Ohjelmoinnin perusteet Y Python

T-106.1208

14.4.2010

Mitä tämän kurssin jälkeen?

- T-106.1223 Tietorakenteet ja algoritmit Y (5 op)
  - Tietorakenteita, esim. linkitetyt listat, hakupuut, verkot
  - Algoritmeja, esim. järjestäminen, haku
  - Algoritmin tehokkuuden arviointi
- AS-0.1103 C-ohjelmoinnin peruskurssi (6 op)
  - Ohjelmointia C-kielellä.
  - Oletetaan toisen ohjelmointikielen aikaisempi tuntemus.
- Jos haluat opiskella ohjelmointia enemmän, esitiedoksi vaadita T-106.1203 Ohjelmoinnin perusteet L (Java)

Tentti

- Ensimmäinen tenttimahdollisuus on ke 12.5. klo 12:30–16:30 päärakennuksessa.
- Tämän jälkeen on vielä neljä muuta mahdollisuutta:
  - Kesätentti elokuussa (todennäköisesti 16.8. klo 10:00–14:00, tarkista aika myöhemmin).
  - Kaksi lauantaitenttiä syksyllä 2010
  - Kevätlukukauden 2011 ensimmäinen tentti (todennäköisesti helmi- tai maaliskuussa).
- Muista ilmoittautua tenttiin WebOodissa viimeistään viikko etukäteen! (Kesätenttiin jo aikaisemmin.)

Tenttitehtävätyyppejä

- Koodin ymmärtämistä testaavia tehtäviä, esimerkiksi:
  - Mitä annettu ohjelma tulostaa?
  - Mikä arvo ohjelmassa käytetyllä muuttujalla/muuttujilla on oltava, jotta ohjelma tulostaisi X?
  - Mitä annettu funktio/ohjelma tekee?
  - Mitä virheitä annetussa ohjelmassa on?
- Selitystehtäviä
  - Esimerkiksi käsitteiden lyhyitä selityksiä.
  - Näitä ei ole joka tentissä ja selitystehtävien osuus tentin maksimipistemäärästä on korkeintaan 20 prosenttia.

Tenttitehtävätyyppejä, jatkoa

- Omien ohjelmien ja funktioiden kirjoittaminen, esimerkiksi
  - Kirjoita ohjelma, joka tekee vaaditun asian.
  - Kirjoita funktio, joka saa parametrina tehtävässä mainitut arvot ja palauttaa vaaditun arvon.
  - Olkoon olemassa funktio X, jonka parametrit ja paluuarvo on kerrottu tehtävässä. Kirjoita ohjelma, joka tekee vaaditun asian käyttämällä hyväksi funktiota X.
- Ratkaisuja kirjoittaessa voi tarvita esimerkiksi seuraavia asioita:
  - If-käsky
  - Toistokäsky
  - Listojen tai merkkijonojen käsittely
  - Funktion kirjoittaminen (sisältää parametrien ja paluuarvojen käytön)
  - Tiedostosta lukeminen tai tiedostoon kirjoittaminen
  - Poikkeusten käsittely try–except-rakenteen avulla.

Tenttitehtävätyyppejä, jatkoa

- Olio-ohjelmointitehtävä
  - Vastaa vaativuustasoltaan yhteensä harjoitustehtäväkierroksen 9 tehtäviä 1 ja 2. (Kirjoita yksinkertainen luokka ja pääohjelma, joka luo luokan olioita ja kutsuu niille luokan metodeita.)
  - Tällainen tehtävä on joka tentissä ja sen pistemäärä on 25 prosenttia tentin maksimipisteistä.
  - Käytännössä tentistä on vaikea saada arvosanaa 3 tai parempaa, jos ei saa lainkaan pisteitä olio-ohjelmointitehtävästä. (Arvosanan 3 raja on noin 70 prosenttia tentin maksimipistemäärästä.)

Tenttivaatimuksista

- Kaikki opetusmonisteessa kerrotut asiat kuuluvat tenttivaatimuksiin, paitsi seuraavat asiat:
  - Tulostuksen muotoilu (tenttitehtävissä tulostusta ei tarvitse muotoilla).
  - Monikko
  - Sanakirja
  - Listoja käsittelevistä funktioista ja metodeista seuraavia ei tarvitse osata ulkoa (jos niitä tarvitaan, ne on annettu tehtävässä): `index`, `insert`, `remove`, `sort`, `reverse`. Myöskään sellaisia Pythonin valmiita listoja käsitteleviä funktioita ja metodeita ei tarvitse osata, joita ei ole esitelty lainkaan opetusmonisteessa.
  - Merkkijonoja käsittelevistä funktioista ja metodeista seuraavia ei tarvitse osata ulkoa (jos niitä tarvitaan, ne on annettu tehtävässä): `index`, `lower`, `upper`, `strip`. Myöskään sellaisia Pythonin valmiita merkkijonoja käsitteleviä funktioita ja metodeita ei tarvitse osata, joita ei ole esitelty lainkaan opetusmonisteessa.
- Opetusmonisteen lisäksi tenttivaatimuksiin kuuluu 17.2.2010 pidetyn luennon luentokalvot otsikosta Arvot ja viittaukset lähtien.

Tenttiin valmistautumisesta ja vastaamisesta

- Harjoittele tenttiin kirjoittamalla itse pieniä Python-ohjelmia.
- Harjoittele myös ohjelmien kirjoittamista ilman mallia ja paperille.
- Ratkaisujen pienistä kirjoitusvirheistä ei yleensä sakoteta, jos niillä ei ole periaatteellista merkitystä ja muusta ratkaisusta näkyy, että vastaava asia on osattu.
- Poikkeuksena on kuitenkin sisennykset: niiden on oltava selvästi merkitty ja oikein, koska sisennyksillä on koodin toimintaan ratkaiseva merkitys. Jos sisennykset on epäselvästi merkitty tai puuttuvat, se vähentää pisteitä selvästi. Käytä kahden ruudun levyisiä sisennyksiä.
- Jos joku tehtävä tuntuu liian vaikealta, älä jää liian pitkäksi aikaa tekemään sitä, vaan tee ensin ne tehtävät, jotka osaat.

Esimerkki koodin ymmärtämistehtävästä

- Mitä alla esitetty funktio `mysteeri` tekee? Älä selitä funktion toimintaa käsky käskyltä, vaan selitä 1-2 lauseella, mikä on funktion tarkoitus. Voit olettaa, että funktiolle annetaan parametrina kokonaislukuja sisältävä lista.

```
def mysteeri3(luvut):
    i = 2
    while i < len(luvut):
        if luvut[i] != luvut[i-1] + luvut[i-2]:
            return False
        i = i + 1
    return True
```

Vastaus edellisen kalvon tehtävään

- Funktio tutkii, onko sille parametrina annetussa listassa jokainen luku (kahta ensimmäistä lukuunottamatta) kahden edellisen luvun summa. Funktio palauttaa arvon True, jos näin on. Muussa tapauksessa funktio palauttaa arvon False.

Kertausta: if / elif

- Miten seuraavat kaksi koodia poikkeavat toisistaan?

```
if pituus < 160:
    print "Olet lyhyt."
if pituus < 185:
    print "Olet keskimittainen."
else:
    print "Olet pitka."
```

```
if pituus < 160:
    print "Olet lyhyt"
elif pituus < 185:
    print "Olet keskimittainen."
else:
    print "Olet pitka."
```

Kertausta: while-käsky

- Yleinen muoto

```
while ehto:
    kasky
```

- Muista:
  - Mahdolliset alustukset.
  - Joku käsky (toistettavan silmukan sisällä), joka saa ehdon lopulta epätodeksi.

Esimerkki while-käskystä

```
def main():
    HELLERAJA = 25.0
    print "Anna lampotiloja, lopeta -300:lla."
    hellepaivien_lkm = 0
    rivi = raw_input("Anna ensimmainen lampotila.\n")
    lampotila = float(rivi)
    while lampotila > -300.0:
        if lampotila >= HELLERAJA:
            hellepaivien_lkm += 1
        rivi = raw_input("Anna seuraava lampotila.\n")
        lampotila = float(rivi)
    print "Hellepaivia oli", hellepaivien_lkm, "kpl."


main()
```

For-käsky

- Yleinen muoto:

```
for muuttuja in rakenne:
    kasky
```

- Rakenne voi olla esimerkiksi lista, merkkijono, sanakirja tai tiedosto.
- `muuttuja` saa arvokseen vuorotellen kunkin rakenteen alkion tms.

For-käsky ja range-funktio, esimerkki

```
def main():
    rivi = raw_input("Anna lampotilojen maara.\n")
    lkm = int(rivi)
    summa = 0.0
    print "Anna lampotilat."
    for i in range(lkm):
        rivi = raw_input()
        lampotila = float(rivi)
        summa += lampotila
    if lkm > 0:
        keskiarvo = summa / lkm
        print "Keskiarvo on %.2f" % (keskiarvo)
    else:
        print "Keskiarvoa ei voi laskea."

main()
```

Funktiot

- Funktio on ohjelmakoodin osa, jolle on annettu oma nimi.
- Kun ohjelma jaetaan funktioihin, sen rakenne selkiytyy.
- Parametrien avulla välitetään tietoa funktioon sen ulkopuolelta. Näin samaa funktioita voidaan käyttää useita kertoja eri lähtöarvoilla.
- Funktio voi välittää tietoa laskemistaan arvoista ulospäin paluuarvojen avulla.
- Funktiota kutsutaan sen nimen avulla. Kutsussa kerrotaan myös, mitkä arvot annetaan funktiossa käytettäville parametreille.

Funktioesimerkki

```
def laske_kokonaispalkka(tunnit, tuntipalkka):
    if tunnit <= 0:
        palkka = 0.0
    elif tunnit <= 40:
        palkka = tunnit * tuntipalkka
    else:
        palkka = 40 * tuntipalkka + \
                 1.5 * (tunnit - 40) * tuntipalkka
    return palkka
```

Funktioesimerkki jatkuu

```
def main():
    rivi = raw_input("Anna tuntipalkka.\n")
    tuntip = float(rivi)
    rivi = raw_input("Anna tyontekijan 1 tunnit.\n")
    tuntimaara1 = int(rivi)
    rivi = raw_input("Anna tyontekijan 2 tunnit.\n")
    tuntimaara2 = int(rivi)
    palkka1 = laske_kokonaispalkka(tuntimaara1, tuntip)
    palkka2 = laske_kokonaispalkka(tuntimaara2, tuntip)
    print "Tyontekijan 1 palkka on %.2f euroa." % (palkka1)
    print "Tyontekijan 2 palkka on %.2f euroa." % (palkka2)


main()
```

Totuusarvon palauttavat funktiot

- Funktio voi myös palauttaa totuusarvon `True` tai `False`
- Seuraava esimerkkiohjelma laskee käyttäjän iän tämän syntymävuoden perusteella.
- Ohjelmaan on lisätty funktio `onko_kelvollinen`, joka tutkii, onko sille parametrina annettu syntymävuosi hyväksytyllä välillä 1890–2010.
- Funkton palauttamaa arvoa voitaisiin käyttää esimerkiksi if-käskyn ehdossa seuraavasti:

```
if onko_kelvollinen(syntymavuosi) == True:
```

- Vertailu arvoon `True` on kuitenkin tarpeeton, koska funktio itsessään palauttaa arvon `True` tai `False`. Sen vuoksi ehto voidaan kirjoittaa lyhyemmin:

```
if onko_kelvollinen(syntymavuosi):
```

län laskeminen, koodi

```
NYKYINEN_VUOSI = 2010

def onko_kelvollinen(vuosi):
    ALARAJA = 1890
    if vuosi < ALARAJA or vuosi > NYKYINEN_VUOSI:
        return False
    else:
        return True
```

län laskeminen, koodi jatkuu

```
def main():
    print "Ohjelma laskee ikasi."
    rivi = raw_input("Anna syntymavuotesi.\n")
    syntymavuosi = int(rivi)
    if onko_kelvollinen(syntymavuosi):
        ika = NYKYINEN_VUOSI - syntymavuosi
        print "Ikasi on", ika, "vuotta."
    else:
        print "Virhe syntymavuodessa."


main()
```

Kertausta: listat

- Tyhjä uusi lista luodaan kirjoittamalla esimerkiksi

  `lampotilat = []`

  (jolloin `lampotilat` on luotuun listaan viittaavan muuttujan nimi).
- Listan loppuun voi lisätä uuden arvon kirjoittamalla

  `lampotilat.append(arvo)`
- Jos listassa on jo vähintään `i+1` alkiota, voi indeksille `i` sijoittaa uuden arvon kirjoittamalla

  `lampotilat[i] = arvo`

  Tällöin indeksillä `i` ollut vanha arvo häviää.
- Indeksit ovat kokonaislukuja ja listan ensimmäisen alkion indeksi on aina 0.
- Listan yksittäistä alkiota voi käyttää esimerkiksi tulostuskäskyissä tai lausekkeissa:

```
print "5. lampotila", lampotilat[4]
summa = summa + lampotilat[4]
```

Kertausta: listan läpikäynti

- While-käskyn avulla:

```
summa = 0
i = 0
while i < len(lampotilat):
    summa += lampotilat[i]
    i += 1
```

- For-käskyn avulla

```
summa = 0.0
for astemaara in lampotilat:
    summa += astemaara
```

Muuttuja `astemaara` saa arvokseen vuorotellen jokaisen listan alkion.

Listaesimerkki

- Esimerkkinä funktio, joka käy läpi parametrina annetun listan ja palauttaa uuden listan, joka sisältää alkuperäisen listan ne alkiot, jotka ovat annettujen ala- ja ylärajan välissä.

```
def rajojen_sisalla(lukulista, alaraja, ylaraja):
    tuloslista = []
    for alkio in lukulista:
        if alaraja <= alkio <= ylaraja:
            tuloslista.append(alkio)
    return tuloslista

def main():
    lista1 = [12, 4, 8, 0, 16, 10]
    lista2 = rajojen_sisalla(lista1, 6, 13)
    print lista2

main()
```

Listaesimerkki 2

- Vastaava funktio kirjoitettuna while-käskyn ja alkioiden indeksoinnin avulla:

```
def rajojen_sisalla(lukulista, alaraja, ylaraja):
    tuloslista = []
    i = 0
    while i < len(lukulista):
        if alaraja <= lukulista[i] <= ylaraja:
            tuloslista.append(lukulista[i])
        i += 1
    return tuloslista
```

Kertausta: 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.
- Tentissä pitää muistaa ulkoa poikkeustyypit `ValueError` ja `IOError`. Muut poikkeukset kerrotaan, jos niitä pitää käyttää.

Esimerkki try–except-rakenteen käytöstä

```
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()
```

Toinen esimerkki: apufunktio kokonaisluvun lukemiseen

```
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
```

Kertausta: tiedostosta lukeminen

- Aluksi käsiteltävä tiedosto pitää avata:

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

- Sen jälkeen tiedostosta voi lukea rivin kerrallaan:

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

  Luettu rivi sisältää myös sen lopussa olevan rivinvaihtomerkin.

- Jos tiedosto on jo luettu loppuun ja kutsutaan `readline`-metodia, se palauttaa arvona tyhjän merkkijonon `""`

- Kun tiedoston lukeminen päättyy, tiedosto pitää sulkea:

  ```
  tiedostomuuttuja.close()
  ```

Kertaus jatkuu

- Toinen vaihtoehto on käydä kaikki tiedoston rivit läpi järjestyksessä for-käskyllä:

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

- Tiedostosta voidaan myös lukea kaikki (jäljellä olevat) rivit metodilla `readlines`.
- Metodi palauttaa listan, joka sisältää tiedoston eri rivit merkkijonoina.
- Rivit sisältävät rivinvaihtomerkin.

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()
```

Kertausta: tiedostoon kirjoittaminen

- Tiedosto on avattava, mutta käsitettelytapa on eri kuin tiedostoon kirjoitettaessa (toinen vaihtoehto `"a"`).

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

- Tiedostoon voi tulostaa rivin `write`-metodilla. Se ei lisää rivinvaihtomerkkiä, vaan merkki on lisättävä kirjoitettavaan riviin.

```
tiedostomuuttuja.write("Kirjoitettava rivi\n")
```

- Metodilla `write` voi tulostaa tiedostoon vain merkkijonoja. Esimerkiksi luvut pitää muuttaa ennen tulostamista merkkijonoiksi joko `str`-tyypinmuunnoksella tai käyttämällä tulostuksen muotoilua.

Esimerkki: lukuja tiedostoon

```
import math

def main():
    print "Ohjelma laskee ympyroiden pinta-aloja ja"
    print "tallentaa ne tiedostoon."
    nimi = raw_input("Anna kirjoitettavan tiedoston nimi: ")
    try:
        tulostiedosto = open(nimi, "w")
        tulostiedosto.write("sade    pinta-ala\n")
        print "Anna sateet, lopeta negatiivisella."
        rivi = raw_input()
        sade = float(rivi)
        while sade >= 0:
            pinta_ala = math.pi * sade * sade
            tulostiedosto.write("%-7.2f %-10.2f\n" % \
                (sade, pinta_ala))
```

Esimerkki: lukuja tiedostoon (jatkuu)

```
            rivi = raw_input()
            sade = float(rivi)
        tulostiedosto.close()
        print "Tulokset on kirjoitettu tiedostoon", nimi
    except IOError:
        print "Virhe tiedoston", nimi, "kirjoittamisessa."
    except ValueError:
        print "Ei ollut luku. Tiedoston kirjoitus saattoi"
        print "epaonnistua."


main()
```