Ohjelmoinnin perusteet Y Python

T-106.1208

4.3.2009

Tiedostot

- Tiedostojen käsittelyä tarvitaan esimerkiksi seuraavissa tilanteissa:
  - Ohjelman käsittelemiä tietoa halutaan säilyttää ohjelman suorituskerrasta toiseen (esim. puhelinluettelo, opiskelijarekisteri).
  - Halutaan, että käyttäjän ei tarvitse syöttää ohjelman lähtötietoja jokaisella suorituskerralla, vaan lähtötiedot (esimerkiksi mittaussarjan parametrit) luetaan tiedostosta.
  - Ohjelman on käsiteltävä jonkun muun ohjelman tuottamaa dataa.
- Ohjelma lukee tarvittavat lähtötiedot tiedostosta.
- Jos ohjelma tekee tietoihin muutoksia ja muuttuneita tietoja halutaan käyttää seuraavalla suorituskerralla, ohjelma kirjoittaa muuttuneet tiedot tiedostoon.

Tekstitiedosto vs. binääritiedosto

- Tiedostot jaetaan tekstitiedostoihin ja binääritiedostoihin.
- Tekstitiedostossa tiedot on tallennettu merkkeinä, esimerkiksi luku 147 merkkeinä 1, 4 ja 7.
- Tekstitiedostoa voi muokata millä tahansa tekstieditorilla.
- Binääritiedostossa tiedot on esitetty binääriesitysmuodossa, esimerkiksi luku 147 vastaavana binäärilukuna.
- Binääritiedostoa ei yleensä pysty käsittelemään järkevästi tavallisella tekstieditorilla.
- Tällä kurssilla opetetaan ainoastaan tekstitiedostojen käsittely.

Tiedoston avaaminen

- Kun ohjelma haluaa lukea tai kirjoittaa tekstitiedostoon, on ohjelmalle kerrottava, mikä fyysinen tiedosto vastaa ohjelmassa käytettyä tiedostomuuttujaa.
- Samalla käyttöjärjestelmäpuolella varaudutaan käsittelemään ko. tiedostoa.
- Tätä kutsutaan *tiedoston avaamiseksi*.
- Esimerkki tiedoston avaamisesta

```
tiedostomuuttuja = open("teksti.txt","r")
```

- Ensimmäinen parametri on käsiteltävän tiedoston nimi käyttöjärjestelmässä. Jos tiedosto ei ole samassa hakemistossa kuin missä ohjelmaa ajetaan, on nimeen sisällytettävä polku tiedoston hakemistoon.
- Toinen parametri kertoo tiedoston käsittelytavan. Arvo `"r"` kertoo, että tiedosto avataan lukemista varten.

Lisää tiedoston avaamisesta

- Mahdolliset käsittelytavat:
  - `r` tiedosto avataan lukemista varten
  - `w` tiedosto avataan kirjoittamista varten, vanha sisältö häviää
  - `a` tiedosto avataan kirjoittamista varten, kirjoitetaan vanhan sisällön perään.
- Tiedoston avaaminen lukemista varten aiheuttaa `IOError`-tyyppisen poikkeuksen, jos tiedostoa ei ole tai sitä ei pystytä jostain muusta syystä lukemaan.
- Myös moni muu virhe tiedoston lukemisessa tai siihen kirjoittamisessa voi aiheuttaa `IOError`-tyyppisen poikkeuksen. Sen vuoksi poikkeus on syytä käsitellä try–except-rakenteella aina, kun luetaan tiedostosta tai kirjoitetaan tiedostoon.

Rivin lukeminen ja tiedoston sulkeminen

- Jos muuttuja `tiedostomuuttuja` viittaa lukemista varten avattuun tiedostoon, niin siitä voi lukea rivin kerrallaan metodin `readline` avulla seuraavasti:

  ```
  luettu_rivi = tiedostomuuttuja.readline()
  ```

  Luettu rivi sisältää myös sen lopussa olevan rivinvaihtomerkin.
- Seuraava `readline`-käsky lukee tiedoston seuraavan rivin jne.
- Jos tiedosto on jo luettu loppuun ja kutsutaan `readline`-metodia, se palauttaa arvona tyhjän rivin `""`
- Kun tiedoston lukeminen päättyy, tiedosto pitää sulkea `close`-käskyllä:

  ```
  tiedostomuuttuja.close()
  ```

Esimerkkiohjelma tiedoston lukemisesta

```
def main():
    try:
        lahtotiedosto = open("tekstia.txt", "r")
        rivi = lahtotiedosto.readline()
        while rivi != "":
            print rivi
            rivi = lahtotiedosto.readline()
        lahtotiedosto.close()
    except IOError:
        print "Virhe tiedoston lukemisessa. Ohjelma paattyy."


main()
```

- Edellisen kalvon esimerkkiohjelma lukee tiedostosta rivin kerrallaan ja tulostaa sen käyttäjälle.
- Ohjelma tulostaa kuitenkin ylimääräisen tyhjän rivin jokaisen rivin jälkeen.
- Tämä johtuu siitä, että tiedostoista luettujen rivien lopussa on rivinvaihtomerkki.
- Jos ylimääräiset rivinvaihdot halutaan välttää, pitää rivinvaihtomerkki poistaa tiedoston lopusta ennen rivin tulostamista.
- Yksi tapa poistaa rivinvaihtomerkki on käyttää metodia `rstrip`. Se poistaa kuitenkin myös muut "tyhjät merkit" rivin lopusta.
- Seuraava esimerkkiohjelma käyttää tätä tapaa. Se myös kysyy luettavan tiedoston nimen käyttäjältä.

Parannettu versio tiedostonlukuohjelmasta

```
def main():
    nimi = raw_input("Anna luettavan tiedoston nimi: ")
    try:
        lahtotiedosto = open(nimi, "r")
        rivi = lahtotiedosto.readline()
        while rivi != "":
            rivi = rivi.rstrip()
            print rivi
            rivi = lahtotiedosto.readline()
        lahtotiedosto.close()
    except IOError:
        print "Virhe tiedoston", nimi, \
            "lukemisessa. Ohjelma paattyy."

main()
```

Tiedoston rivien lukeminen for-käskyllä

- Jos ohjelman on luettava kaikki tiedoston rivit, on usein helpointa käydä ne läpi for-käskyn avulla.
- Käskyn yleinen muoto on

```
for rivi in lahtotiedosto:
    tee jotain riville rivi
```

- Käskyyn ei tarvitse kirjoittaa lainkaan rivin tiedostosta lukevaa käskyä (esim. `readline`), vaan for-käsky pitää huolen siitä, että rivit luetaan tiedostosta tarvittaessa.
- Seuraavan kalvon esimerkkiohjelma lukee rivit käyttäjän antamasta tiedostosta ja tulostaa ne kuvaruudulle for-käskyn avulla.

Esimerkki tiedoston lukemisesta for-käskyllä

```
def main():
    nimi = raw_input("Anna luettavan tiedoston nimi: ")
    try:
        lahtotiedosto = open(nimi, "r")
        for rivi in lahtotiedosto:
            rivi = rivi.rstrip()
            print rivi
        lahtotiedosto.close()
    except IOError:
        print "Virhe tiedoston", nimi, \
            "lukemisessa. Ohjelma paattyy."


main()
```

Esimerkki: rivin etsiminen tiedostosta

- Seuraavan kalvon esimerkkiohjelma pyytää käyttäjältä tiedoston nimen ja yhden henkilön nimen.
- Se tutkii, löytyykö annettu henkilön nimi tiedostosta joltain riviltä.
- Oletetaan, että kukin tiedoston rivi sisältää vain yhden nimen.
- Vastaavaa rakennetta voi käyttää, jos halutaan etsiä tiedostosta mitä tahansa tekstiriviä.

Rivin etsiminen, koodi

```
def main():
    nimi = raw_input("Anna tiedoston nimi: ")
    etsittava_nimi = raw_input("Anna etsittava nimi: ")
    loytyi = False
    try:
        lahtotiedosto = open(nimi, "r")
        for rivi in lahtotiedosto:
            rivi = rivi.rstrip()
            if rivi == etsittava_nimi:
                loytyi = True
        lahtotiedosto.close()
        if loytyi:
            print "Nimi", etsittava_nimi, "loytyi."
        else:
            print "Nimea", etsittava_nimi, "ei loytynyt."
```

Rivin etsiminen, koodi jatkuu

```
    except IOError:
        print "Virhe tiedoston", nimi, \
            "lukemisessa. Ohjelma paattyy."


main()
```

Kaikkien rivien lukeminen yhdellä käskyllä

- Tiedostosta voidaan myös lukea kaikki (jäljellä olevat) rivit metodilla `readlines`.
- Metodi palauttaa listan, joka sisältää tiedoston eri rivit merkkijonoina.
- Rivit sisältävät myös rivinvaihtomerkin.
- Seuraavan kalvon esimerkkiohjelma lukee tiedoston rivit listaan ja tulostaa ne.

Rivien lukeminen yhdellä käskyllä: koodi

```
def main():
    nimi = raw_input("Anna luettavan tiedoston nimi: ")
    try:
        lahtotiedosto = open(nimi, "r")
        rivilista = lahtotiedosto.readlines()
        lahtotiedosto.close()
        for rivi in rivilista:
            rivi = rivi.rstrip()
            print rivi
    except IOError:
        print "Virhe tiedoston", nimi, \
            "lukemisessa. Ohjelma paattyy."


main()
```

Rivien jakaminen

- Jos tiedostosta luettu rivi pitää jakaa osiin, voidaan käyttää `split`-metodia.
- Seuraava esimerkkiohjelma lukee tiedoston, joka sisältää nimiä niin, että samalla rivillä on etu- ja sukunimi toisistaan välilyönnillä erotettuna.
- Ohjelma tulostaa luetut nimet toisinpäin (sukunimi ennen etunimeä).
- Jos joku riveistä ei muodostu kahdesta osasta, ohjelma ilmoittaa virheestä, mutta jatkaa tiedoston lukemista seuraavalta riviltä.

Rivien jakaminen, koodi

```
def main():
    nimi = raw_input("Anna luettavan tiedoston nimi: ")
    try:
        lahtotiedosto = open(nimi, "r")
        rivinro = 0
        for rivi in lahtotiedosto:
            rivinro +=1
            rivi = rivi.rstrip()
            osat = rivi.split()
            if len(osat) != 2:
                print "Virhe rivilla", rivinro
            else:
                print osat[1], osat[0]
        lahtotiedosto.close()
```

Rivien jakaminen, koodi jatkuu

```
    except IOError:
        print "Virhe tiedoston", nimi, \
            "lukemisessa. Ohjelma paattyy."


main()
```

Lukujen lukeminen tiedostosta

- Kun tiedostosta luetaan lukuja, pitää luetut rivit muuttaa tyypinmuunnoksella oikeantyyppiksiksi.
- Jos riviä ei voida muuttaa luvuksi, aiheutuu `ValueError`, joka on syytä käsitellä.
- Jos samalla rivillä on useita lukuja, pitää rivi ensin jakaa. Tyypinmuunnos tehdään vasta jaon tuloksena syntyneille luvuille.
- Seuraava esimerkkiohjelma lukee tiedostosta lämpötiloja ja laskee niiden keskiarvon. Kukin lämpötila on annettu omalla rivillään. Jos tiedostossa on virhe, ohjelma ilmoittaa virheestä ja lopettaa toimintansa.

Desimaalilukuja tiedostosta, koodi

```
def main():
    nimi = raw_input("Mista tiedostosta lampotilat luetaan: ")
    summa = 0.0
    lkm = 0
    try:
        lampotiedosto = open(nimi, "r")
        for rivi in lampotiedosto:
            rivi = rivi.rstrip()
            lampotila = float(rivi)
            summa += lampotila
            lkm += 1
        lampotiedosto.close()
```

Desimaalilukuja tiedostosta, koodi jatkuu

```
        if lkm == 0:
            print "Tiedostossa ei ollut yhtaan lampotilaa."
        else:
            keskiarvo = summa / lkm
            print "Lampotilojen keskiarvo on", keskiarvo
    except IOError:
        print "Virhe tiedoston", nimi, \
            "lukemisessa. Ohjelma paattyy."
    except ValueError:
        print "Virheellinen rivi tiedostossa", nimi, \
            ". Ohjelma paattyy."


main()
```