Listat

Ohjelmoinnin perusteet

7

Turun yliopisto
University of Turku

Tiedon säilömisestä

- Usein ohjelmoitaessa on tärkeää säilöä tietoa **tietorakenteeseen**

- Yksittäisten merkkijonojen ja lukujen sijasta tarvitaan usein keino tallentaa ja käsitellä suurempia tietomääriä **yhtenä kokonaisuutena**

Lista

- Tätä tarkoitusta varten useimmat ohjelmointikielet sisältävät esimerkiksi jonkinlaisen **listarakenteen**.

- Listaan voidaan säilöä useita **alkioita**.

Lista ja alkiot

- Listan alkiot on tallennettu johonkin **järjestykseen.**

- Järjestys voi noudattaa jotain luonnollista järjestystä (esimerkiksi pienimmästä suurimpaan tai aakkosjärjestys), mutta välttämättä näin ei ole. Listan alkioita voidaan myös järjestellä uudestaan.

Listan ominaisuuksia

- Lista on **mutatoituva:** alkioita voidaan lisätä, poistaa ja muuttaa

- Tämä erottaa listan esimerkiksi merkkijonosta, jonka merkkejä ei voida luomisen jälkeen lisätä tai poistaa (joskin niiden perusteella voidaan luoda uusia merkkijonoja).

Lista Pythonissa

- Pythonissa lista alustetaan [ ]-operaattorilla. Alkiot erotetaan toisistaan pilkulla. Syntaksi:

```
<listamuuttuja> = [ <alkiot> ]
```

Lista Pythonissa (2)

- Lista voidaan siis luoda valmiita alkuarvoja käyttäen:

```
lista1 = [1, 2, 4, 6]
```

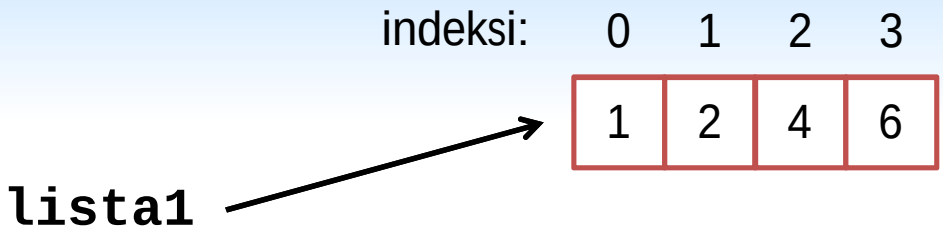
- ...tai alustaa tyhjä lista:

```
tyhjaLista = []
```

Listan indeksointi

- Kuten merkkijonot, lista on indeksoitu nollasta alkaen:

```
lista1 = [1, 2, 4, 6]
```



Alkion poimiminen

- Yksittäinen alkio voidaan poimia [ ]-operaattorilla. Mekaniikka on sama kuin yksittäisen merkin poimiminen merkkijonosta:

```
lista = [10, 9, 7, 8, 3]

print lista[0] # Tulostaa 10

summa = lista[1] + lista[3] # 9 + 8
print summa # Tulostaa 17
```

Alkion arvon muutos

- Toisin kuin merkkijonoissa, listan alkioiden arvoja voidaan myös muuttaa:

```
lista = [10, 9, 7, 8, 3]

lista[0] = 5
print lista # Tulostaa [5, 9, 7, 8, 3]
```

Uusien alkioiden lisääminen

- Uusia alkioita voidaan lisätä listan loppuun `append`-metodilla.

Syntaksi:

```
<listamuuttuja>.append(<alkio>)
```

Esimerkki

```
lista = [10, 9, 7, 8, 3]

lista.append(5)
print lista # Tulostaa [5, 9, 7, 8, 3, 5]

lista2 = []
lista2.append(1)
lista2.append(10)
print lista2 # Tulostaa [1,10]
```

Alkioiden poisto

- Alkioita voidaan poistaa listasta joko `pop`- tai `remove`-metodin avulla.

- Erotuksena on, että `pop`-metodille annetaan poistettavan alkion indeksi, mutta `remove`-metodille poistettavan alkion arvo.

Alkion poisto pop-metodilla

- Metodi `pop` poistaa siis alkion annetun indeksin kohdalta. Metodi myös palauttaa poistetun alkion.

```
lista = [10, 9, 7, 8, 3]

lista.pop(0)
print lista # Tulostaa [9, 7, 8, 3]

alkio = lista.pop(2)
print lista # Tulostaa [9, 7, 3]
print alkio # Tulostaa 8
```

Alkion poisto remove-metodilla

- Metodi `remove` sen sijaan saa parametrikseen poistettavan alkion arvon. Jos alkiota ei löydy listasta, seuraa virheilmoitus.

```
lista = [10, 9, 7, 8, 3]

lista.remove(9)
print lista # Tulostaa [10, 7, 8, 3]

lista.remove(5) # VIRHE!
```

Lista-alkioista

- Listaan voidaan tallentaa minkä tahansa tyyppisiä alkioita, esimerkiksi kokonaislukuja, desimaalilukuja, totuusarvoja ja merkkijonoja
- Listaa alustaessa voidaan käyttää mitä tahansa lausekkeita; lausekkeen evaluoitu arvo tallennetaan lista-alkioksi.
- Listaan voidaan myös tallentaa toisia listoja.

Lista-alkioista (2)

- Python sallii eri tyyppisten arvojen tallentamisen samaan listaan.

- Yleensä tämä ei kuitenkaan ole hyvä idea, koska tämä saattaa aiheuttaa listaa läpikäytäessä vaikeasti löydettäviä virheitä.

Eri tyyppisiä listoja

- Esimerkkejä eri tyyppisistä listoista:

```
luvut = [7, 4, 12, 42, 4] # Kokonaislukuja

totuudet = [True, False, False, True] # Totuusarvoja

desim = [1.0, 3.5, 2.25, 0.757] # Desimaalilukuja

nimet = ["Pekka", "Arto", "Jalmari"] # Merkkijonoja

lista = [[1, 2, 3], [4, 5, 6]] # Listoja listassa
```

Listan pituus

- Listan pituus (eli alkioiden määrä) voidaan palauttaa `len`-funktiolla. Funktio toimii samalla tavalla kuin merkkijonojen yhteydessä:

```
luvut = [7, 4, 12, 42, 4] # Kokonaislukuja

pituus = len(luvut)
print pituus # Tulostaa 5

print luvut[0] # Eka alkio
print luvut[len(luvut) - 1] # Viimeinen alkio
```

Alilistat

- Listasta voidaan myös poimia "alilistoja" samalla mekaniikalla kuin merkkijonojen alijonoja poimittiin (ks. **kalvot 3**)

```
lista = [1, 2, 4, 8, 16]

ali = lista[0 : 3]
print ali # Tulostaa [1, 2, 4]

print lista[2 : 4] # Tulostaa [4, 8]
```

Listan läpikäynti

- Usein ohjelmissa on tarpeen käydä läpi kaikki listan alkiot. Tätä kutsutaan **iteroinniksi.**

- Lista voidaan iteroida käyttäen `while`-silmukkaa, niin kuin aikaisemmin on opittu.

Esimerkki

```
lista = [1, 2, 4, 8, 16]

indeksi = 0
while indeksi < len(lista):
    print lista[indeksi]
    indeksi = indeksi + 1
```

Esimerkki 2

- Funktio, joka laskee ja palauttaa listan alkioiden summan:

```
def alkioidenSumma(lista):
    summa = 0
    indeksi = 0
    while indeksi < len(lista):
        summa = summa + lista[indeksi]
        indeksi = indeksi + 1
    return summa

# Testi
lista = [1, 2, 3, 5]
print alkoidenSumma(lista) # Tulostaa 11
```

For-silmukka

- Pythonissa on myös toinen silmukkarakenne, joka on erittäin kätevä listan läpikäyntiin. Syntaksi:

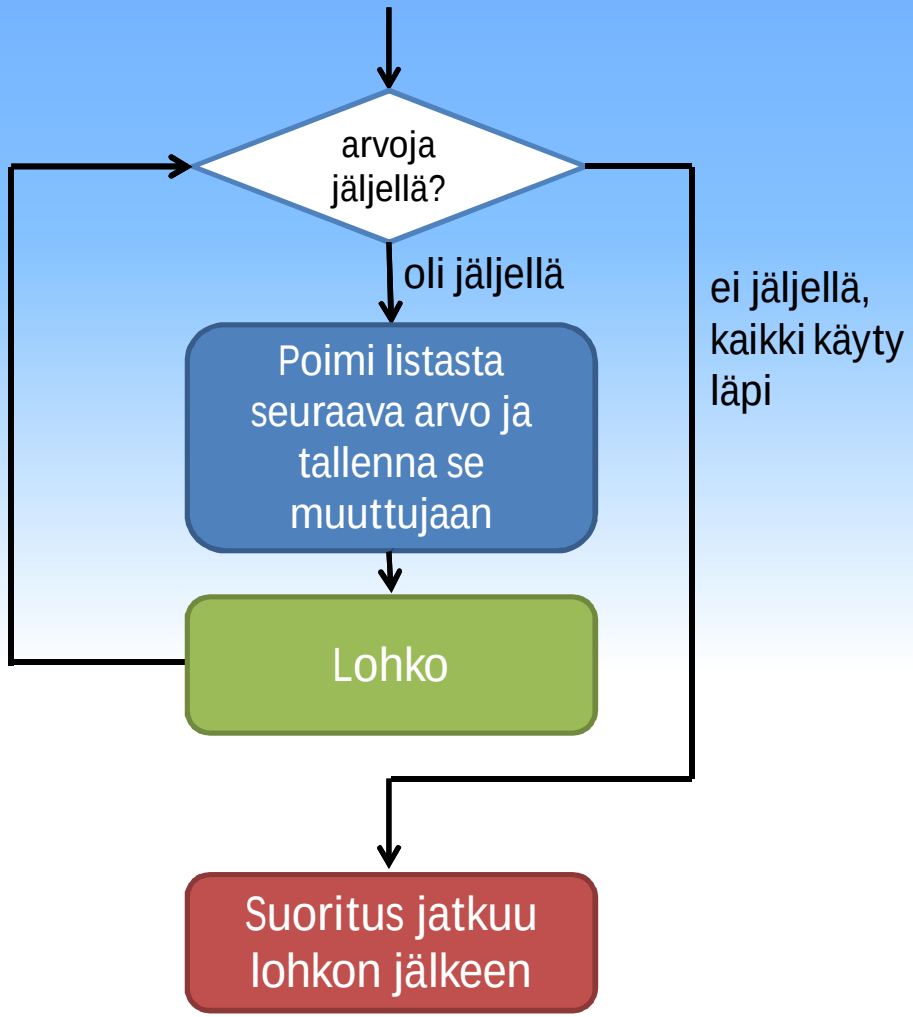
```
for <muuttuja> in <lista>:
    <lohko>
```

For-silmukka (2)

- For-silmukassa listan alkiot poimitaan yksi kerrallaan alusta alkaen, ja tallennetaan annettuun muuttujaan.

- Tämän jälkeen lohko suoritetaan ja poimitaan uusi arvo. Tämä jatkuu kunnes koko lista on käyty läpi.

For-silmukan suoritus



Esimerkki

- Funktio, joka laskee listan alkioiden summan toteutettuna `for`-silmukalla:

```
def alkioidenSumma(lista):
    summa = 0
    for arvo in lista:
        summa = summa + arvo
    return summa

# Testi
lista = [1, 2, 3, 5]
print alkoidenSumma(lista) # Tulostaa 11
```

Esimerkki 2

- Funktio, joka laskee kuinka monta yli kolmen merkin pituista merkkijonoa listassa on:

```
def pidempiaJonoja(mlista):
    maara = 0
    for mjono in mlista:
        if len(mjono) > 3:
            maara = maara + 1
    return maara

# Testi
lista = ["a", "ab", "abcd", "abcdef"]
print pidempiaJonoja(lista) # Tulostaa 2
```

Funktio range

- Pythonista löytyy `range`-funktio, jolla voidaan generoida kokonaislukuja sisältävä lista helposti. Syntaksi:

```
range(<alku>, <loppu>)
```

- ...tai

```
range(<alku>, <loppu>, <askel>)
```

Esimerkki

```
lista = range(1,5)
print lista # Tulostaa [1, 2, 3, 4]

lista2 = range(2,10,2)
print lista2 # Tulostaa [2, 4, 6, 8]

lista3 = range(5,0,-1)
print lista3 # Tulostaa [5, 4, 3, 2, 1]
```

Range ja for-silmukka

- Funktio range on näppärä for-silmukan yhteydessä, mikäli halutaan käydä läpi tietty, ennalta määrätty arvoalue:

```
# Silmukka tulostaa arvot 0-4
for i in range(0,5):
    print i

# Silmukka laskee arvojen 1-100 summan:
summa = 0
for i in range(1,101):
    summa = summa + i
print summa
```

Range ja xrange

- Jos `range`-funktiota käytetään vain `for`-silmukan yhteydessä (eli listaa ei käsitellä muuten), ja kyseessä on hyvin pitkä silmukka, kannattaa käyttää ennemmin `xrange`-funktiota.

- Funktio `xrange` **ei luo listaa** vaan "virtuaalisen" listan läpikäyntiä varten. Pitkissä silmukoissa muistia kuluu huomattavasti vähemmän.

Esimerkki

- Funktion xrange syntaksi on täsmälleen samanlainen kuin funktion range:

```
# Silmukka laskee arvojen 1-100 000 summan:
summa = 0
for i in xrange(1,100001):
    summa = summa + i
print summa
```

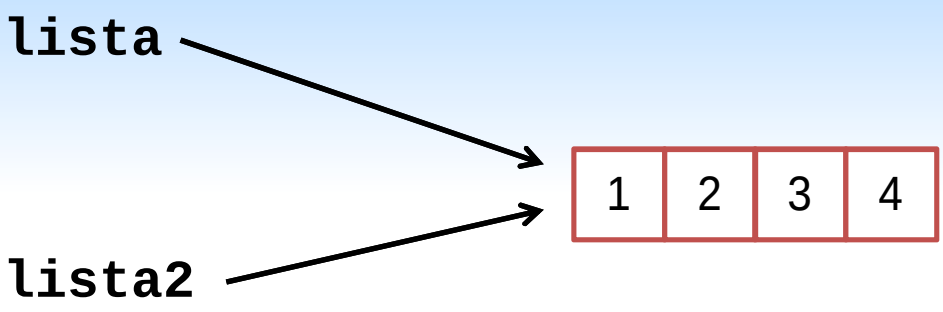
Listamuuttujat ovat viittauksia

- Samoin kuin merkkijonoissa, listamuuttujat ovat viittauksia listaan muistissa.

- Tämä tarkoittaa, että kaksi (tai vielä useampi) muuttujaa voi viitata samaan listaan.

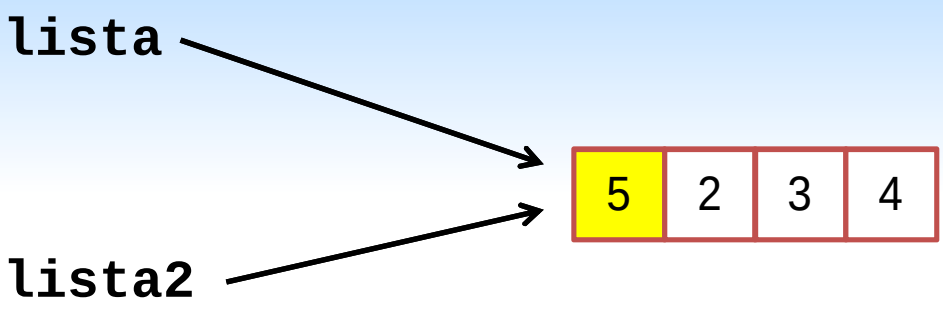
Esimerkki

```
lista = range(1,5)
lista2 = lista
```



Esimerkki (jatkuu)

```
lista[0] = 5
print lista2[0] # Tulostaa mitä?
```



Esimerkki

- Näin ollen myös listaa parametrina välittäessä välitetään viittaus listan sijasta:

```
def muutaListaa(lista):
    # Kopioi viimeinen alkio ekaksi
    lista[0] = lista[len(lista) - 1]

lista = range(1,5)
print lista # Tulostaa [1, 2, 3, 4]
muutaListaa(lista)
print lista # Tulostaa [4, 2, 3, 4]
```