Ohjelmoinnin perusteet Y Python

T-106.1208

17.3.2010

Olioista (kertausta)

- Olioiden avulla voidaan kuvata useammasta arvosta koostuvaa kokonaisuutta yhtenä oliona.
- Olioilla on *kenttiä*. Kenttien avulla kuvataan olion ominaisuuksia ja niiden arvoja.
- Oliolle mahdolliset toimenpiteet määritellään *metodien* avulla.
- Luokassa määritellään, millaisia luokan oliot ovat ja mitä operaatoita olioille voidaan tehdä. Operaatiot määritellään metodien avulla.

Luokka ja metodit (kertausta)

- Metodi `__init__` määrittelee, mitä tapahtuu, kun luodaan uusi luokan olio.
- Kun metodi on määritelty, uusia olioita voidaan luoda (yleensä luokan ulkopuolella) luokan nimen avulla:

```
kurssilainen1 = Opiskelija("Matti Virta", "11223U")
```

- Luokkaan määritellään muita metodeita esim. kenttien arvojen selvittämiseen ja muuttamiseen sekä muihin tarvittaviin operaatioihin. Esimerkiksi metodia `muuta_tenttiarvosana` voidaan kutsua

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

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

`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 kokonaisluku!"
            print "Anna uusi!"
    return luku
```

Vesisailio-olioita käyttävä pääohjelma, koodi jatkuu

```
def main():
    print "Anna ensimmaisen sailion koko."
    koko1 = lue_desimaaliluku()
    sailio1 = vesisailio.Vesisailio(koko1)
    print "Anna toisen sailion koko."
    koko2 = lue_desimaaliluku()
    sailio2 = vesisailio.Vesisailio(koko2)

    print "Lisataan vetta 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 vetta 2. sailioon."
print "Anna lisattava maara."
lisays = lue_desimaaliluku()
lisattiin = sailio2.lisaa(lisays)
print "Sailioon lisattiin %.2f l." % (lisattiin)

print "Poistetaan vetta 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 vetta 2. sailiosta."
    print "Anna poistettava maara."
    poisto = lue_desimaaliluku()
    poistettiin = sailio2.poista(poisto)
    print "Sailiosta poistettiin %.2f l." % (poistettiin)

    print "Ensimmaisen sailion tiedot:"
    print sailio1
    print "Toisen sailion tiedot:"
    print sailio2


main()
```

Olio metodin parametrina: luokka `Tasovektori`

- Halutaan kirjoittaa luokka kaksiulotteisen vektorin $a\bar{i} + b\bar{j}$ kuvaamiseen.
- Kuvataan kertoimia *a* ja *b* kentillä `__x_kerroin` ja `__y_kerroin`.
- Määritellään metodit mm. vektorin pituuden ja kahden vektorin pistetulon laskemiseen.
- Pistetulon laskeva metodi tarvitsee tiedon kahdesta eri `Tasovektori`-oliosta.
- Toinen olio yksilöidään metodin kutsussa ennen pistettä ja metodin nimeä, toinen annetaan metodille parametrina.

Parametriolion kenttien arvon selvittäminen

- Parametrina saadun olion kentät voidaa selvittää joko pistenotaation avulla

```
def pistetulo(self, toinen_vektori):
    tulo = self.__x_kerroin * toinen_vektori.__x_kerroin + \
           self.__y_kerroin * toinen_vektori.__y_kerroin
    return tulo
```

- Toinen vaihtoehto on käyttää luokan metodeita:

```
def pistetulo2(self, toinen_vektori):
    tulo = self.__x_kerroin * \
           toinen_vektori.kerro_x_kerroin() + \
           self.__y_kerroin * \
           toinen_vektori.kerro_y_kerroin()
    return tulo
```

Luokka Tasovektori, koodi

```
import math

class Tasovektori:

    def __init__(self, eka_kerroin, toka_kerroin):
        self.__x_kerroin = eka_kerroin
        self.__y_kerroin = toka_kerroin

    def kerro_x_kerroin(self):
        return self.__x_kerroin

    def kerro_y_kerroin(self):
        return self.__y_kerroin
```

Luokka Tasovektori, koodi jatkuu

```
def laske_pituus(self):
    return math.sqrt(self.__x_kerroin ** 2 + \
                     self.__y_kerroin ** 2)

def kerro_luvulla(self, kertoja):
    self.__x_kerroin *= kertoja
    self.__y_kerroin *= kertoja

def pistetulo(self, toinen_vektori):
    tulo = self.__x_kerroin * \
           toinen_vektori.__x_kerroin + \
           self.__y_kerroin * \
           toinen_vektori.__y_kerroin
    return tulo
```

Luokka Tasovektori, koodi jatkuu

```
def __str__(self):
    if self.__y_kerroin >= 0:
        mjono = "%.3fi + %.3fj" % \
                (self.__x_kerroin, self.__y_kerroin)
    else:
        mjono = "%.3fi - %.3fj" % \
                (self.__x_kerroin, - self.__y_kerroin)
    return mjono
```

Esimerkki `Tasovektori`-olioiden käytöstä

- Seuraavien kalvojen pääohjelma luo kaksi `Tasovektori`-oliota ja kutsuu niille eri metodeita.
- Käsiteltävien vektoreiden tiedot pyydetään käyttäjältä.
- Desimaalilukujen lukemista varten on jälleen määritelty oma apufunktio.

Esimerkki vektoreiden käsittelystä, koodi

```
import tasovektori

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
```

Esimerkki vektoreiden käsittelystä, koodi jatkuu

```
def main():
    print "Anna ensimmaisen vektorin i-kerroin."
    i_kerroin = lue_desimaaliluku()
    print "Anna ensimmaisen vektorin j-kerroin."
    j_kerroin = lue_desimaaliluku()
    a_vektori = tasovektori.Tasovektori(i_kerroin, j_kerroin)
    print "Antamasi vektori on", a_vektori

    print "Anna toisen vektorin i-kerroin."
    i_kerroin = lue_desimaaliluku()
    print "Anna toisen vektorin j-kerroin."
    j_kerroin = lue_desimaaliluku()
    b_vektori = tasovektori.Tasovektori(i_kerroin, j_kerroin)
    print "Antamasi vektori on", b_vektori
```

Esimerkki vektoreiden käsittelystä, koodi jatkuu

```
    pituus1 = a_vektori.laske_pituus()
    pituus2 = b_vektori.laske_pituus()
    print "Vektorin %s pituus on %.3f" % (a_vektori, pituus1)
    print "Vektorin %s pituus on %.3f" % (b_vektori, pituus2)

    laskettu_tulo = a_vektori.pistetulo(b_vektori)
    print "Vektoreiden pistetulo on %.3f." % (laskettu_tulo)

    print "Milla luvulla ensimmainen vektori kerrotaan?"
    kerroin = lue_desimaaliluku()
    a_vektori.kerro_luvulla(kerroin)
    print "Ensimmainen vektori kertomisen jalkeen", a_vektori


main()
```

Luokkien kenttien näkyvyys

- Esimerkeissä luokkien kenttien nimet ovat alkaneet kahdella alaviivalla.
- Tämä saa aikaan sen, että kenttiin ei pääse suoraan käsiksi luokan ulkopuolelta, vaan olion kenttiä on aina käsiteltävä luokan metodien avulla.
- Seuraavan esimerkin avulla esitetään, miksi tämä on toivottava tapa käsitellä kenttiä.

Esimerkki kenttien käsittelystä suoraan

- Olkoon määritelty luokka henkilötietojen käsittelyyn.

```
class Henkilo1:

    def __init__(self, nimi1):
        self.nimi = nimi1
        self.ika = 0
```

Luokkaa käytetään apuna monessa eri ohjelmassa, esimerkiksi:

```
oppilas = Henkilo1("Matti")
oppilas.ika = 15
if oppilas.ika < 18:
    print "Oppilas on alaikainen"
else:
    print "Oppilas on taysi-ikainen"
oppilas.ika = 16
print "Oppilaan ika on", oppilas.ika
```

Esimerkki kenttien käsittelystä jatkuu

- Jossain vaiheessa kenttä `ika` päätetään korvata kentällä `syntymavuosi`. Luokan määrittely alkaa nyt

```
class Henkilo1:

    def __init__(self, nimi1):
        self.nimi = nimi1
        self.syntymavuosi = 0
```

- Millaisia muutostarpeita tämä aiheuttaa niissä ohjelmissa, jotka käyttävät tätä luokkaa?

Esimerkki vaihtoehtoisesta tavasta

- Tarkastellaan vaihtoehtoista tapaa: luokan kenttiä voi käsitellä vain luokan metodien avulla:

```
class Henkilo2:

    def __init__(self, nimi1):
        self.__nimi = nimi1
        self.__ika = 0

    def kerro_ika(self):
        return self.__ika

    def muuta_ika(self, uusi_ika):
        if 0 <= uusi_ika <= 150:
            self.__ika = uusi_ika
```

Esimerkki vaihtoehtoisesta tavasta jatkuu

- Tätäkin luokkaa käytettäisiin apuna monessa ohjelmassa henkilötietojen käsittelyyn:

```
oppilas = Henkilo2("Matti")
oppilas.muuta_ika(15)
if oppilas.kerro_ika() < 18:
    print "Oppilas on alaikainen"
else:
    print "Oppilas on taysi-ikainen"
print "Oppilaan ika on", oppilas.kerro_ika()
```

Esimerkki vaihtoehtoisesta tavasta jatkuu

- Luokan kenttä `ika` korvataan kentällä `syntymavuosi`:

```
NYKYINEN_VUOSI = 2010

class Henkilo2:

    def __init__(self, nimi1):
        self.__nimi = nimi1
        self.__syntymavuosi = 0

    def kerro_ika(self):
        return NYKYINEN_VUOSI - self.__syntymavuosi

    def muuta_ika(self, uusi):
        if 0 <= uusi <= 150:
            self.__syntymavuosi = NYKYINEN_VUOSI - uusi
```

- Mitä pitää muuttaa tätä luokkaa käyttävissä ohjelmissa?

Huomautus

- Python-kieltä käytettäessä kenttien yksityisyys ei ole aivan niin tärkeää kuin edellä on esitetty, koska luokkaa jälkikäteen muokatessa pystyy Pythonin valmiin `property`-funktion avulla määräämään sen, että kentän suoran käsittelyn sijasta käytetäänkin määrättyä metodia.
- Tätä mahdollisuutta ei ole kuitenkaan monessa muussa olio-ohjelmointikielessä (esim. Java).
- Lisäksi kun luokan kenttien arvoja muutetaan vain metodien avulla, on metodien yhteyteen helppo lisätä tarkastuksia siitä, että uusi arvo on järkevä.