

Ohjelmoinnin perusteet Y Python

T-106.1208

14.3.2011

Mitä oliot ovat?

- ▶ Esimerkki: halutaan laatia ohjelma, joka käsittelee erään ohjelmointikurssin opiskelijoita. Kurssilla on noin 100 opiskelijaa.
- ▶ Jokaisesta opiskelijasta halutaan ohjelman käyttöön ainakin nimi, opiskelijanumero, tenttiarvosana ja harjoitusarvosana.
- ▶ Ongelma: miten opiskelijoiden tietoja esitetään ja käsitellään ohjelmassa?
- ▶ 1. ratkaisu (huono): otetaan käyttöön 400 muuttujaa eri arvoja varten.
- ▶ 2. ratkaisu (huono): otetaan käyttöön 4 eri listaa: nimet, opiskelijanumerot, tenttiarvosanat ja harjoitusarvosanat. Jokaisessa listassa on 100 alkia ja saman opiskelijan tiedot ovat listassa aina samalla indeksillä.

Mitä oliot ovat (jatkoa)

- ▶ 3. ratkaisu (parempi): tehdään yhden opiskelijan tiedoista lista, jossa on neljä alkiota. Kurssin kaikista opiskelijoista muodostetaan lista, jonka alkiot ovat listoja.
- ▶ Olioita käyttävä ratkaisu: tehdään jokaista oikeaa opiskelijaa kohti ohjelmaan yksi `Opiskelija`-olio. Luokassa `Opiskelija` kerrotaan, millaisia `Opiskelija`-oliot ovat ja millaisia toimintoja niille voi tehdä. Kurssin kaikkia opiskelijoita esitetään `Opiskelija`-olioita sisältävänä listana.
- ▶ Olioita käyttävän ratkaisun etuja:
 - ▶ Yhden opiskelijan tietoja käsitellään yhtenä kokonaisuutena (yksi olio).
 - ▶ Opiskelijan eri tiedot (esimerkiksi nimi ja opiskelijanumero) voidaan nimetä selvästi.
 - ▶ Samalla kun määritellään, millainen olio on, määritellään myös sille mahdolliset toimenpiteet (esimerkiksi tenttiarvosanan muuttaminen, harjoitusarvosanan muuttaminen, kokonaisarvosanan laskeminen).
- ▶ Olio-ohjelmointi tarjoaa myös monia sellaisia mahdollisuuksia ja etuja, joita ei käsitellä lainkaan tällä kurssilla.

Olioista

- ▶ Olioilla on kenttiä (ilmentymämuuttujia / attribuutteja). Esimerkiksi Opiskelija-oliolla voisi olla kentät nimi, opiskelijanumero, harjoitusarvosana ja tenttiarvosana.
- ▶ Kenttien avulla kuvataan olion ominaisuuksia. Esimerkiksi kentän nimi arvon avulla voidaan kertoa, mikä on jonkun Opiskelija-olion nimi.
- ▶ Jokaisella oliolla on omat kenttien arvot. Muutos yhden olion kentän arvossa ei vaikuta toisen olion kenttien arvoihin.

nimi = "Teemu Teekkari"
opiskelijanumero = "67558U"
tenttiarvosana = 3
harjoitusarvosana = 5

nimi = "Oili Opiskelija"
opiskelijanumero = "72111R"
tenttiarvosana = 4
harjoitusarvosana = 4

nimi = "Iiro Ikiteekkari"
opiskelijanumero = "18999T"
tenttiarvosana = 2
harjoitusarvosana = 3

Olion kenttien arvon muuttaminen

- ▶ Periaatteessa olion kenttien arvoihin voi viitata pistenotaation avulla.
- ▶ Oletetaan, että on luotu yksi `Opiskelija`-olio ja pantu muuttuja `kurssilainen1`-viittaamaan siihen. Tällöin olion tietoja voi periaatteessa käsitellä pistenotaation avulla esimerkiksi seuraavasti:

```
kurssilainen1.nimi = "Niilo Lahti"  
kurssilainen1.harjoitusarvosana = 5  
print kurssilainen1.harjoitusarvosana
```
- ▶ Tämä tapa ei ole kuitenkaan suositeltava (syy selviää myöhemmin), vaan yleensä olion kenttiä käsitellään luokan metodien avulla. Pistenotaation ymmärtäminen kuitenkin helpottaa metodien määrittelyjen ymmärtämistä.

Luokka ja olio

- ▶ Luokassa määritellään, millaisia luokan oliot ovat ja mitä operaatioita olioille voidaan tehdä.
- ▶ Jos ohjelmassa halutaan käsitellä `Opiskelija`-olioita, kirjoitetaan luokka `Opiskelija`.
- ▶ Luokkaan kirjoitetaan metodit, jotka määrittelevät `Opiskelija`-olioille mahdolliset operaatiot.
- ▶ Luokka on kuin koneen piirustukset ja olio kuin niiden mukaan tehty kone.
- ▶ Yksien piirustusten perusteella voidaan rakentaa monta konetta. Vastaavasti yhdestä luokasta voidaan tehdä monta oliota.
- ▶ Luokan määrittely aloitetaan luokan otsikolla:
`class Opiskelija:`

Olioiden luonti

- ▶ Ensimmäiseksi luokkaan kirjoitetaan yleensä metodi `__init__`, joka määrittelee, mitä tapahtuu, kun luodaan uusi luokan olio. Tyypillisesti metodissa asetetaan alkuarvoja olion kentille.

```
def __init__(self, annettu_nimi, numero):  
    self.__nimi = annettu_nimi  
    self.__opiskelijanumero = numero  
    self.__tenttiarvosana = 0  
    self.__harjoitusarvosana = 0
```

Metodin ensimmäinen parametri `self` tarkoittaa sitä oliota, jota ollaan juuri luomassa.

- ▶ Kun metodi on määritelty, uusia `Opiskelija`-olioita voidaan luoda (yleensä luokan ulkopuolella) seuraavaan tapaan:

```
kurssilainen1 = Opiskelija("Matti Virta", "11223U")  
kurssilainen2 = Opiskelija("Oili Lahti", "77445X")
```

Muiden metodien määrittely ja kutsuminen

- ▶ Määritellään metodi, jonka avulla voidaan vaihtaa `Opiskelija`-olion tenttiarvosana. Uusi arvosana annetaan parametrina. Metodi tarkistaa, että uusi arvosana on sallitulla välillä 0–5.

```
def muuta_tenttiarvosana(self, arvosana):  
    if 0 <= arvosana <= 5:  
        self.__tenttiarvosana = arvosana
```

Parametri `self` tarkoittaa sitä `Opiskelija`-oliota, jolle metodia kutsutaan.

- ▶ Jos on aikaisemmin luotu `Opiskelija`-olio ja pantu muuttuja `kurssilainen1` viittaamaan tähän olioon, voidaan tälle oliolle kutsua yllä määriteltyä metodia kirjoittamalla

```
kurssilainen1.muuta_tenttiarvosana(4)
```

Arvon palauttava metodi

- ▶ Määritellään seuraavaksi metodi, jonka avulla voidaan laskea jonkin Opiskelija-olion kokonaisarvosana.

```
def laske_kokonaisarvosana(self):
    if self.__tenttiarvosana == 0 or \
        self.__harjoitusarvosana == 0:
        arvosana = 0
    else:
        arvosana = (self.__tenttiarvosana +
                    self.__harjoitusarvosana + 1) / 2
    return arvosana
```

- ▶ Esimerkki metodin kutsumisesta (luokan ulkopuolella):

```
arvos = kurssilainen1.laske_kokonaisarvosana()
print "Kokonaisarvosana on", arvos
```

Metodit kenttien arvojen selvittämiseen

- ▶ Opiskelija-olion kenttiin ei pääse suoraan käsiksi luokan ulkopuolelta pistenotaation avulla. Sen sijaan määritellään metodit, joiden avulla voidaan kysyä kenttien arvoa, esimerkiksi

```
def kerro_nimi(self):  
    return self.__nimi  
  
def kerro_tenttiarvosana(self):  
    return self.__tenttiarvosana
```

Kahdelle muulle kentälle määritellään vastaavat metodit.

- ▶ Esimerkki metodin kutsumisesta:

```
print "Nimi on", kurssilainen1.kerro_nimi()
```

Opiskelija-luokka kokonaan

```
class Opiskelija:

    def __init__(self, annettu_nimi, numero):
        self.__nimi = annettu_nimi
        self.__opiskelijanumero = numero
        self.__tenttiarvosana = 0
        self.__harjoitusarvosana = 0

    def kerro_nimi(self):
        return self.__nimi

    def kerro_opiskelijanumero(self):
        return self.__opiskelijanumero
```

Opiskelija-luokka jatkuu

```
def kerro_tenttiarvosana(self):  
    return self.__tenttiarvosana  
  
def kerro_harjoitusarvosana(self):  
    return self.__harjoitusarvosana  
  
def muuta_tenttiarvosana(self, arvosana):  
    if 0 <= arvosana <= 5:  
        self.__tenttiarvosana = arvosana  
  
def muuta_harjoitusarvosana(self, arvosana):  
    if 0 <= arvosana <= 5:  
        self.__harjoitusarvosana = arvosana
```

Opiskeliija-luokka jatkuu

```
def laske_kokonaisarvosana(self):
    if self.__tenttiarvosana == 0 or \
        self.__harjoitusarvosana == 0:
        arvosana = 0
    else:
        arvosana = (self.__tenttiarvosana +
                    self.__harjoitusarvosana + 1) / 2
    return arvosana
```

Esimerkki luokkaa käyttävästä pääohjelmasta

- ▶ Seuraavalla kalvolla on esimerkkiohjelma, joka pyytää kahden opiskelijan tiedot ja luo heitä vastaavat `Opiskelija`-oliot.
- ▶ Ohjelma on kirjoitettu luokan `Opiskelija` ulkopuolelle.
- ▶ Opiskelijoiden arvosanat annetaan kokonaislukuina. Kokonaisluvun lukemista varten on määritetty apufunktio, joka käsittelee mahdolliset virheelliset syötteen.
- ▶ Muu ohjelma on selvyuden vuoksi kirjoitettu pääohjelmaan, vaikka olisi parempaa tyyliä jakaa se useampaan funktioon.
- ▶ Jos haluttaisiin käsitellä kurssin kaikkia opiskelijoita, tehtäisiin lista, johon kerättäisiin `Opiskelija`-olioita. Tästä tulee esimerkkejä vasta myöhemmin.

Opiskelija-olioita käyttävä ohjelma, 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
```

Opiskelija-olioita käyttävä ohjelma, koodi jatkuu

```
def main():  
    nimi1 = raw_input("Anna 1. opiskelijan nimi: ")  
    op_nro1 = raw_input("Anna 1. opiskelijan numero: ")  
    kurssilainen1 = Opiskelija(nimi1, op_nro1)  
    nimi2 = raw_input("Anna 2. opiskelijan nimi: ")  
    op_nro2 = raw_input("Anna 2. opiskelijan numero: ")  
    kurssilainen2 = Opiskelija(nimi2, op_nro2)  
  
    print "Anna 1. opiskelijan tenttiarvosana."  
    tentti1 = lue_kokonaisluku()  
    kurssilainen1.muuta_tenttiarvosana(tentti1)  
    print "Anna 1. opiskelijan harjoitusarvosana."  
    harjoitus1 = lue_kokonaisluku()  
    kurssilainen1.muuta_harjoitusarvosana(harjoitus1)
```

Opiskelija-olioita käyttävä ohjelma, koodi jatkuu

```
print "Anna 2. opiskelijan tenttiarvosana."  
tentti2 = lue_kokonaisluku()  
kurssilainen2.muuta_tenttiarvosana(tentti2)  
print "Anna 2. opiskelijan harjoitusarvosana."  
harjoitus2 = lue_kokonaisluku()  
kurssilainen2.muuta_harjoitusarvosana(harjoitus2)
```

```
print "1. opiskelijan tiedot:"  
print kurssilainen1.kerro_opiskelijanumero(),  
print kurssilainen1.kerro_nimi()  
print "Tenttiarvosana:", \  
    kurssilainen1.kerro_tenttiarvosana()  
print "Harjoitusarvosana:", \  
    kurssilainen1.kerro_harjoitusarvosana()  
print "Kurssiarvosana:", \  
    kurssilainen1.laske_kokonaisarvosana()
```

Opiskelija-olioita käyttävä ohjelma, koodi jatkuu

```
print "2. opiskelijan tiedot:"  
print kurssilainen2.kerro_opiskelijanumero(),  
print kurssilainen2.kerro_nimi()  
print "Tenttiarvosana:", \  
    kurssilainen2.kerro_tenttiarvosana()  
print "Harjoitusarvosana:", \  
    kurssilainen2.kerro_harjoitusarvosana()  
print "Kurssiarvosana:", \  
    kurssilainen2.laske_kokonaisarvosana()
```

```
main()
```

Merkkijonoesitys oliosta

- ▶ Ohjelmissa halutaan hyvin usein tulostaa jonkin olion kaikkien kenttien arvot.
- ▶ Tämä voidaan tehdä käyttämällä apuna kenttien arvot palauttavia metodeita, kuten edellisessä esimerkissä tehtiin.
- ▶ Tulostaminen kuitenkin helpottuu, jos luokkaan määritellään metodi, joka tekee oliosta merkkijonoesityksen. Metodi siis palauttaa merkkijonon, joka sisältää olion kenttien arvot.
- ▶ Tämän metodin nimeksi annetaan `__str__`. Kun metodi on nimetty näin, olion tiedot voi tulostaa (esimerkiksi pääohjelmassa) käyttämällä vain suoraan olioon viittaavan muuttujan nimeä, esimerkiksi

```
print kurssilainen1
```

Merkkijonoesitys Opiskelija-olioista

- Kirjoitetaan Opiskelija-luokkaan metodi `__str__`.

```
def __str__(self):  
    miono = self.__nimi + ", " + \  
            self.__opiskelijanumero + \  
            ", tenttiarvosana: " + \  
            str(self.__tenttiarvosana) + \  
            ", harjoitusarvosana: " + \  
            str(self.__harjoitusarvosana)  
    return miono
```

- Nyt opiskelijoiden tiedot voidaan tulostaa selvästi helpommin (koodi seuraavalla kalvolla)

Opiskelijoiden tietojen tulostus, koodi

```
print "1. opiskelijan tiedot:"  
print kurssilainen1  
print "Kurssiarvosana:", \  
    kurssilainen1.laske_kokonaisarvosana()  
  
print "2. opiskelijan tiedot:"  
print kurssilainen2  
print "Kurssiarvosana:", \  
    kurssilainen2.laske_kokonaisarvosana()
```

Luokka ja pääohjelma eri moduuleissa

- ▶ Käytännössä käytetään usein ohjelmia, jotka koostuvat useista eri luokista.
- ▶ Tällöin on monesti selvintä kirjoittaa kukin luokka omaan moduuliinsa.
- ▶ Näin samaa luokkaa voidaan helposti käyttää osana eri ohjelmia.
- ▶ Jos luokka `Opiskelija` ja sen olioita käyttävä pääohjelma (tai muu ohjelma) kirjoitetaan eri moduuleihin, pitää pääohjelmamoduulin alkuun kirjoittaa (jos `Opiskelija`-luokka on tallennettu tiedostoon `opiskelija.py`)

```
import opiskelija
```

- ▶ Lisäksi `Opiskelija`-olioita luodessa pitää luokan nimen edessä käyttää moduulin nimeä (muuta muutoksia ei tarvita):

```
kurssilainen1 = opiskelija.Opiskelija(nimi1, op_nro1)
kurssilainen2 = opiskelija.Opiskelija(nimi2, op_nro2)
```

Toinen esimerkki: luokka Vesisailio

- ▶ Määritellään luokka vesisäiliön kuvaamiseen.
- ▶ Säiliöön voi lisätä vettä ja sieltä voi poistaa vettä.
- ▶ Säiliöllä on kuitenkin koko, eikä säiliöön voi lisätä vettä enempää kuin siihen mahtuu. Säiliöstä ei voi myöskään ottaa vettä enempää kuin siellä on.
- ▶ Vettä lisäävät ja sitä poistavat metodit palauttavat lisätyn tai poistetun veden määrän.
- ▶ Lähes samanlaista rakennetta voi käyttää esimerkiksi erilaisissa varastosovelluksissa. Yhden tuotteen varastotilannetta kuvataan Vesisailio-luokkaa muistuttavalla luokalla.

Luokka Vesisailio, koodi

```
class Vesisailio:

    def __init__(self, annettu_koko):
        if annettu_koko >= 0.0:
            self.__koko = annettu_koko
        else:
            self.__koko = 0.0
        self.__maara = 0.0

    def kerro_koko(self):
        return self.__koko

    def kerro_maara(self):
        return self.__maara
```

Luokka Vesisailio, koodi jatkuu

```
def lisaa(self, paljonko):  
    if paljonko > 0.0:  
        mahtuu = self.__koko - self.__maara  
        if paljonko <= mahtuu:  
            self.__maara += paljonko  
            return paljonko  
        else:  
            self.__maara = self.__koko  
            return mahtuu  
    else:  
        return 0.0
```

Luokka Vesisailio, koodi jatkuu

```
def poista(self, paljonko):
    if paljonko > 0.0:
        if paljonko > self.__maara:
            poistetaan = self.__maara
            self.__maara = 0.0
            return poistetaan
        else:
            self.__maara -= paljonko
            return paljonko
    else:
        return 0.0

def __str__(self):
    miono = "Sailio: vetta " + str(self.__maara) + \
            " / " + str(self.__koko) + " l"
    return miono
```

Vesisailio-olioita käsittelevä pääohjelma

- ▶ Seuraavilla kalvoilla on esimerkkiohjelma, joka luo kaksi Vesisailio-oliota sekä lisää ja poistaa niistä vettä.
- ▶ Ohjelma tulostaa lisätyt ja poistetut määrät sekä säiliöiden tilan ohjelman lopussa.
- ▶ Desimaalilukujen lukemista varten on kirjoitettu oma apuohjelma, joka myös käsittelee virheelliset syötteet.
- ▶ Esimerkissä pääohjelma on kirjoitettu eri moduuliin kuin itse luokka.

Vesisailio-olioita käyttävä pääohjelma, koodi

```
import vesisailio

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

Vesisailio-olioita käyttävä pääohjelma, koodi jatkuu

```
def main():  
    print "Anna ensimmäisen sailion koko."  
    koko1 = lue_desimaaliluku()  
    sailio1 = vesisailio.Vesisailio(koko1)  
    print "Anna toisen sailion koko."  
    koko2 = lue_desimaaliluku()  
    sailio2 = vesisailio.Vesisailio(koko2)  
  
    print "Lisataan vettä 1. sailioon."  
    print "Anna lisattava maara."  
    lisays = lue_desimaaliluku()  
    lisattiin = sailio1.lisaa(lisays)  
    print "Sailioon lisattiin %.2f l." % (lisattiin)
```

Vesisailio-olioita käyttävä pääohjelma, koodi jatkuu

```
print "Lisataan vettä 2. sailioon."  
print "Anna lisattava maara."  
lisays = lue_desimaaliluku()  
lisattiin = sailio2.lisaa(lisays)  
print "Sailioon lisattiin %.2f l." % (lisattiin)  
  
print "Poistetaan vettä 1. sailiosta."  
print "Anna poistettava maara."  
poisto = lue_desimaaliluku()  
poistettiin = sailio1.poista(poisto)  
print "Sailiosta poistettiin %.2f l." % (poistettiin)
```

Vesisailio-olioita käyttävä pääohjelma, koodi jatkuu

```
print "Poistetaan vettä 2. sailiosta."  
print "Anna poistettava maara."  
poisto = lue_desimaaliluku()  
poistettiin = sailio2.poista(poisto)  
print "Sailiosta poistettiin %.2f l." % (poistettiin)  
  
print "Ensimmäisen sailion tiedot:"  
print sailio1  
print "Toisen sailion tiedot:"  
print sailio2
```

```
main()
```