

Tekstifailide kasutamine - andmefailid

Suuremaid andmehulkasid on sobiv panna andmebaasidesse. Lihtsamal juhul sobib aga andmete pikemaajaliseks säilitamiseks tavaline faili. Failist saab andmeid programmi uuel käivitamisel lugeda. Andmefailidena saab kasutada nii **tekstifaili** (*ingl text file*) kui ka **binaarfaile** (*ingl binary file*). Tekstifail koosneb sümbolitest (*ingl character*) ehk kõik salvestatakse justkui stringidena. Andmevahetus tekstifailiga toimub stringide kaudu.

Järgnevas tegeleme tekstifailidega.

Faili kasutamiseks tuleb ta kõigepealt avada, seejärel saab faili kirjutada või failist lugeda.

Fail avatakse funktsiooniga `open()` (OOP mõistetes konstruktor) ning avamise tulemusena luuakse failiobjekt. Edasine suhtlemine failiga toimub **failiobjekti** ehk failimuutuja (*ingl file object*) kaudu.

```
failiobject = open(faili_nimi, ligipääs="r", encoding=tekstifaili kodeering)
```

Muutuja failiobjekt on failiobjekt ehk failimuutuja. Faili_nimi on avatava (või loodava) faili nõ pärisnimi, mille all ta operatsioonisüsteemi mõttes eksisteerib. Nimi võib sisaldada nii ketta kui kataloogitähiseid. Ligipääs seab paika, mida failiga teha saab. Tekstifail võib olla avatud kahes režiimis: lugemiseks või kirjutamiseks. Vaikimisi avatakse fail lugemiseks ("r"). Ligipääsutähisega "w" avatakse fail kirjutamiseks ning kui sellise nimega fail on juba olemas, siis kirjutatakse ta üle. Ligipääsutähisega "a" avatakse fail lõpu lisamiseks. Täiendavate võimaluste kohta võib lugeda raamatutest ja abist. Parameetriga encoding määratakse failis kasutatav kodeering. Vaikimisi määratud kodeeringu saab teada funktsiooniga `locale.getpreferredencoding()` (moodul locale tuleb eelnevalt importida). Peamised väärtused parameetri encoding jaoks on "cp1257", "utf-8", "iso8859-1"

Kõvakettal oleva failiga ei suhtle programm otse, vaid suhtlemist vahendab mälus olev failipuhver, mis kiirendab tööd. Puhvri mõtteks on koguda kokku „ports“ andmeid ja siis korraga faili kirjutada. Peale töö lõppu tuleb fail sulgeda: `failiobjekt.close()`. Kui kirjutatavat faili ei suleta, võib osa andmeid jääda puhvrast faili ringikirjutamata.

Kirjutamine faili

Faili kirjutamiseks on meetod `write()`, mis ootab argumendiks ühte stringi. Vajadusel tuleb arvud enne faili kirjutamist teisendada stringideks. Automaatselt ühtegi reavahetust ei lisata, seega tuleb nende eest ise hoolt kanda. Meetodiga `writelines()` saab faili kirjutada terve stringidest koosneva listi, kuidas ka see läheb faili ühegi reavahetuseta.

Näide: fail "tervitus.txt" avatakse kirjutamiseks, kirjutatakse sinna "Tere talv!" ning seejärel suletakse. Faili kodeeringuks on UTF-8:

```
fm = open("tervitus.txt", "w", encoding = "utf-8")
fm.write("Tere talv!")
fm.close()
```

Et failiobjekt töötab ka kui järjehoidja, kirjutab iga järgmine meetodi `write()` väljakutse järgmise väljundi senise teksti lõppu. Kuid vahepeal ei tohi faili sulgeda!! Kui eelmise rea kirjutamisel on lisatud reavahetuse märk, siis järgmine `write()` alustab kirjutamist uuel realt.

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`-funktsiooni kontekstis tutvustatud vormindamine `format()`-meetodi abil. Et vorminduskäskud 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 {0:d} karu".format(karude_arv))
```

Muuhulgas annab see võimaluse ujukoma arve püsikoma kujul salvestada, määrates sobiliku murdosa täpsuse (komakohtade arvu).

Lugemine failist

Failist lugemiseks saab kasutada meetodit `readline()`. Meetod loeb failist ühe rea (st baidid kuni reavahetuseni, viimane kaasaarvatud) ning see info on võimalik omistada stringitüüpi muutujale. Parameetriks saab lisada 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 listi element.

Näide: fail "andmefail.txt" avatakse lugemiseks, eeldatakse, et fail on kodeeringus utf-8. Failist loetud rida trükitakse ekraanile

```
fm = open("andmefail.txt", "r", encoding = "utf-8")
k6ik_read = fm.readlines()
print(k6ik_read)
fm.close()
```

Suuremahulise faili korral ei pruugi terve faili listi lugemine otstarbekas olla. for-tsükli abil on võimalik lugeda kogu fail välja rida haaval (lugemine kuni faili lõpuni). Näiteks nii:

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

Eelneva näitega seoses peab arvestama, et siin on kenasti ära peidetud kaks olulist tegevust – failist lugemine ja faili lõputähise jälgimine.

Loomulikult saab väljatrüki asemel teha kõikvõimalikke teisi toiminguid.

Faili lugemisel on failiobjekt ka nõ järjehoidjaks. Iga järgmine `readline()`-meetodi käivitamine loeb failist stringina järgmise rea. Mida ja kuidas loetud stringidega pihta hakata, on ülesande, failistruktuuri ja stringitöötamise küsimus. Teades ette faili struktuuri ei pruugi olla võimalik või mõistlik kõiki ridu ühel viisil töödelda. Võimalik, et esimesed read sisaldavad hoopis infot järgnevate ridade ülesehituse kohta. Sel juhul on vaja ridade edasisele töötlemisele erineval viisil läheneda.

Et lugemisel reavahetuse märke automaatselt ei likvideerita, kasutatakse tihti stringi-meetodit `strip()` rea reavahetuse sümbolist vabanemiseks. Täpsemalt kõrvaldab meetod `strip()` nii stringi lõpust kui algusest nn *whitespace* 'i ehk mittetrükitavad sümbolid ehk tühemikud – reavahetused, tühikud, ... Näiteks nii:

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

Faili nimi ja otsitee

Kust faili otsitakse? Kuhu fail kirjutatakse? Milline on lubatud faili nimi? Mis juhtub, kui faili ei leita?

Eristatakse **absoluutteed (täielikku teed)** (algab juurkataloogist) ja **suhtelist teed** (algab jooksvast kataloogist) (*ingl absolute path, relative path*). Olulised mõisted on **jooksev kataloog** või **töökataloog** (*ingl current directory, 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, kuid see võib ka sõltuda töökeskkonn seadistustest.

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

Faili absoluuttee avaldub aga kujul: '\Maali\Prog_alused\minufail.txt'. Siia võib lisanduda ka kettatähis (C:, D: vms). Faili asukoht esitatakse kataloogipuu juurelemendist lähtudes (mis ei pruugi serveri tegelikust kataloogipuust rääkides absoluutne tõde olla).

Kõiki eelpool nimetatud võimalusi tohib faili nime esitamisel funktsioonis `open()` 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 kataloogis olevate failide ja kataloogide nimedest; kataloog võib olla esitatud nii suhtelise kui ka absoluutse nimega.