Tartalomjegyzék

Fájl kezelés

Fájl megnyitása

Fájlt a python beépített open() függvényével lehet megnyitni, hivatalos leírást röviden itt találhattok a használatáról általában. Ezt hasznos specifikusan megnézni, mert egyszer? programok írójaként a legtöbb programotok fájlokból fog olvasni és fájlokba fog írni.

Az *open()* függvénynek két paramétere van, az els? a megnyitni kívánt fájl neve, a második pedig azt mondja meg, hogy mit akarunk csinálni a fájllal, egy karakterláncban. Egyel?re tekinthetjük úgy hogy a második paraméterre két lehet?ség van, "http://wiki.math.bme.hur"http://wiki.math.bme.hu, ami azt jelenti hogy olvasni szeretnénk a fájlt, és "http://wiki.math.bme.huw"http://wiki.math.bme.hu, ami azt jelenti, hogy felül szeretnénk írni a fájl tartalmát. Az *open()* függvény egy file típusú objektummal tér vissza, a továbbiakban ezen az objektumon keresztül érjük el a fájlt. Nézzük meg miket tud:

Fájl olvasása

Legegyszer?bb módja a fájl végigolvasásának egy *for* ciklus, így valahogy:

```
fajl = open("adatok.txt", "r")

adatok = []
for sor in fajl:
    adatok.append(float(sor))
```

Ez a legegyszer?bb megoldás, de nem mindig alkalmazható, mert az kell hozzá, hogy a fájl összes sorát együtt kezeljük.

A további példákhoz tegyük fel, hogy van egy m?szerünk ami egy adatsort olvasott le, és a számokat kiírta egy fájlba, minden sorba egyet. Ennek az adatsornak a beolvasására jó volt az el?z? példa is. De mi van ha a fájl els? sorába leírja a m?szer a saját azonosítóját (hogy tudjuk hogy honnan származnak az adatok), és csak a többi sor tartalmaz adatokat. Ekkor már mindjárt gondban vagyunk a *for*-al, mert nem minden sort kell egyformán kezelni.

El?ször is még ismerjük meg a file osztály readline() metódusát, ami csak egy sort olvas be a fájlból. Aztán amit még fontos megérteni, hogy a *file* objektum tárolja a bels? állapotában hogy meddig lett már beolvasva a fájl. Ezért m?ködik ebben az esetben a következ? kód:

```
fajl = open("adatok.txt", "r")

muszer_azonosito = fajl.readline().strip()

adatok = []
for sor in fajl:
    adatok.append(float(sor))
```

A *for* ciklus nem az egész fájlt olvassa be, csak ami még hátravan. Még jobb példa erre, ha a m?szer két adatsort is beolvas, mondjuk súlyt és térfogatot, és egy olyan sor választja el a két adatsort, ami csak azt tartalmazza hogy "http://wiki.math.bme.hu--"http://wiki.math.bme.hu, tehát az adatfájl valahogy így néz ki:

```
Acme Muszer
1.21253
2.44362
3.19158
--
7.746
5.315
4.196
```

Ezt be lehet így olvasni:

```
fajl = open("adatok.txt", "r")

muszer_azonosito = fajl.readline().strip()

sulyok = []
for sor in fajl:
    if sor.strip() == "--":
        break
    sulyok.append(float(sor))

terfogatok = []
for sor in fajl:
    terfogatok.append(float(sor))
```

Ugyan két *for* ciklus is van, ami elvileg a *fajl* változón megy végig, mégis más eredményeket produkálnak. Ez azért van, mert a *file* típusú objektumok tárolják magukon belül hogy hol tartanak a fájlban, és a for ciklusnál is csak onnan kezdik olvasni ahol épp tartanak. (Van speciális metódus amivel lehet el?re vagy vissza ugrani a fájlban, a metódusok leírása között megtaláljátok, azok ritkábban kellenek.)

Fájl Írása

Ehhez képest a fájl írása egyszer?. A *file* osztálynak van egy write() metódusa, amivel egy karakterláncot beleírhatunk a fájlba.

```
fajl = open("adatok.txt", "w")
```

```
muszer = Muszer()

fajl.write(muszer.azonosito + "\n")

for suly in muszer.sulyok():
    fajl.write("{:.6}\n".format(suly))

fajl.write("--\n")

for terfogat in muszer.terfogatok():
    fajl.write("{:.4}\n".format(terfogat))

fajl.close()
```

Ez nagyjából hasonló ahhoz ahogy a *print*-tel kiírunk a parancssorba. Fontos különbségek, hogy ez csak egy karakterláncot fogad el, ellenben a *print*-tel, ami több dolgot is kaphat, és nem kell karakterláncnak lenniük, akkor is át lesznek konvertálva. Itt nekünk kell átkonvertálni, vagy a *format*-tal, ahogy a fenti példában, vagy az *str* konstruktorral, így valahogy:

```
l = [5, 6, 7]

fajl.write(str(l))
```

Másik fontos különbség, hogy a *print* automatikusan beszúr szóközöket az elemek közé, és újsorokat a sor végére, míg a *write*-nál mindkett?t nekünk kell megcsinálni. Az újsort a python karakterláncokban a "http://wiki.math.bme.hu\n"http://wiki.math.bme.hu jelképezi, mint látható a fenti példában.

A harmadik fontos különbség, hogy a *write()*-tal még nem vagyunk kész, mert technikai okok miatt nem mindig írja ki egyb?l a merevlemezre amit odaadtunk a *write()*-nak. A végén még le kell zárni a fájlt, ezzel jelezzük hogy kész vagyunk, most már mindent ténylegesen írjon ki. A close() metódus meghívásával tudjuk ezt megtenni, viszont miután ez megtörtént, utána már nem lehet írni a fájlba.

Ezt könny? elfelejteni, és néha el?fordulhat hogy nem felejtjük el, de a kódnak az a része ahol a *close()* van át lesz ugorva (mert egy *break*, vagy *return* vagy akár egy kivétel máshova ugrik a kódban), ezért van egy speciális szintaxis, ami azt jelenti hogy "http://wiki.math.bme.hua kód ezen a részen használom a fájlt, amikor ezt bárhogy elhagyom, zárjuk le"http://wiki.math.bme.hu. Ez a with kulcsszó, amit így kell használni (az el?z? példát módosítva):

```
with open("adatok.txt", "w") as fajl:

    muszer = Muszer()

    fajl.write(muszer.azonosito + "\n")

    for suly in muszer.sulyok():
        fajl.write("{:.6}\n".format(suly))

    fajl.write("--\n")

    for terfogat in muszer.terfogatok():
        fajl.write("{:.4}\n".format(terfogat))
```

Ajánlott a *with* használata mindig amikor lehet, ha fájlt nyitunk meg, akár írásra, akár olvasásra, mert egyszer?síti a kódot, és így biztos használjuk amikor kell.

(Amúgy fájl olvasásnál is jó ha lezárjuk a fájlt így, vagy a *close()*-al, de kevésbé fontos, csak a hatékonyságon változtat. Hacsaknem több ezer fájlt megnyit a programunk, akkor ez a hatékonyság változás

nem jelent?s.)

Iterátorok

Igény

Nézzük a következ? példát:

```
lista = [
    'Acme Muszer\n',
    '1.21253\n', '2.44362\n', '3.19158\n', '--\n',
    '7.746\n', '5.315\n', '4.196\n']

for sor in lista:
    print sor,

with open("adatok.txt", "r") as fajl:
    for sor in fajl:
        print sor,
```

Ez a kód (még mindig a korábbi példa fájlt használva) ugyanazt írja ki kétszer. A *for* ugyanúgy tud végigmenni rajtuk. Azonban, mint láttuk a fájl olvasás részben, a fájl néha nem egészen úgy m?ködik mint a lista, pl. az el?bb látott olvasási példánál. Nézzük meg azt a példát függvényként megvalósítva:

```
def olvas_fajlbol(fajl):
    muszer_azonosito = fajl.readline().strip()

    sulyok = []
    for sor in fajl:
        if sor.strip() == "--":
            break
        sulyok.append(float(sor))

    terfogatok = []
    for sor in fajl:
        terfogatok.append(float(sor))

    return sulyok, terfogatok

with open("adatok.txt", "r") as adat_fajl:
    sulyok, terfogatok = olvas_fajlbol(adat_fajl)
    print sulyok, terfogatok
```

Majd ha ugyanilyent próbálunk a listára csinálni, azt máshogy kell, for a *for elem in l* mindig a lista elejér?l indulna. Itt egy megoldás, pl. így lehet:

```
def olvas_listabol(l):
    muszer_azonosito = l[0]

    i = 1
    sulyok = []
    while True:
        if lista[i].strip() == "--":
            break
        sulyok.append(float(lista[i]))
        i += 1

    i += 1
    terfogatok = []
```

```
    while i < len(l):
        terfogatok.append(float(lista[i]))
        i += 1

    return sulyok, terfogatok

adat_lista = [
    'Acme Muszer\n',
    '1.21253\n', '2.44362\n', '3.19158\n', '--\n',
    '7.746\n', '5.315\n', '4.196\n']
sulyok, terfogatok = olvas_listabol(adat_lista)
print sulyok, terfogatok
```

Ez a két kód elég különböz?. Pedig a *for*-al végigmenve ugyanazt kaptam, úgyhogy kell lennie olyan megoldásnak, ami együtt tudja mindkett?t kezelni. Erre jó az iterátor.

Iterátor objektumok

Az iterátorok egy speciális objektum típus, amiket a speciális iter() beépített függvénnyel lehet létrehozni. Az a lényegük, hogy a segítségükkel végig lehet menni bármilyen sorozat jelleg? dolgon, amin a *for* is végig tudna menni. A legfontosabb tulajdonságuk, hogy van egy next() metódusuk, ami mindig visszaadja a következ? elemet. És az iterátoron belül le van tárolva, hogy éppen hol tartunk a sorozatban.

Már csak az a kérdés, hogy mi történik, ha a végére ér a sorozatnak, és nincs több elem? Ilyenkor kivételt emel, aminek StopIteration a típusa, úgyhogy azt kell várni.

link

Az iterátort is lehet *for* ciklusban is használni, és akkor (mint a fájl) megjegyzi hogy meddig haladt már el, ha nem hagyjuk a *for* ciklust végigfutni, hanem *break*-kel kilépünk, akkor az iterátor még nincs a végén, és lehet továbbra is használni a sorozat további részének megnézésére. Ezt figyelembe véve általános megoldás a korábbi adatbeolvasásra pl.:

```
def olvas(sorozat):
    it = iter(sorozat)

    muszer_azonosito = it.next().strip()

    sulyok = []
    for sor in it:
        if sor.strip() == "--":
            break
        sulyok.append(float(sor))

    terfogatok = []
    for sor in it:
        terfogatok.append(float(sor))

    return sulyok, terfogatok

with open("adatok.txt", "r") as adat_fajl:
    sulyok, terfogatok = olvas(adat_fajl)
    print sulyok, terfogatok

adat_lista = [
    'Acme Muszer\n',
    '1.21253\n', '2.44362\n', '3.19158\n', '--\n',
    '7.746\n', '5.315\n', '4.196\n']
sulyok, terfogatok = olvas(adat_lista)
```

```
print sulyok, terfogatok
```

És ez a kód már nem csak a fájlra és a listára m?ködik, hanem bármire, amire a *for* ciklus is, így elég általános.

itertools

Másik el?nye az iterátoroknak, hogy van egy külön beépített könyvtár, az itertools, ami a mindenféle ügyes kezelésükre van. Csak egy példa a tee() függvénnyel:

link

Saját osztályba iterátor

Az osztály speciális, __iter__() nev? metódusa az amit le kell kódolni, hogy a mi objektumjaink is terálhatóak legyenek, m?ködjön rájuk a *for* ciklus. Ennek a függvénynek egy iterátort kell visszaadnia, és ugyan saját iterátor osztályt írni kicsit nehezebb, a legtöbb tipikus esethez használhatunk el?re elkészített dolgokat az *itertools* könyvtárból.

Például ha szeretnénk hogy a m?szerünkön végig-iterálva a súly-térfogat párokat párokként adja vissza, itt egy lehetséges megoldás:

link

Nagy HF

Részletek

Kitekintés

Ha szeretnétek a python-t tovább tanulni, és nem elfelejteni.

Magyarul

A python.hu-n található egy gy?jtemény a különböz? magyar nyelv? anyagokról itt.

Sajnos nem sokan foglalkoznak ezzel, ezért sok ilyen magyar anyag elavult.

Van egy részleges fordítása a hivatalos python tutorial-nak, ami 2006-os és a python 2.4-es változatához készült (mi most a 2.7-est használjuk). (Amiatt viszont nem kell aggódni, hogy részleges, ami számotokra egyel?re fontos lehet, azt már mind lefedi.)

Ez a másik pdf oktatóanyag jónak néz ki els? ránézésre, és legalább 2010-es, az már jóval közelebbi python a ma használthoz.

Angolul

El?ször is természetesen ott a hivatalos dokumentáció, aminek része a python tutorial, ami elég alapos, de talán kicsit túl száraz.

Els?dleges programozás tanulási ajánlat általában a www.codecademy.com, bár az ottani python tutorial nem sokkal fed le többet mint amit ebb?l a tárgyból megtanultunk, úgyhogy a továbbfejl?désre annyira nem jó, de ha esetleg 3-4 év múlva fel kell frissítenetek a tudásotok, akkor arra tökéletes lehet, nagyon barátságos kezel?felülettel, és jól megalkotott feladatokkal.

Ha valaki hozzám hasonlóan szeret versengeni, akkor annak jó lehet valamilyen online programozó versenyen részt venni, én tudom ajánlani a TopCoder-t. Körülbelül heti rendszerességgel vannak versenyek, úgyhogy könny? alkalmat találni, és egy verseny másfél óra, úgyhogy nem olyan nagy id?befektetés. Lehet python-t használni. 3 feladat van, és az els? általában körülbelül olyan szinten van, vagy néha könnyebb, mint amiket itt laboron feladtam, úgyhogy már ennyi tanulás után is lehetnek sikerélményeitek valamennyire ott. A beadó rendszerük használata nem teljesen intuitív, most nem szánok 5-10 percet a kényelmes használat elmagyarázására (azért gondolom a többséget nem érdekli), ha valaki szeretne egy kis segítséget az elindulásban, dobjon egy emailt, és segítek.

Következ? hétt?l C!