Toistolauseet

Ohjelmoinnin perusteet

5

Turun yliopisto
University of Turku

Silmukat

- Tietokoneiden tehokkuus perustuu nopeuteen: ne voivat laskea miljoonia (tai miljardeja) laskutoimituksia sekunnissa.

- Eräs ohjelmoijan tärkeimmistä työkaluista onkin silmukoiden käyttö.

Silmukat (2)

- Silmukkarakenteen avulla voidaan määrätä ohjelmakoodi suoritettavaksi uudestaan ja uudestaan (yleensä pienillä muutoksilla) niin kauan kun jokin ehto on tosi.

- Kun ehto muuttuu epätodeksi, silmukan suoritus päättyy ja ohjelman suoritus jatkuu silmukan jälkeen.

Mihin toistoa tarvitaan?

- Pitkien tai tuntemattoman pituisten rakenteiden (listat, merkkijonot, tiedostot) läpikäynti ja prosessointi.
- Ohjelman tai sen osion toisto niin pitkään kun käyttäjä haluaa.
- Säännöllisten tai epäsäännöllisten viestien (esimerkiksi verkkoliikenteen) havainnointi.

Toistolause: while

- Pythonissa toisto toteutetaan `while`-lauseen avulla. Syntaksi:

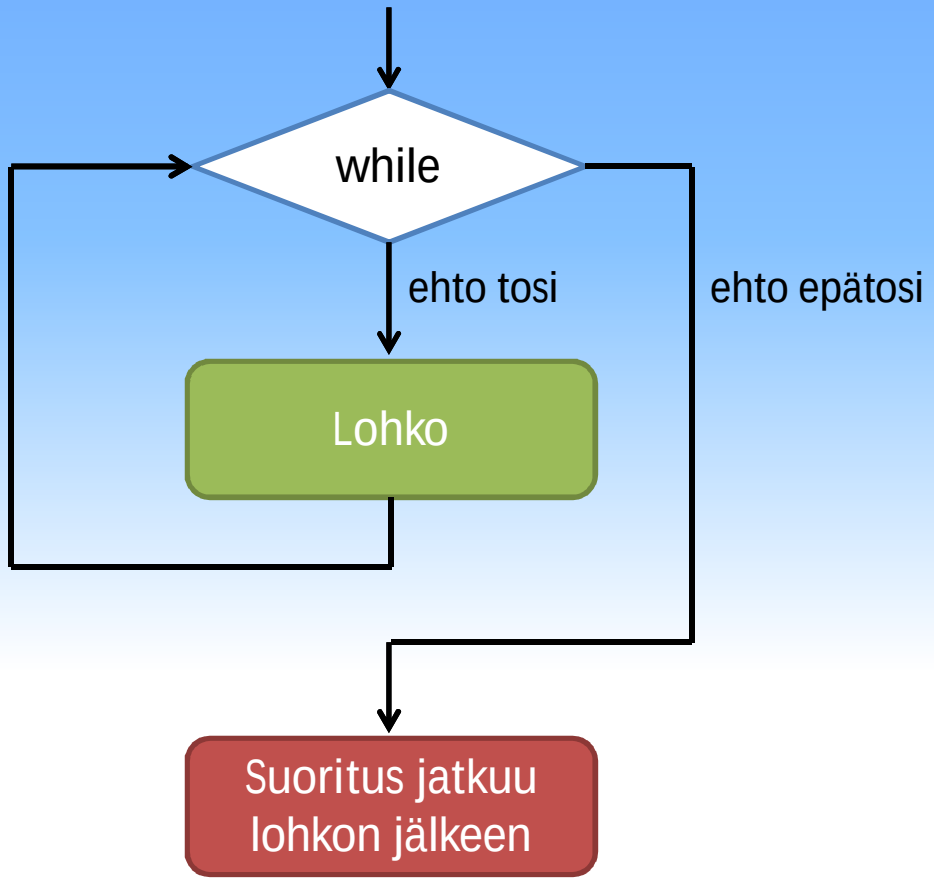
```
while <ehtolauseke>:
    <suoritettava lohko>
```

Toistolause: while (2)

- `while`-lause siis muistuttaa huomattavan paljon `if`-lausetta (ja jopa sallii `else`-haaran).

- Erona on se, että `while`-lausetta seuraavaa lohkoa suoritetaan uudestaan ja uudestaan niin pitkään kun ehto on tosi; suoritus päättyy vasta kun ehto muuttuu epätodeksi tai käyttäjä pysäyttää ohjelman.

While-lohkon suoritus



While-lohko suoritus (2)

- While-lohkon suoritus alkaa ehdon evaluoinnilla
- Tämä tarkoittaa, että jos ehto on heti epätosi, ei lohkoa suoriteta kertaakaan
- Ehto evaluoidaan uudestaan aina joka kerta kun lohko on suoritettu.

Esimerkki

- Ohjelma tulostaa luvut 1-10 näytölle allekkain:

```
# alustetaan muuttuja
n = 1

# silmukkaa suoritetaan niin kauan
# kun n <= 10
while n <= 10:
    print n # tulosta n
    n = n + 1 # kasvata yhdellä
```

Toistolauseen osat

- Toimiva toistolause vaatii, että seuraavat kolme osaa on toteutettu:

- **Alustus**: alustetaan silmukassa käytettävä silmukkamuuttuja tai -muuttujat
- **Ehto**: verrataan silmukkamuuttujan tai –muuttujien arvoa johonkin ehtoon
- **Muutos**: muutetaan silmukkamuuttujan tai –muuttujien arvoja jokaisella kierroksella.

Toistolauseen osat (2)

- Jos jokin osista jää toteuttamatta, tai on toteutettu väärin, on mahdollista että syntyy ikuinen silmukka.

Esimerkiksi:

```
# alustetaan muuttuja
n = 1

# silmukkaa suoritetaan niin kauan kun n <= 10
while n <= 10:
    print n # tulosta n
    # Koska n:n arvo ei muutu, silmukka ei pääty
```

Suorituksen pysäyttäminen

- Silmukan suoritus (tai minkä tahansa muun Python-ohjelman suoritus) voidaan pysäyttää painamalla CTRL ja C -näppäimiä yhtä aikaa näppäimistöltä.

Ehtolausekkesta

- While-lauseen ehdossa voidaan käyttää mitä tahansa ehtolauseketta, joka evaluoituu arvoksi `True` tai `False`.

- Silmukan suoritus jatkuu niin kauan kunnes ehtolauseke evaluoituu arvoksi `False`.

Esimerkki

- Ohjelma laskee annettujen lukujen neliöitä kunnes käyttäjä haluaa lopettaa:

```
uudestaan = True # Alustetaan todeksi

while uudestaan:
    luku = input("Anna luku:")
    print "Luvun nelio on", luku ** 2

    k = raw_input("Lasketaanko toinen (k/e)? ")
    if k != "k":
        uudestaan = False

print "Kiitos ja moi!"
```

Esimerkki (2)

- Ohjelma etsii merkkijonosta ensimmäisen vokaalin:

```
vok = "aeiouyääö" # Vokaalit
n = 0 # aloitetaan alusta
loytyi = False
mjono = raw_input("Anna merkkijono:")

while not loytyi and n < len(mjono):
    if vok.find(mjono[n]) > -1: # onko vokaali?
        loytyi = True
    else:
        n = n + 1 # ei ollut, seuraava merkki

if loytyi: # Löytyiko vokaalia?
    print "Vokaali oli kohdassa",n
```

Toisto- ja ehtolauseiden yhdistäminen

- Kuten edellisestä esimerkistä nähtiin, usein on tarpeen yhdistää toisto- ja ehtolauseita.

- Tällaisissa tapauksissa on syytä olla erittäin huolellinen lohkoja kirjoittaessa.

Esimerkki

- Ohjelma kysyy käyttäjältä lukuja kunnes annetaan -1, ja laskee lukujen keskiarvon.

```
luku = 1 # luvun alkuarvoksi joku muu kuin -1
summa = 0 # alusta summa
maara = 0 # alusta määrä
while luku > -1:
    luku = input("Anna luku tai -1 lopettaaksesi:")
    if luku > -1:
        summa = summa + luku # lisataan summaan
        maara = maara + 1 # kasvata maaraa yhdella
# laske keskiarvo, kerro summa 1.0:lla,
# jotta saadaan oikea keskiarvo eikä alas pyöristettyä
print "Keskiarvo:",(summa * 1.0) / maara
```

Esimerkki (2)

- Ohjelma laskee käyttäjän antaman luvun kertoman (eli luvun **n** kertoman **n!**).

```
n = input("Anna luku:") # Kysy luku

if n < 0: # Negatiivinen, ei onnistu
   print "Kertomaa ei voida laskea"
elif n == 0 or n == 1: # 1! == 0! == 1
   print "Kertoma: 1"
else:
   k = n - 1 # k:n alkuarvoksi n-1
   while k > 1:
      n = n * k # kerro n k:lla
      k = k - 1 # vähennä k:sta yksi
   print "Kertoma:",n
```

Silmukan suorituksen katkaiseminen

- Silmukan suoritus voidaan katkaista ohjelmallisesti käyttämällä `break`-lauseella.

- Kun `break`-lause suoritetaan, suoritus jatkuu välittömästi ensimmäisestä silmukan jälkeisestä komennosta.

Esimerkki

- Ohjelma tulostaa lukuja ensimmäiseen viidellä jaolliseen lukuun asti:

```
luku = 1

while True: # "ikuinen" silmukka
    print luku

    if luku % 5 == 0: # Testaa jaollisuus
        break # Pois silmukasta

    luku = luku + 1
```

Jakojäännösoperaattori %

- Niin kuin edellisestä esimerkistä nähtiin, Pythonin jakojäännösoperaattorilla voidaan palauttaa jakolaskun jakojäännös.

- a % b siis palauttaa jakolaskun a / b jakojäännöksen. Esimerkiksi 5 % 2 == 1.

Jakojäännösoperaattori (2)

- Operaattori on hyödyllinen esimerkiksi silloin, kun pitää selvittää onko luku parillinen vai pariton:

```
luku = input("Anna luku: ")

if luku % 2 == 0: # Parillinen luku
   print "Luku on parillinen"
else:
   print "luku on pariton"
```

Break-lauseesta vielä

- Huomaa, että `break`-lausetta ei välttämättä tarvita, koska saman toiminnallisuuden voi saada aikaan muokkaamalla while-lauseen ehtoa.

- Joskus `break` kuitenkin selkeyttää ja/tai nopeuttaa ohjelman toimintaa.

Esimerkki

- Versio, joka käyttää `break`-lausetta:

```
luku = 1

while True: # "ikuinen" silmukka
    print luku

    if luku % 5 == 0: # Testaa jaollisuus
        break # Pois silmukasta

    luku = luku + 1
```

Esimerkki (jatkuu)

- Versio ilman `break`-lausetta:

```
luku = 0

while luku % 5 != 0:
    luku = luku + 1 # Kasvatetaan ensin...
    print luku
```

Continue-lause

- Silmukan suoritus voidaan siirtää suoraan ehdon evaluointiin `continue`-lauseella

- Lause siis katkaisee silmukan nykyisen kierroksen suorituksen.

Esimerkki

- Ohjelma laskee parillisten lukujen neliöt:

```
luku = 0

while luku < 20:
    luku = luku + 1 # kasvata ensin
    if luku % 2 == 1:
        continue # jos pariton, siirry while-riville
    nelio = luku * luku
    print nelio
```

Esimerkki (jatkuu)

- Sama ohjelma ilman `continue`-lausetta:

```
luku = 0

while luku < 20:
    luku = luku + 1 # kasvata ensin
    if luku % 2 == 0: # jos parillinen...
        nelio = luku * luku
        print nelio
```

Continue ja ikuinen silmukka

- Käytettäessä continue-lausetta on oltava tarkkana, ettei synny ikuista silmukkaa:

```
luku = 0

while luku < 20:
    if luku % 2 == 1:
        continue # jos pariton, aloita alusta
    nelio = luku * luku
    print nelio
    luku = luku + 1 # jos pariton, tänne ei päästä
```