Failide kasutamine

Andmete pikemaajaliseks säilitamiseks on sobiv neid kirjutada faili. Siis saab neid ka programmi uuel käivitamisel kasutada. Fail ei pruugi alati olla andmefail selle otseses mõttes. Kuid siin kursuses keskendume just andmefailide kirjutamisel ja lugemisele. Räägime tekstifailidest. Tekstifailid koosnevad sümbolitest (character) ja andmevahetus tekstifailiga kulgeb stringide kaudu.

Faili kasutamiseks tuleb ta kõigepealt avada. Alles seejärele on sinna võimalik midagi kirjutada või sealt otsida.

Faili avamisel tagastab vastav funktsioon (`open()`) nn failiobjekti ja selle objekti abil saab edaspidi nimetatud failiga "suhelda". St funktsioon (OOP mõistetes konstruktor) `open()` loob failiobjekti.

Faili **avamiseks** on järgmine käsk (tegelikult funktsioon)

```
faili_object = open(faili_nimi, ligipääs='r', puhvrikasutus=-1)
```

Tagastatav `faili_objekt` on faili-tüüpi muutuja. `Faili_nimi` on avatava (või loodava) faili nö pärisnimi, mille all ta operatsioonisüsteemi mõttes eksisteerib. Nimi võib sisaldada nii ketta kui kataloogitähiseid. `Ligipääs` seab paika, mida failiga teha saab. Vaikimisi avatakse fail ainult lugemiseks `(r)`. Lisaks võib ta avatud olla kirjutamiseks `(w)` ja lõppu lisamiseks `(a)`. Täiendavate võimaluste kohta võib lugeda raamatutest ja abist. `Puhvrikasutus` on seotud sellega, kui suurte osade kaupa faili infot kirjutatakse või failist loetakse. Negatiivne arv (ja parameetri puudumine) ütleb, et kasutatagu op-süsteemi vaikeseadeid. Positiivne arv annab korraga loetavate/kirjutatavat puhvri suuruse. Tavaliselt soovitatakse kasutada op-süsteemi vaikeseadet. Puhvri mõtteks on koguda kokku „ports" infot ja see siis korraga faili kirjutada, et kiirendada programmi tööd. Sest kõvakettaga suhtlemine on ajamahukas ettevõtmine.

Et failmuutuja on objekt, siis saab temaga tööd teha meetodeid kasutades (stringide juures sai erinevusest tavalise funktsiooniga räägitud!!)

Esimene oluline meetod on sulgemismeetod `close()`. Peale kirjutamist tuleb fail sulgeda. Muidu ei kirjutata sinna viimast puhvritäit andmeid ära ja fail võib jääda poolikuks.

Näide: fail „`asi.txt`" avatakse kirjutamiseks ning seejärel suletakse:

```
fm = open(„asi.txt",„w")
fm.close()   #NB! Tuleta meelde, kuidas meetodit välja kutsuti!
```

Selle tulemusena tekib tühi fail, sest ühtegi faili kirjutamise käsku ei antud.

Faili kirjutamine

Faili kirjutamiseks on meetod `write()`. Write vajab sisendiks stringi, ning ta kirjutab selle faili. Seega vajadusel tuleb arvulised väärtused enne kirjutamist teisendada stringideks. Automaatselt ühtegi reavahetust ei lisata, seega tuleb nende eest ise hoolt kanda. Meetodiga `writelines()` saab faili kirjutada terve stringidest koosneva listi.

Näide: fail „`asi.txt`" avatakse kirjutamiseks, kirjutatakse sinna *'Tere talv!'* ning seejärel suletakse:

```
fm = open(„asi.txt",„w")
fm.write(„Tere talv!")
fm.close()
```

Et failiobjekt on ka järjehoidja, kirjutab uus `write()`-meetodi väljakutse väljundi senise teksti lõppu. Kuid vahepeal ei tohi faili `close()`-iga sulgeda!!

Stringide saamiseks on kaks peamist moodust. Kõigepealt funktsioon `str()`, mis parameetriks oleva arvu stringiks teisendab. Näiteks nii:

```
karude_arv = 12
fm.write(str(karude_arv))
```

Abiks on ka `print`-käsu juures räägitud vormindamine `%`-i abil. Et vorminduskäsud asuvad stringi sees, on kogu tulemus string ja seega sobilik faili kirjutamiseks. Täpselt samasugust vormindamist lubavad kasutada meetodid `write()` ja `writelines()`. Näiteks nii:

```
karude_arv = 12
fm.write('Metsas oli %d karu' % (karude_arv))
```

Muuhulgas annab see võimaluse ujukoma arve püsikoma kujul salvestada.

Failist lugemine

Failist lugemiseks saab kasutada meetodit `readline()`. Meetod loeb falist ühe rea (st baidid kuni reavahetuseni, viimane kaasaarvatud) ning see info on võimalik omistada stringitüüpi muutujale. Parameetrina on võimalik ka ette anda korraga loetavate baitide arvu. Seda tehakse siiski vaid erilise vajaduse korral. Meetodiga `readlines()` saab korraga sisselugeda kogu faili. Tulemuseks on stringidest koosnev list – iga rida on eraldi listi element.

Näide: fail „`asi.txt`" avatakse lugemiseks, sellest loetud rida trükitakse ekraanile

```
fm = open(„asi.txt")       # Lugemiseks avatakse fail vaikimisi
read = fm.readlines()
print(read)
fm.close()
```

Suuremahulise faili korral ei pruugi terve faili listi lugemine otstarbekas olla. Juba tuttava `for`-tsükli abil on võimalik lugeda kogu fail välja rida haaval. Näiteks nii:

```
for rida in fm:
    print(rida)
```

Loomulikult saab väljatrüki asemel teha kõikvõimalikke muid toiminguid.

Failist lugemisel on failiobjekt ka nö järjehoidjaks. Iga järgmine `readline()`-meetodi käivitamine loeb sisse järgmise rea failist. Mida ja kuidas pihta hakata failist kätte saadud stringidega on juba konkreetse ülesande ja stringitöötluse küsimus. Teades ette faili struktuuri ei pruugi olla võimalik või mõistlik kõiki ridu ühel viisil lugeda ja töödelda. Võimalik, et esimesed read sisaldavad hoopis infot järgnevate ridade ülesehituse kohta. Sel juhul oleks vaja ridade edasisele töötlemisele erineval viisil läheneda. Sest enamasti ei ole sisseloetud stringiga niisama tervikuna midagi pihta hakata.

Et lugemisel reavahetuse märke automaatselt ei likvideerita, siis kasutatakse tihti stringi-meetodit `strip()` rea reavahetuse sümbolist vabanemiseks. Meetod `strip()` kõrvaldab stringi lõpust mittetrükitavad sümbolid – reavahetused, tühikud, ... Näiteks nii:

```
rida = fm.readline()
rida = rida.strip()
```

Nimed ja otsiteed

Kust faili otsitakse? Ja kuhu fail kirjutatakse? Milline saab olla faili nimi? Mis juhtub siis, kui faili ei leita?

Failidega töötamisel on ka nendele küsimustele vaja vastuseid. Kõigepealt eristatakse **absoluutset** ja **suhtelist failinime** ning **otsiteed** *(absolute path, relative path)*. Ja oluline on ka mõiste **jooksev kataloog** *(current directory)* või siis **töökataloog** *(working directory)*.

Faili nimi kujul `'minufail.txt'` on **suhteline faili nimi** ja faili tegelik asukoht sõltub töökataloogist, st faili otsitakse või kirjutatakse töökataloogi. Töökataloogiks on tavaliselt see kataloog, kus paikneb käivitatav programm.

Faili **suhteline nimi ja otsitee** on näiteks järgmine: `'andmed\minufail.txt'`. See tähendab, et töökataloogis on kataloog `andmed` ning fail paikneb seal.

Faili **absoluutne nimi ja otsitee** avaldub aga kujul: `'\if07\Maali\Prog_alused\minufail.txt'`. Siia võib lisanduda ka kettatähis (`C:, Y:` vms). Faili asukoht esitatakse kataloogipuu juurest lähtudes (mis ei pruugi serveri

tegelikust kataloogipuust rääkides absoluutne tõde olla).

Kõiki eelpool nimetatud võimalusi tohib faili nime esitamisel `open()`-funktsioonis kasutada. Ainus tingimus on, et see korrektne oleks ja et failid-kataloogid olemas oleksid.

Võimalusi otsiteede uurimiseks ning failide-kataloogide olemasolu tuvastamiseks pakub moodul `os`. Kasutusele saab mooduli võtta juba tuntud viisil: `import os`.

Mõned kasulikud funktsioonid:

`os.getcwd()` - tagastab töökataloogi nime. Võid seda proovida nii *Python Shellis* kui programmi abil välja trükkida.

`os.path.exists(failinimi)` – on `boolean`-tüüpi funktsioon ning ta annab teada, kas parameetriks olev fail või kataloog eksisteerib. Suhtelise failinime korral otsitakse töökataloogi suhtes.

`os.listdir(kataloog)` – tagastab listi, mis koosneb kataloogi sisuks olevatest failide ja kataloogide nimedest; kataloog võib olla esitatud nii suhtelise kui ka absoluutse nimega.

Vea- ehk erinditöötlus

Probleemid tekivad faili avamisel lugemiseks, kui soovitud faili ei ole (või teda ei ole selles kohas, kust teda otsitakse). Kui fail avatakse kirjutamiseks, võib viga tekkida juhul kui on määratud olematu kataloogi nimi. Fail ise tekib ju uuena, seega ei pea teda eelnevalt olemas olema. Kuid faili kirjutamiseks soovitud kohta ei pruugi programmi kasutajal olla piisavalt õigusi. Ka see olukord põhjustab faili avamisel vea. Et vea võimalusi on veelgi, on tülikas kirjutada programmi, mis enne avamist mooduli `os` võimalusi kasutades kõiki vigu kontrollida ja ennetada püüab.

Lihtsam viis on kasutada `try: ... except`: püünist erindi püüdmiseks:

```
try:
    fm = open('olematu_fail.txt')
    for rida in fm:
        print(rida)
    fm.close()
except:
    print(„Miski on mäda!“)
```

Antud näitest ei ole tegelikkuses suuremat kasu, kuid vea korral saab ka anda võimalusi näiteks uue failinime sisestamiseks jms, et veaolukorrast väljuda. Või vähemalt programmi töö viisakalt lõpetada. Lisaks ei ole `try..exept` püünise lisamine programmi kirjutamise ja silumise ajal hea ka seetõttu, et ainsad vead siin ei pruugi tuleneda faili puudumisest. Ka sisendi töötlemisel saavad tekkida probleemid (nt stringi teisendamisel arvuks) ning üldine veateade, et miski mäda on, ei aita viga mitte kuidagi lokaliseerida. Seega siis: programmi kirjutades väldi esialgu erinditöötlust, loe täitmisaegseid veateateid ja kõrvalda nende abil vead programmist.