Ohjelmoinnin perusteet Y Python

T-106.1208

16.3.2009

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.

Tiedostoon kirjoittaminen: tiedoston avaaminen

- Tiedosto on avattava myös silloin, kun tiedostoon halutaan kirjoittaa. Käsitettelytapa on kuitenkin eri kuin tiedostoon kirjoitettaessa.

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

  Tai

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

- Käsittelytapojen ero: `"w"` kirjoittaa olemassaolevan tiedoston päälle (vanha sisältö häviää kokonaan), `"a"` kirjoittaa olemassaolevan tiedoston loppuun.

Tiedostoon kirjoittaminen: rivin kirjoittaminen

- 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.

```
kanta = 3.5
ekspo = 5
tulos = kanta ** ekspo
tulostiedosto.write("%.2f potenssiin %d on %.2f\n" % \
    (kanta, ekspo, tulos))
tulostiedosto.write(str(kanta) + " potenssiin " + \
    str(ekspo) + " on " + str(tulos) + "\n")
```

Virheiden käsittely ja tiedoston sulkeminen

- Kun kaikki haluttu on kirjoitettu tiedostoon, on tiedosto syytä sulkea.

  `tiedostomuuttuja.close()`
- Tiedostoa ei ole syytä yrittää lukea ennen close-käskyn suorittamista, sillä silloin kirjoitettu tieto ei ole välttämättä vielä itse tiedostossa vaan puskurissa odottamassa kirjoittamista.
- Tiedostoon kirjoittaessa voi aiheutua erilaisista virhetilanteista `IOError`-tyyppinen poikkeus, joka on syytä käsitellä try–except-rakenteella.

Esimerkkejä tiedostoon kirjoittamisesta

- Seuraavilla kalvoilla on kaksi esimerkkiohjelmaa tiedostoon kirjoittamisesta.
- Ensimmäinen kirjoittaa käyttäjän antamat nimet tiedostoon.
- Toinen pyytää käyttäjiltä ympyröiden säteitä. Se kirjoittaa tiedostoon kullekin riville yhden säteen ja sitä vastaavan pinta-alan.

Esimerkki: nimiä tiedostoon

```
def main():
    print "Ohjelma kirjoittaa vieraslistan tiedostoon."
    nimi = raw_input("Anna kirjoitettavan tiedoston nimi: ")
    try:
        tulostiedosto = open(nimi, "w")
        print "Anna tallennettavat nimet."
        print "Lopeta tyhjalla rivilla."
        rivi = raw_input()
        while rivi != "":
            tulostiedosto.write(rivi + "\n")
            rivi = raw_input()
        tulostiedosto.close()
        print "Nimet on kirjoitettu tiedostoon", nimi
    except IOError:
        print "Virhe tiedoston", nimi, "kirjoittamisessa."

main()
```

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()
```

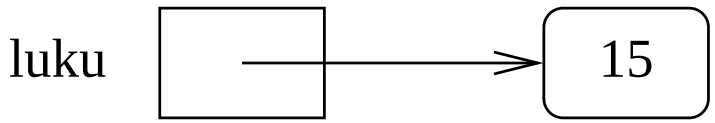
Tietojen tallentaminen ohjelman suorituskertojen välillä

- Monissa sovelluksissa ohjelman käyttämät tiedot halutaan tallentaa tiedostoon ohjelman eri suorituskertojen välillä.
- Jos käsiteltävät tietomäärät ovat kohtuullisen kokoisia, menetellään seuraavasti:
  - Ohjelman suorituksen alussa tiedot luetaan tiedostosta ja tallennetaan sopivaan tietorakenteeseen (esim. lista tai sanakirja).
  - Ohjelman suorituksen aikana mahdolliset muutokset tehdään käytettävään tietorakenteeseen, ei suoraan itse tiedostoon.
  - Ohjelman suorituksen päättyessä koko tietorakenne tallennetaan tiedostoon.

Arvot ja viitteet

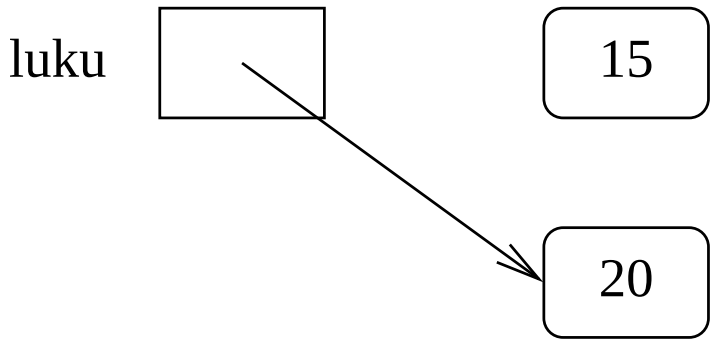
- Pythonissa kaikkien muuttujien arvoja käsitellään *viitteen* avulla. Muuttujan arvona ei ole varsinainen arvo (esim. kokonaisluku), vaan viite varsinaisen arvon sisältävään muistipaikkaan.

```
luku = 15
```



- Kun muuttujalle sijoitetaan uusi arvo, varsinaista arvoa ei korvata uudella, vaan muuttuja pannaan viittaamaan uuteen muistipaikkaan.

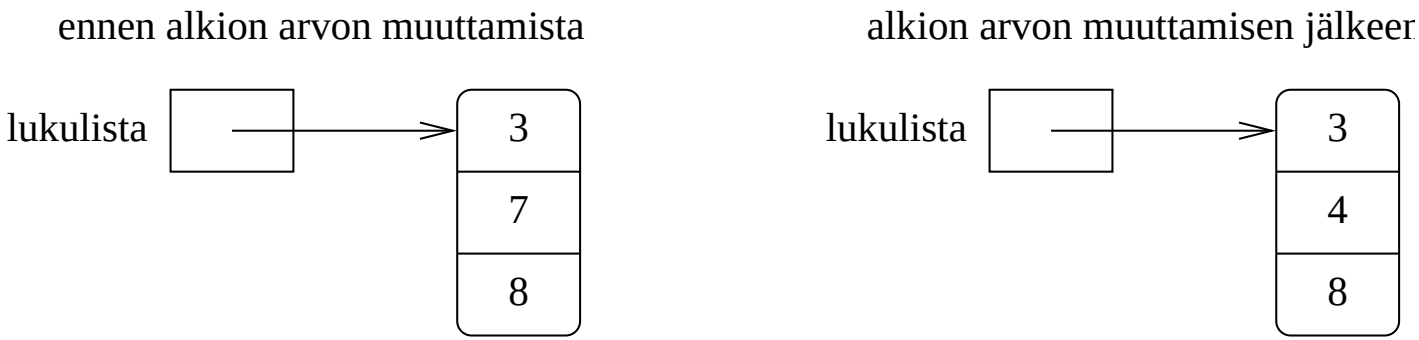
```
luku = 20
```



Muutettavat tyypit

- Sijoituskäsky siis vaihtaa muuttujan viittaamaan toiseen arvoon.
- Osa Pythonin tyypeistä (esimerkiksi listat ja sanakirjat) on kuitenkin muuttuvia (engl. mutable). Niillä varsinaista arvoa voi muuttaa.

```
lukulista = [3, 7, 8]
lukulista[1] = 4
```



- Muuttuja `lukulista` viittaa samaan listaan, mutta listan sisältö on muuttunut.

Parametrin arvon muuttaminen funktiossa

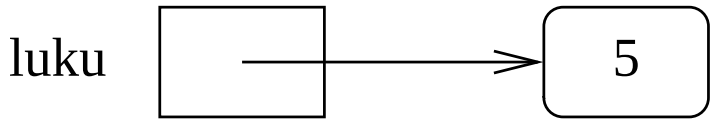
- Tähän asti esitetyistä tyypeistä lista ja sanakirja ovat muuttuvia. Muut esitetyt tyypit (esim. kokonais- ja desimaaliluvut, merkkijonot) ovat muuttumattomia (engl. immutable).
- Muuttumattomat ja muuttuvat tyypit toimivat eri tavalla funktion parametreina. Jos funktio muuttaa muuttumattomaa tyyppiä olevan parametrina arvoa, muutos ei näy mitenkään funktion ulkopuolella.
- Jos taas funktio muuttaa muuttuvaa tyyppiä olevan parametrin varsinaista arvoa, muutos näkyy myös funktion ulkopuolella.

Esimerkki 1: parametrina lukuja

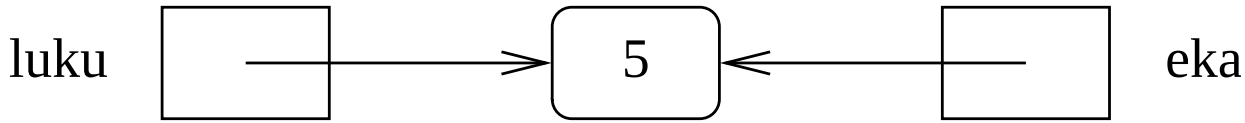
```
def muuta_luku(eka):
    print "Arvo funktiossa aluksi", eka
    eka = 10
    print "Arvo funktiossa lopuksi", eka


def main():
    luku = 5
    print "Arvo paaohjelman aluksi", luku
    muuta_luku(luku)
    print "Arvo paaohjelman lopuksi", luku


main()
```

Mitä edellisen kalvon ohjelman suorituksessa tapahtuu?

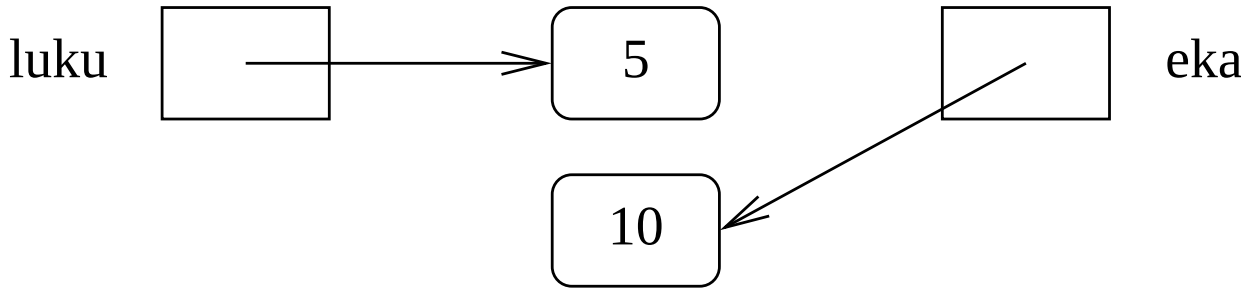
- Pääohjelmassa muuttujille `luku` pannaan viittaamaan arvoon 5.



- Funktion alussa parametri pannaan viittaamaan samaan arvoon.



- Funktion sijoituskäskyssä parametri pannaan viittaamaan uuteen arvoon. Itse arvoa ei kuitenkaan muuteta, joten pääohjelman muuttuja `luku` viittaa edelleen samaan arvoon kuin aikasemminkin.



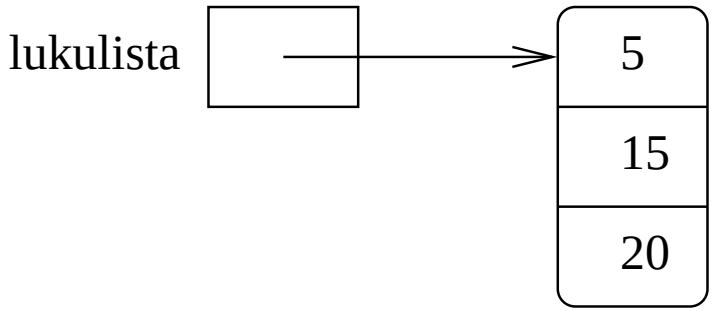
- Kun palataan takaisin pääohjelmaan, muuttujan `luku` arvo ei ole muuttunut.

Esimerkki 2: parametrina lista

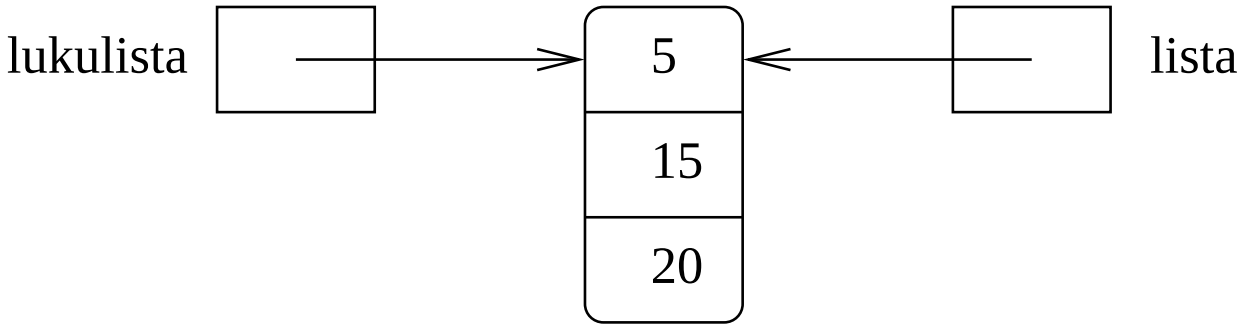
```
def muuta_alkio(lista):
    print "Lista funktiossa aluksi", lista
    lista[1] = 12
    print "Lista funktiossa lopuksi", lista


def main():
    lukulista = [5, 15, 20]
    print "Lista paaohjelman aluksi", lukulista
    muuta_alkio(lukulista)
    print "Lista paaohjelman lopuksi", lukulista


main()
```

Mitä toisen esimerkin suorituksessa tapahtuu?

- Pääohjelmassa luodaan lista ja pannaan muuttuja `lukulista` viittaamaan siihen.

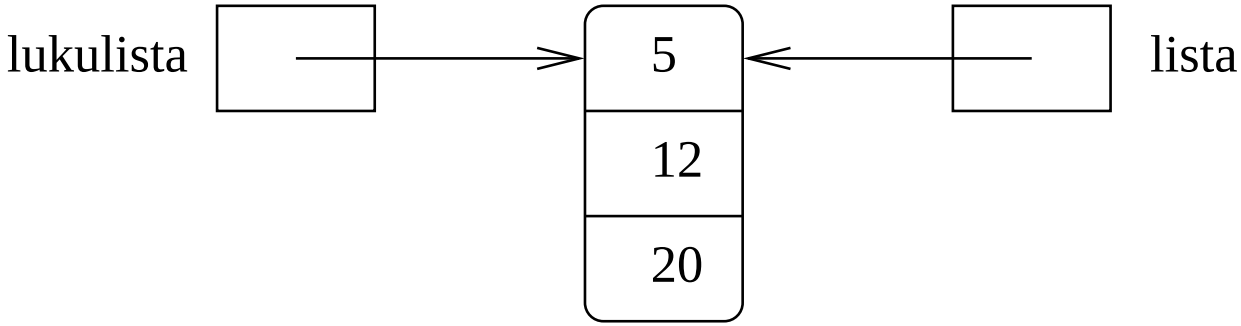


- Funktion suorituksen alussa parametri pannaan viittaamaan samaan listaan.

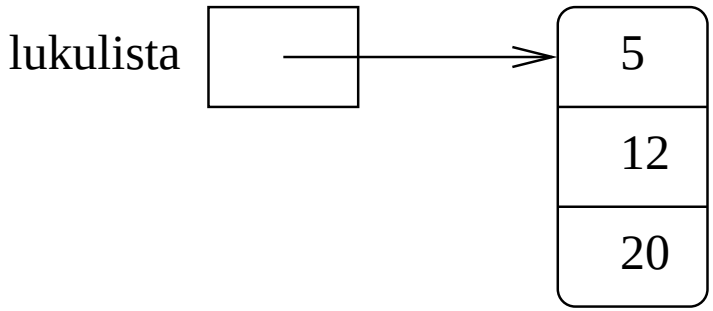


Mitä toisen esimerkin suorituksessa tapahtuu? (jatkoa)

- Funktiossa muutetaan yhtä listan alkiota, mutta parametri `lista` viittaa edelleen samaan listaan kuin suorituksen alussa. Vain listan sisältö on muuttunut.



- Pääohjelman muuttuja `lukulista` viittaa edelleen samaan listaan kuin aluksi. Koska tämän listan alkiota on muutettu, muutos näkyy myös pääohjelmassa.



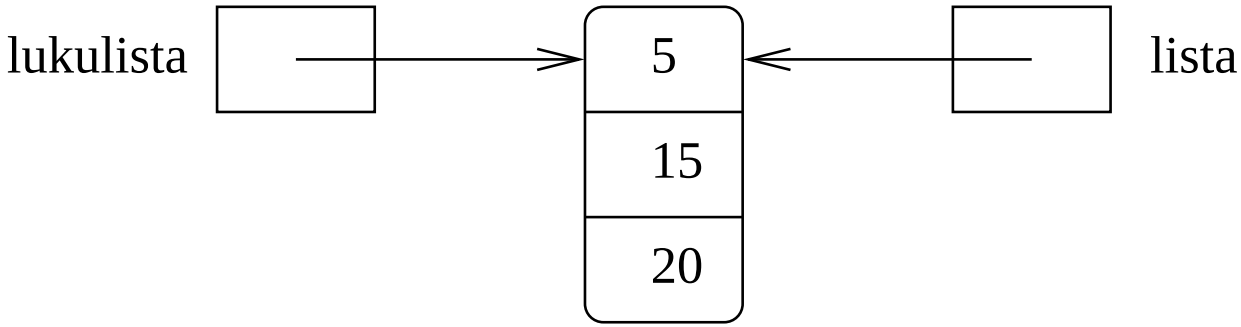
Kolmas esimerkki

- Jos kuitenkin funktio muuttaa itse listaparametria eikä listan sisältöä, muutos ei näy funktion ulkopuolella.

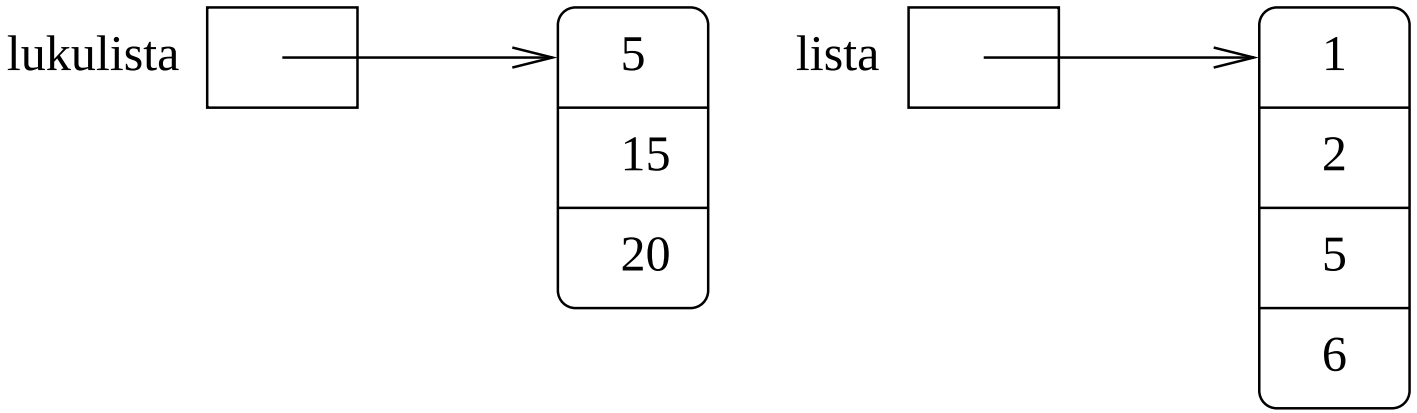
```
def muuta_lista(lista):
    print "Lista funktiossa aluksi", lista
    lista = [1, 2, 5, 6]
    print "Lista funktiossa lopuksi", lista

def main():
    lukulista = [5, 15, 20]
    print "Lista paaohjelman aluksi", lukulista
    muuta_lista(lukulista)
    print "Lista paaohjelman lopuksi", lukulista

main()
```

Mitä kolmannessa esimerkissä tapahtuu?

- Funktion suorituksen alussa parametri `lista` viittaa samaan listaan kuin pääohjelman muuttuja `lukulista`



- Kun funktiossa tehdään sijoituskäsky parametriin `lista`, panee se parametrin viittaamaan kokonaan uuteen listaan. Se ei siis muuta vanhan listan sisältöä.



Mitä oliot ovat?

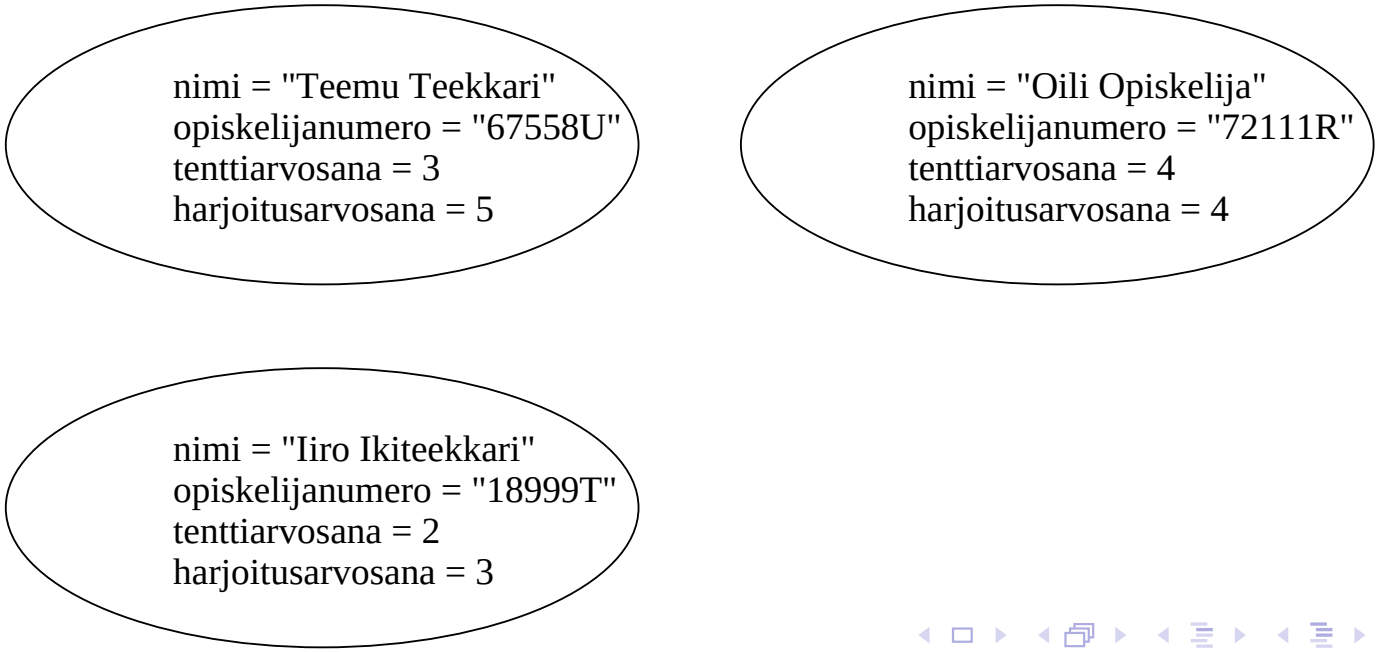
- Esimerkki: halutaan laatia ohjelma, joka käsittelee erään ohjelmointikurssin opiskelijoita. Kurssilla on noin 100 opiskelijaa.
- Jokaisesta opiskelijasta halutaan ohjelman käyttöön ainakin nimi, opiskelijanumero, tenttiarvosana ja harjoitusarvosana.
- Ongelma: miten opiskelijoiden tietoja esitetään 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 alkiota 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. 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.



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 määrittelyn ymmärtämistä.