Vannak olyan függvények is, amiket magán az osztályon lehet hívni, és nem egy példányon. Pythonban ezeket *"http://wiki.math.bme.huclass method"http://wiki.math.bme.hu*-nek és *"http://wiki.math.bme.huclass attribute"http://wiki.math.bme.hu*-nak hívják, ellenben azokkal amiket az el?bb említettem, amiket pontosabban *"http://wiki.math.bme.huinstance method"http://wiki.math.bme.hu*-nek és *"http://wiki.math.bme.huinstance attribute"http://wiki.math.bme.hu*-nak hívnak. Ezeket az "http://wiki.math.bme.huosztálymetódus"http://wiki.math.bme.hu-okat akkor használják, ha valamely függvény/változó szervesen kapcsolódik az adott osztályhoz, de nem egy konkrét példányhoz. Lényegében úgy érhet?ek el, mint ha lenne az osztály nevével egy azonos nev? modul is, és abban lennének. Itt példa a *datetime.date.min*, ami a lehetséges legkisebb dátum amit a *datetime.date* osztály tárolni tud: ez az érték az osztály minden példányára ugyanaz, és szükségünk lehet már akkor rá amikor még nem hoztunk létre egy példányt se, ezért az osztályhoz tartozik, nem az egyes objektumokhoz. De ez csak plusz lehet?ség, továbbra is elérhet? az objektumokon keresztül is, pl. a korábbi példában a *ma.min* is m?ködne, és ugyanezt adná. Alapból ha *metódus*ról beszélünk, az "http://wiki.math.bme.huinstance method"http://wiki.math.bme.hu-öt értjük alatta.

Laboron majd fogunk nézni néhány objektum osztályt a beépített könyvtárból.

Függvény objektumok

A python-ban a függvények is objektumok. Hasonlóan a listához és a szótárhoz, speciális szintaxis van a létrehozására, a *def* parancs. Nézzünk egy példát:

link

Tegyük fel hogy a program más, már korábban megírt részeiben így használjuk a kártyákat, hogy a színük van el?bb, és utána a számuk. De aztán itt szeretnénk a számuk szerint sorba rendezni ?ket, de a *sort()* a párokat magától az els? tagja alapján rendezi. A megoldás egy függvény objektum használata.

Egyik dolog amit itt látunk, hogy a lista *sort()* metódusának van egy *cmp* nev? opcionális paramétere, ami egy függvény objektumot vár. A sort() dokumentációja leírja, hogy olyan függvényt vár, ami ha a lista két elemét kapja paraméterként, akkor negatív számot ad vissza ha az els? kisebb, pozitívat ha a második kisebb, és nullát ha egyenl?ek. (Tehát két egész számnál elég lenne a kett? különbségét visszaadni.) Ezért megírtuk így a *kartya_sorrend()* függvényt és azt adtuk oda.

A függvény objektumok alapvet? tulajdonsága, hogy meghívhatóak. Ez azt jelenti, hogy a függvény neve után tehetek egy zárójelet, abba a függvény paramétereit, és akkor lefut a függvény kódja azokkal a paraméterekkel. De ezen kívül van néhány más tagváltozója és metódusa is, például itt láthatjuk hogy a *__doc__* tagváltozóban megtalálható a docstring-ben megadott dokumentáció.

Lambda függvények

Nézzünk egy másik példát: van egy nagy számunk, és egy listában megvan már néhány osztója. Szeretnénk megnézni, hogy mi a szám ami marad ha ezekkel már elosztottuk. Egy lehetséges megoldás:

```
szam = 30000
osztok = [2, 2, 3, 5]

kisszam = szam
for oszto in osztok:
    kisszam = kisszam / oszto

print kisszam
```

Ez teljesen logikus, remélem mindenki számára érthet?. Most egy alternatív megoldás, ami a beépített reduce() függvényt használja:

```
def oszt(a, b):
    return a/b

kisszam = reduce(oszt, osztok, szam)
```

Sikeresen elértem, hogy a megoldás 3 sor helyett... továbbra is három sor. De, a függvény objektumok létrehozására nem csak egy speciális szintaxis van, hanem kett? is! Ha egy függvény csak egy sorból áll, ami *return*-öl valamit, akkor azt így is le lehet írni, lambda függvénnyel:

```
oszt = lambda a, b : a / b

kisszam = reduce(oszt, osztok, szam)
```

Ez pontosan ugyanazt jelenti mint az el?z? változat. De akkor már nem is kell ezt külön eltárolni egy változóba, egyb?l oda lehet adni paraméternek:

```
kisszam = reduce(lambda a, b : a / b, osztok, szam)
```

Így már csak egy soros lett.

Ez fölösleges bonyolításnak t?nhet most számotokra, de amikor az ember már sokezredjére ír kódot ami végigmegy egy listán, akkor próbálja elkerülni :) De komolyabban mondva, a reduce(), filter() és map() beépített függvények mind egy-egy sztenderd esetet kezelnek, amihez normálisan végig kéne menni egy listán. Ha ezek közül a függvények közül használjuk valamelyiket, akkor a kód olvasója egyb?l láthatja, hogy ez az a tipikus eset, míg ha csak azt látná hogy "http://wiki.math.bme.hufor"http://wiki.math.bme.hu, akkor meg kéne néznie a bels? kódot is. Így segítheti is az érthet?séget, bár persze a tömörségével nehezíti is.

Kisebb témák

Szöveg formatálás

Nézzük a következ? kódot, ami egy "http://wiki.math.bme.huosztótáblát"http://wiki.math.bme.hu ír ki:

```
for i in range(1, 6):
    for j in range(1, 6):
        print i / float(j),
    print
```

Ennek a kimenete így néz ki:

```
1.0 0.5 0.333333333333 0.25 0.2
2.0 1.0 0.666666666667 0.5 0.4
3.0 1.5 1.0 0.75 0.6
4.0 2.0 1.33333333333 1.0 0.8
5.0 2.5 1.66666666667 1.25 1.0
```

(Mellesleg, azt hogy a print nem tesz újsort, ha a végén van egy vessz?, tudtátok? Hasznos!)

Hát ez lehet hogy egy táblázat, de nem túl átlátható. Mennyivel jobb lenne, ha így nézne ki:

```
1.000 0.500 0.333 0.250 0.200
2.000 1.000 0.667 0.500 0.400
3.000 1.500 1.000 0.750 0.600
4.000 2.000 1.330 1.000 0.800
5.000 2.500 1.670 1.250 1.000
```

Vagy esetleg így:

```
1.0    0.5    0.3333 0.25   0.2
2.0    1.0    0.6667 0.5    0.4
3.0    1.5    1.0    0.75   0.6
4.0    2.0    1.333  1.0    0.8
5.0    2.5    1.667  1.25   1.0
```

Többek között erre használható a karakterlánc osztály (angol neve *string*, a python osztály neve *str*) format() metódusa.

Hivatalos példák.

Az az alapvetés, hogy a *format()* metódus visszaad egy másik stringet, amiben a *{}* (kapcsos zárójeles) dolgok ki vannak cserélve az oda ill? értékekre. (A *format()* metódus nem tudja azt az *str* objektumot módosítani amin meghívják, mert az *str* egy immutable típus, tehát nem módosítható.) Egyszer? példa:

```
felhasznalo = "Csirke"
domain = "BME Matematika Intézet"

print "Üdvözöllek {} a {} wikin!".format(felhasznalo, domain)
```

Ebben az esetben csak arra használtam a *format*-ot, hogy a szöveg változtatható részeit különválogattam, ami már magában hasznos. Persze mivel ez egy metódus, ezért ilyen formában is lehet hívni:

```
felhasznalo = "Csirke"
domain = "BME Matematika Intézet"
udvozles = "Üdvözöllek {} a {} wikin!"

print udvozles.format(felhasznalo, domain)
```

Általában a korábbi formát használják. Néha azonban jól jöhet ez, és az els? dolog amit a behelyettesítési helyeken meg lehet adni, pl. ha ugyanezt a kódot szeretném más nyelv? honlap létrehozására is használni:

```
felhasznalo = "Csirke"
domain = "BME Matematika Intézet"
udvozles = "Welcome to {1} wiki, {0}!"

print udvozles.format(felhasznalo, domain)
```

Mint látható, az itt megadott angol szövegben más sorrendben voltak a behelyettesítend? paraméterek, de ez megoldható volt azzal, hogy explicit megmondtuk, hogy hányadik paramétert kérjük, nem az alapértelmezett sorrendet használtuk. Így elég volt csak az üdvözl? szöveget változtatni a kódban. (Ez fontos, mert ha kódot is kell módosítani a különböz? nyelvek használatához, azt sokkal bonyolultabb kezelni, mint ha csak egy adatbázisból a megfelel? szövegeket kell kiolvasni.)

Na de nézzük még milyen dolgokat lehet megadni. Lehet válogatni a paraméterek közül nem csak számokkal, hanem más módon is, ez annyira nem érdekes nekünk, ha valamikor mégis kellene, elolvashatjátok a dokumentációt. Ami érdekesebb nekünk, a táblázatos esetnél például, hogy ha a kapcsos zárójelen belülre teszünk egy kett?spontot, akkor az után pedig a beillesztett dolog formáját lehet változtatni. Legegyszer?bb eset ha csak egy számot teszünk oda, akkor az azt jelenti, hogy legalább annyi bet?b?l fog állni az a mez?. Ez minden fajta bemenettel m?ködik:

```
>>> "b{:5}b".format("a")
'ba    b'
>>> "b{:5}b".format(5)
'b    5b'
```

Valós számoknál még meg lehet adni, hogy hány tizedesjegy pontossággal írja oda a dolgokat, egy pont után:

```
>>> "b{:5.2}b".format(5.12345)
'b  5.1b'
```

További dolgok amiket használhatunk itt: Az elejére írva hogy "http://wiki.math.bme.hu<"http://wiki.math.bme.hu, "http://wiki.math.bme.hu^"http://wiki.math.bme.hu vagy "http://wiki.math.bme.hu>"http://wiki.math.bme.hu, megmondhatjuk, hogy melyik oldalra legyen rendezve a fix hosszon belül a szám. Ha odaírunk egy "http://wiki.math.bme.hu0"http://wiki.math.bme.hu-t, akkor a szabad helyeket 0-kkal tölti fel.

Így a táblázat egy lehetséges megoldása:

```
for i in range(1, 6):
    for j in range(1, 6):
        print "{:<5.3}".format(i / float(j)),
    print
```

Aminek kimenete:

```
1.0   0.5   0.333 0.25  0.2
2.0   1.0   0.667 0.5   0.4
3.0   1.5   1.0   0.75  0.6
4.0   2.0   1.33  1.0   0.8
5.0   2.5   1.67  1.25  1.0
```