

Ohjelmoinnin perusteet Y Python

T-106.1208

28.2.2011

Ohjelmointiprojektin vaiheet

1. Määrittely
2. Ohjelman suunnittelu (ohjelman rakenne ja ohjelman käyttämät tietorakenteet)
3. Koodaus ohjelmointikielelle
4. Testaus
5. Käyttöönotto
6. Ylläpito

Suunnittelu: mitä funktioita ja tietorakenteita ohjelmaan tulee?

- ▶ Kirjoita kuvaus ohjelman toiminnasta.
- ▶ Millaisista osatehtävistä ohjelman toiminta koostuu?
- ▶ Yleensä kutakin osatehtävää varten kirjoitetaan oma funktio.
- ▶ Aloita ohjelman tärkeimmistä osatehtävistä ja tarkenna sitten näiden toimintaa. Tällöin saattaa osoittautua tarpeelliseksi määritellä uusia osatehtäviä.
- ▶ Lisäksi on mietittävä, mitä tietoja funktio tarvitsee muulta ohjelmalta (parametrit) ja mitä tietoja se tuottaa muulle ohjelmalle (paluuarvot).
- ▶ Huomaa: tämä lähestymistapa ei sovi olio-ohjelmointiin.
- ▶ Tietorakenteet: Mieti, mitä tietoa ohjelma joutuu käsittelemään ja missä muodossa se kannattaa tallentaa. Tarvitaanko esim. merkkijonoja, listoja. sanakirjoja tms. yksittäisiä lukuja esittävien muuttujien lisäksi?

Lisää funktioiden suunnittelusta

- ▶ Tavoitteena on se, että funktio näyttää ulkopuolelle mustalta laatikolta: funktion käyttäjän tarvitsee tietää, mitä lähtötietoja funktio tarvitsee ja mitä se palauttaa, mutta ei funktion toiminnan yksityiskohtia.
- ▶ Funktion sisäinen toteutus ei saa vaikuttaa muuhun ohjelmaan.
- ▶ Funktioiden pituus pitää suunnitella sopivaksi. Yhden rivin mittaisista käskyistä kannattaa yleensä tehdä funktioita vain silloin, jos ne laskevat jonkin matemaattisen lausekkeen arvon tai tarkastavat monimutkaisemman ehdon totuusarvon. Toisaalta liian pitkät funktiot vaikeuttavat ohjelman rakenteen ymmärtämistä.
- ▶ Funktion pitäisi olla loogisesti yhtenäinen kokonaisuus.

Esimerkki: valikkopohjainen puhelinluettelo

- ▶ Laajempi puhelinluettelo-ohjelma, jossa voidaan kysyä haluttua puhelinnumeroa, lisätä uusi numeroita, muuttaa ja poistaa luettelossa jo olevia numeroita.
- ▶ Käyttäjälle tulostetaan valikko, joka kertoo mahdolliset toimenpiteet. Käyttäjä valitsee valikosta aina yhden toimenpiteen kerrallaan, kunnes hän lopettaa ohjelman suorituksen.
- ▶ Kirjoitetaan oma funktio jokaista eri toimenpidettä (numeron kysyminen, numeron lisäys, numeron muuttaminen, numeron poisto) varten.
- ▶ Lisäksi kirjoitetaan oma funktio, joka tulostaa käyttäjälle valikon ja pyytää käyttäjän valinnan. Se palauttaa käyttäjän valinnan.
- ▶ Käytettävä puhelinluettelo välitetään sitä käsitteleville funktioille parametrina.
- ▶ Pääohjelma sisältää toistokäskyn, joka kutsuu aina valikon tulostavaa funktiota ja sen jälkeen valitsee suoritettavan funktion käyttäjän valinnan mukaan.

Poikkeukset

- ▶ Ohjelmaa suoritettaessa voidaan törmätä virhetilanteisiin.
- ▶ Osa virheistä johtuu ohjelmointivirheistä, mutta osaan ohjelmoija ei voi vaikuttaa (käyttäjä antaa vääräntyyppisen syötteen, ohjelman pitäisi kirjoittaa tiedostoon, mutta kovalevytila on täynnä).
- ▶ Virhetilanteiden käsittely if-else-rakenteen avulla tekee ohjelmasta helposti sekavan.
- ▶ Python tarjoaa virhetilanteiden käsittelyyn oman mekanismin, *poikkeukset*
- ▶ Poikkeus voidaan käsitellä try-except-rakenteen avulla.

try-except-rakenne

```
try:  
    # Jono kaskyja, joista jokin tai jotkin  
    # voivat aiheuttaa poikkeuksen.  
except poikkeuksen_tyyppi:  
    # Kaskyja, jotka jotenkin selvittavat  
    # virhetilanteen, jos on aiheutunut  
    # poikkeuksen_tyyppi-tyyppinen poikkeus.
```

- ▶ Try-osassa olevia käskyjä suoritetaan normaalisti.
- ▶ Jos aiheutuu tyypin poikkeuksen_tyyppi poikkeus, hypätään välittömästi except-osaan, eikä enää palata try-osaan.
- ▶ Jos poikkeusta ei aiheudu, except-osan käskyjä ei suoriteta lainkaan.

Virheelliseen syötteeseen varautuminen

- ▶ Yleensä halutaan varautua käyttäjältä syötettä lukiessa siihen, että käyttäjä antaa virheellisen syötteen.
- ▶ Jos käyttäjän antama syöte on väärää tyyppiä (esimerkiksi kirjaimia sisältävä merkkijono, kun pitäisi olla kokonaisluku), aiheutuu `ValueError`-tyyppinen poikkeus, kun syötettä yritetään muuntaa oikean tyyppiseksi.
- ▶ Tämä poikkeus voidaan käsitellä `try-except`-rakenteessa.
- ▶ Yksinkertaisimmillaan `except`-osassa annetaan käyttäjälle selväsanainen virheilmoitus, toinen vaihtoehto on pyytää käyttäjältä uutta syötettä niin kauan, että hän antaa oikean.
- ▶ Seuraava ohjelma muuntaa käyttäjän nauloina antaman massan kilogrammoiksi. Jos käyttäjä ei anna kokonaislukua, aiheutuu poikkeus. Tällöin ohjelma antaa virheilmoituksen.

Naulamuunnos, koodi

```
def main():
    NAULAKERROIN = 0.4536
    print "Muutan nauloina annetun massan kilogrammoiksi."
    try:
        syote = raw_input("Anna massa nauloina: ")
        naulat = int(syote)
        kilot = NAULAKERROIN * naulat
        print "Massa on %.3f kg" % (kilot)
    except ValueError:
        print "Virhe: et antanut nauloja kokonaislukuna."

main()
```

Syötteen pyytäminen uudelleen

- ▶ Parempi versio ohjelmasta pyytää käyttäjältä nauvoja niin kauan, että hän antaa kokonaisluvun.
- ▶ Uutta pyyntöä ei kuitenkaan sijoiteta except-osaan, sillä käyttäjä voi antaa seuraavallakin kerralla virheellisen syötteen ja myös sen aiheuttamaan poikkeukseen halutaan varautua.
- ▶ Sen sijaan koko try-except-osa sijoitetaan toistokäskyn sisään.
- ▶ Toistokäskyn suoritusta jatketaan niin kauan, että on saatu luettua kelvollinen syöte.

Naulamuunnos: uusi koodi

```
def main():
    NAULAKERROIN = 0.4536
    print "Muutan nauloina annetun massa kilogrammoiksi."
    luku_onnistui = False
    while not luku_onnistui:
        try:
            syote = raw_input("Anna massa nauloina: ")
            naulat = int(syote)
            kilot = NAULAKERROIN * naulat
            print "Massa on %.3f kg" % (kilot)
            luku_onnistui = True
        except ValueError:
            print "Virhe: et antanut nauvoja kokonaislukuna."
            print "Yrita uudelleen!"

main()
```

Apufunktio luvun lukemiseen

- ▶ Jos samassa ohjelmassa luetaan kokonaislukuja useassa kohdassa, kannattaa yleensä kirjoittaa apufunktio kokonaisluvun lukemiseen.
- ▶ Funktio lukee ja palauttaa kokonaisluvun. Jos lukeminen ei onnistu, funktio pyytää käyttäjältä uutta kokonaislukua niin kauan, että saadaan kelvollinen kokonaisluku.
- ▶ Vastaavat apufunktiot voidaan kirjoittaa myös muuntityyppisten arvojen lukemiseen.

Kokonaisluvun lukeminen apufunktion avulla: koodi

```
def lue_kokonaisluku():  
    luku_onnistui = False  
    while not luku_onnistui:  
        try:  
            syote = raw_input()  
            luku = int(syote)  
            luku_onnistui = True  
        except ValueError:  
            print "Virheellinen kokonaisluku!"  
            print "Anna uusi!"  
    return luku
```

Kokonaisluvun lukeminen apufunktion avulla: koodi jatkuu

```
def main():  
    NAULAKERROIN = 0.4536  
    print "Muutan nauloina annetun massa kilogrammoiksi."  
    print "Anna massa nauloina."  
    naulat = lue_kokonaisluku()  
    kilot = NAULAKERROIN * naulat  
    print "Massa on %.3f kg" % (kilot)  
  
main()
```

Huomatuksia poikkeuksista

- ▶ try-except-rakenne voi sisältää useita except-osia erityyppisiä poikkeuksia varten. Tällöin poikkeuksen sattuessa siirrytään ensimmäiseen except-osaan, jonka poikkeuksen tyyppi vastaa aiheutunutta poikkeusta.
- ▶ Ohjelmoija voi myös itse aiheuttaa poikkeuksen (virhetilanteen sattuessa) raise-käskyllä. Sitä ei kuitenkaan käsitellä tällä kurssilla.

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 merkkijonon ""
- ▶ 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 "Nimeä", etsittava_nimi, "ei löytynyt."
```

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()
```