SILMUKAT

Ohjelmoinnin peruskurssi yläkouluun

TURUN YLIOPISTO
Oppimisanalytiikan keskus

TIETOKONEEN TEHOKKUUS

- Tietokoneiden tehokkuus perustuu **nopeuteen**: ne voivat laskea miljoonia (tai miljardeja) laskutoimituksia sekunnissa.
- Yksi ohjelmoijan tärkeimmistä työkaluista onkin silmukat.

SILMUKAT

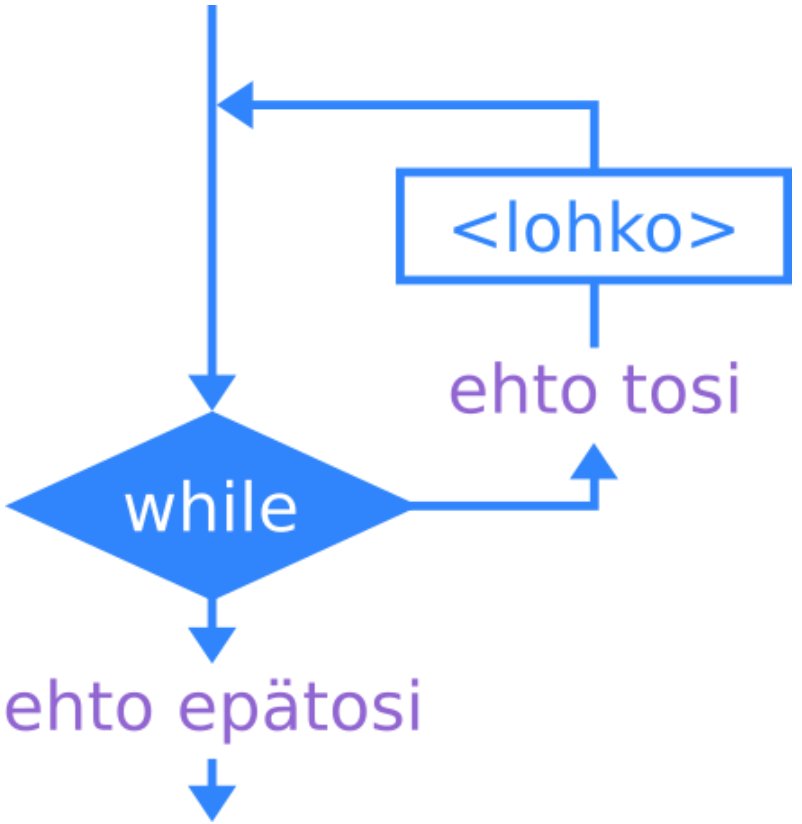
- Silmukoiden avulla voidaan määrittää ohjelmakoodin osa suoritettavaksi **useita kertoja** kirjoittamatta koodia montaa kertaa.
- Silmukoita tarvitaan esimerkiksi **pitkien** tai **tuntemattoman** pituisten rakenteiden läpikäyntiin ja prosessointiin.
- Pythonissa on kaksi silmukkarakennetta: `while` ja `for`.

`while`-SILMUKKA

- `while`-silmukassa määritellään jokin **ehto**, jonka ollessa totta lauseen lohko suoritetaan.
- `while`-silmukan **lohkoa suoritetaan siis niin kauan, kun ehto on tosi**: suoritus päättyy vasta, kun ehto muuttuu epätodeksi tai käyttäjä pysäyttää koko ohjelman suorituksen.

`while`-SILMUKAN SUORITUS

- `while`-silmukan suoritus alkaa ehdon tarkastuksesta: **jos ehto on epätosi, silmukan lohkoa ei suoriteta kertaakaan**.
- Ehto tarkastetaan joka kerta lohkon suorituksen jälkeen.



while-SILMUKAN SYNTAKSI

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

`while`-SILMUKAN EHTO

- `while`-silmukan ehto voi olla **mikä tahansa ehtolauseke**: se voi sisältää vertailu- ja loogisia operaattoreita.
- Jotta silmukan suoritukseen ei jäädä ikuisiksi ajoiksi, on oltava **tarkka siitä, että ehto muuttuu jossain kohtaa epätodeksi**.
- Siksi lohkon sisällä muutetaan usein jotakin arvoa, jota testataan lauseen ehdossa.

ESIMERKKI

```
# Tulostetaan luvut 1-10 näytölle allekkain.

# Alustetaan muuttuja.
n = 1
# Silmukkaa suoritetaan niin kauan, kun n <= 10.
while n <= 10:
    print(n) # Tulostetaan n.
    n = n + 1 # Kasvatetaan n:n arvoa yhdellä.
```

`while`-SILMUKAN OSAT

- Toimiva `while`-silmukka vaatii, että seuraavat kolme osaa on toteutettu:
- **Alustus**: silmukan ehdossa käytettävä(t) muuttuja(t) eli silmukkamuuttuja(t) on alustettu.
- **Ehto**: silmukkamuuttujaa (tai -muuttujia) verrataan johonkin ehtoon.
- **Muutos**: silmukkamuuttujan (tai -muuttujien) arvoja muutetaan silmukan lohkossa jokaisella kierroksella.

IKUINEN SILMUKKA

- Jos jokin silmukan osista on jätetty toteuttamatta, voidaan päätyä ikuiseen silmukkaan.

```
# Alustetaan muuttuja.
n = 1
# Silmukkaa suoritetaan niin kauan, kun n <= 10.
while n <= 10:
    print(n) # Tulostetaan n.
# Koska n:n arvo ei muutu, silmukka ei pääty.
```

SUORITUKSEN PYSÄYTTÄMINEN

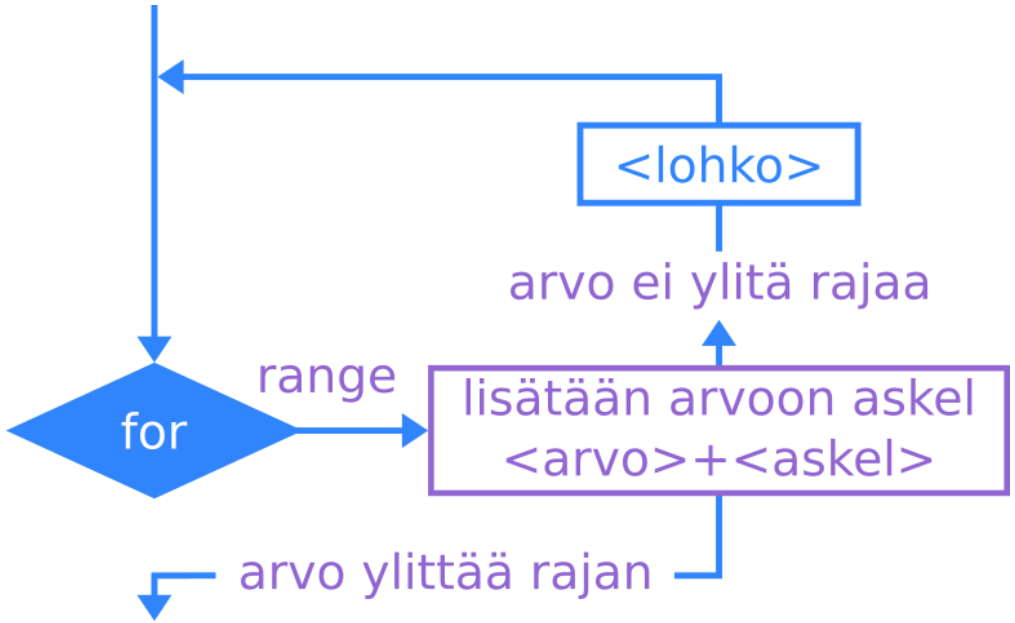
- Ohjelman suoritus voidaan missä tahansa kohtaa ohjelmaa keskeyttää painamalla **CTRL-** ja **C-**näppäimiä yhtä aikaa.
- Näin saadaan myös keskeytettyä ikuisen silmukan suoritus, jos sellaiseen vahingossa joudutaan.

for-SILMUKKA

- Tiettyä lohkoa voidaan toistaa myös for-silmukalla, jonka kanssa käytetään usein range-metodia.
- range-metodin sulkujen sisälle merkitään **alkuarvo**, **loppuarvo** ja **askel**, jonka verran muuttujan arvo muuttuu kierroksen lopussa.
- for-silmukkaa **suoritetaan siis niin monta kertaa kuin alkuarvon ja loppuarvon välissä on askeleen pituisia pätkiä**.

for-SILMUKAN SUORITUS

- for-silmukan suoritus alkaa sillä, että **muuttujan arvoksi** sijoitetaan range-metodissa määritelty **alkuarvo**.
- **Muuttujan arvoa kasvatetaan askeleen verran joka kierroksella**, ja silmukan suoritus loppuu, kun muuttujan arvo kasvaa suuremmaksi kuin loppuarvo.



for-SILMUKAN SYNTAKSI

```
for <muuttuja> in range(<alku>, <loppu>, <askel>):
    <lohko>
```

ESIMERKKI

```
# Tulostetaan luvut 1-10 näytölle allekkain.

# Määritellään range-metodi käymään luvut läpi.
# laitetaan loppuarvoksi 11 koska halutaan tulostaa 10
# ja vertailussa täytyy olla pienempi (eikä yhtä suuri).
for n in range(1,11,1):
    print(n) # Tulostetaan n.
    # range hoitaa arvon muuttamisen.
```

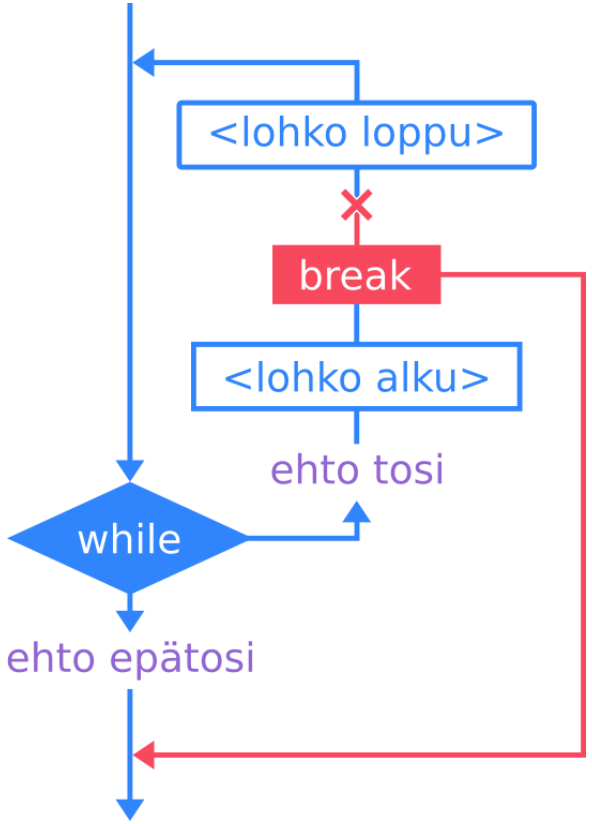
SILMUKAN JA EHTOLAUSEEN YHDISTÄMINEN

- **Silmukoita ja ehtolauseita voi myös käyttää sisäkkäin.**
- Esimerkiksi silmukan sisällä voidaan tarkastaa, onko arvo jaollinen viidellä: jos on, se tulostetaan, ja jos ei, sitä ei tulosteta.
- Silmukoiden ja ehtolauseiden yhdistämisessä **on oltava tarkka lohkojen sisennyksissä**.

```
# Kysytään käyttäjältä merkkijono ja kirjain.
mjono = input("Anna merkkijono:")
kirjain = input("Anna kirjain:")
maara = 0 # Kirjainten määrä merkkijonossa.
for indeksi in range(0, len(mjono), 1):
    if mjono[indeksi] == kirjain: # Onko kirjain?
        # Tämä lohko suoritetaan vain jos kirjaimet ovat samat.
        maara = maara + 1
print("Kirjaimia oli", maara, "kappaletta.")
```

SILMUKAN SUORITUKSEN KATKAISEMINEN

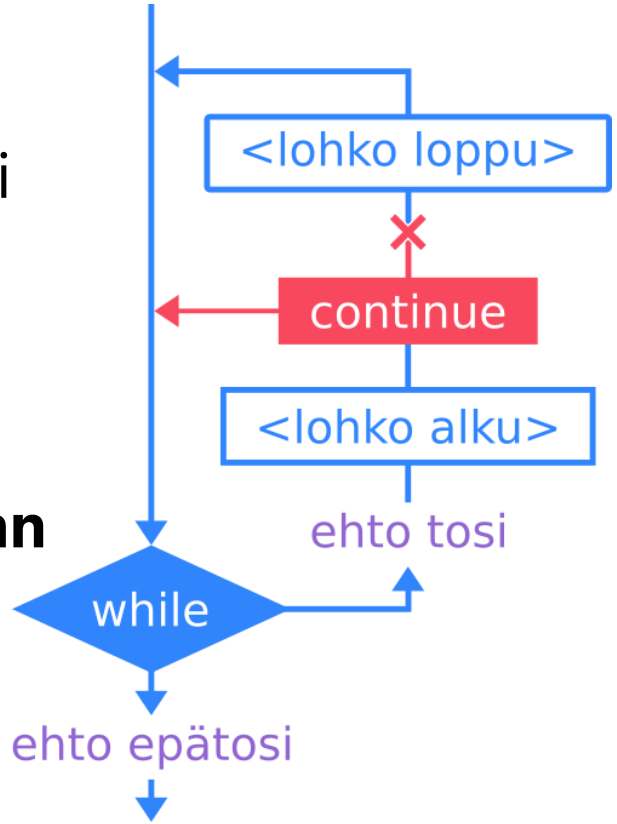
- **Silmukan suoritus voidaan katkaista ohjelmallisesti** käyttämällä `break`-lausetta.
- `break`-lauseen suorituksen jälkeen **ohjelman suoritus jatkuu ensimmäisestä silmukan jälkeisestä lauseesta**, eikä silmukkaan (tai sen lohkoon) palata enää.



```
# Kysytään käyttäjältä lukuja kunnes syöte = 0 ja
# lasketaan lukujen summa.
summa = 0 # Alustetaan summa.
while True: # "Ikuinen" silmukka.
    luku = input("Anna luku tai 0 lopettaaksesi:") # Kysytään luku.
    luku = int(luku) # Muutetaan luku kokonaisluvuksi.
    if luku == 0: # Tarkistetaan, onko luku 0.
        break # Katkaistaan silmukka.
    summa = summa + luku # Lisätään summaan.
# Tulostetaan syötettyjen lukujen summa.
print("Summa:", summa)
```

SILMUKAN SUORITUKSEN SIIRTÄMINEN

- Silmukan suorituksen voi siirtää lohkon suorituksesta **ehdon tarkastukseen** (`while`) tai **muuttujan arvon muuttamiseen** (`for`) `continue`-lauseella.
- `continue`-lauseen suorituksen jälkeen **ohjelman suoritus siis siirtyy silmukan alkuun**, eikä lohkoa suoriteta loppuun sillä kierroksella.



```
mjono = input("Anna merkkijono:")
kirjain = input("Anna kirjain:")
kirjaimia = 0 # Kirjainten määrä merkkijonossa.
muita = 0 # Muiden kirjainten määrä merkkijonossa.
for indeksi in range(0, len(mjono), 1):
    if mjono[indeksi] == kirjain: # Onko kirjain?
        kirjaimia = kirjaimia + 1
        continue # Merkki käsitelty, jatketaan seuraavaan merkkiin.
    muita = muita + 1
print("Kirjaimia", kirjain, "oli", kirjaimia, "ja muita", muita, ".")
```

continue JA IKUINEN SILMUKKA

- continue-lauseen kanssa pitää olla tarkka, ettei tule 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.
```

`continue` VS. `break`

- `break`-lause lopettaa koko silmukan suorituksen kokonaan eli **silmukan lohkoa ei suoriteta enää ollenkaan**.
- `continue`-lause lopettaa vain sen kierroksen, jolla se suoritetaan eli **lohkoa voidaan suorittaa vielä muilla kierroksilla** (jos ehto on tosi `while`-silmukassa tai loppuarvoa ei ole vielä saavutettu `for`-silmukassa).

JAKOJÄÄNNÖSOPERAATTORI

- Eräs Pythonin matemaattisista operaattoreista on **jakojäännösoperaattori** %, jolla on helppo testata arvojen jaollisuutta. Operaattori siis **laskee jakolaskun ja palauttaa sen jakojäännöksen**.
- Esim. 5 % 2 palauttaa 1, koska jakolaskusta 5 / 2 jää yksi yli.

ESIMERKKI

```
# Tulostetaan kahdeksalla jaolliset luvut väliltä 0-100.
for luku in range(0, 101, 1):
    # Testataan jaollisuutta.
    # Jakojäännöksen on siis oltava 0.
    if luku % 8 == 0:
        # Tulostetaan koska oli jaollinen.
        print(luku)
```

TURUN
YLIOPISTO
Oppimisanalytiikan keskus