LUT Python ohjelmointiopas 2023 - osa 4
Sisällysluettelo

Luku 4: Toistorakenteet

Luvun 3 kokoavassa esimerkissä tutustuimme valikkopohjaiseen ohjelmaan. Esimerkissä valikko tulostettiin yhden kerran ja pystyimme suorittamaan vain yhden valikossa näytetyistä valinnoista, joten ohjelman suorituksen jälkeen näytöllä olevaa *Kiitos ohjelman käytöstä.* -kommenttia katsellessa saattoi tulla mieleen, että loppuipas tämä lyhyeen. Tämä johtuu siitä, että luvussa 3 tutustuimme valikkopohjaisen ohjelman valikkoon ja valintarakenteeseen, mutta ohjelman luontevaan käyttöön tarvittava toistorakenne puuttui. Tässä luvussa tutustumme Pythonin tarjoamiin kahteen toistorakenteeseen eli alkuehtoinen ja askeltava toisto, jotka toteutetaan `while` ja `for` -käskyillä. Toistorakenteet tekevät juuri sen minkä lupaavatkin eli mahdollistavat samojen asioiden toistamisen monta kertaa.

Alkuehtoinen toistorakenne while

Yksi tapa toteuttaa toistorakenne on toistaa samaa asiaa siihen asti, että lopetusehto toteutuu. Käytännössä tämä on toteutettu Pythonissa `while`-käskyllä, ehtolausekkeella ja koodilohkolla eli `while`-sanan jälkeen on ehto, jonka ollessa tosi, suoritetaan seuraavan koodilohkon käskyt ja sitten palataan arvioimaan uudestaan ehto eli jatketaanko vai lopetetaanko koodilohkon suorittaminen. Ehtolauseke edellyttää tyypillisesti, että muuttujilla on sopivat arvot ja niiden pitää muuttua, jotta ehtolauseke voi muuttua todesta epätodeksi eli ensin toistorakennetta toistetaan ja sitten toistaminen loppuu.

Esimerkissä 4.1 näkyy kolme tyypillistä tapaa toteuttaa alkuehtoinen toistorakenne `while`. Osassa 1 näkyy toistomäärään perustuva `while`-rakenne, joka halutaan suorittaa nyt 4 kertaa ja rakenteessa on muuttujat toistomäärän laskemiseen sekä maksimimäärää varten, joilla saadaan `while`-rakenteeseen lopetusehto. Osassa 2 lopetusehtona toimii käyttäjän antama syöte. Käyttäjältä pyydetään lukuja ja ilmoitetaan, että luku -1 on lopetusmerkki, jolla tietojen syöttäminen loppuu. Tässä ratkaisussa on oleellista, että lopetusmerkkinä toimiva luku ei kuulu hyväksyttäviin syötteisiin ja siksi se on usein numero 0 tai -1. Osassa 3 lopetusehtona toimii useita ehtoja oppaan luvun 3 mukaisesti. Esimerkissä lopetusehtona toimii syötteiden maksilukumäärä ja toisaalta käyttäjän halu jatkaa tietojen syöttämistä. Tällä kertaa käyttäjältä kysytään erillinen lopetusmerkki, jona toimii kirjaimet ”e” ja ”E” eli jos käyttäjä antaa toisen näistä vastaukseksi jatko-kysymykseen, ohjelma lopetetaan.

Esimerkki 4.1. Alkuehtoinen toistorakenne while

```
# Osa 1. while-rakenteen perustoteutus
Lukumaara = 0                   # Alustus, laskurin nollaus
Maksimi = 4                     # Alustus, maksimin määrittely
while (Lukumaara < Maksimi):    # Lopetusehto, lukumäärä < maksimi
    print(Lukumaara)            # Ohjelman toiminnallinen osuus
    Lukumaara = Lukumaara + 1   # Laskurin kasvatus
print("Kiitos ohjelman käytöstä.") # Ohjelman lopetus

# Osa 2. while-toistorakenteen lopetus lopetusmerkillä
Lukumaara = 0
Summa = 0
Arvosana = int(input("Anna arvosana (-1 lopettaa): "))
while (Arvosana != -1):
    Lukumaara = Lukumaara + 1
    Summa = Summa + Arvosana
    Arvosana = int(input("Anna arvosana (-1 lopettaa): "))
print("Lopetetaan, keskiarvo on", round(Summa / Lukumaara, 2)) #Nollalla jako!
```

```
# Osa 3. while-toistorakenne usealla lopetusehdolla
Lukumaara = 1
Maksimi = 10
Jatka = True
while ((Lukumaara < Maksimi) and (Jatka == True)):
    print("Lukumäärä on", Lukumaara)
    Syote = input("Haluatko jatkaa (k/e): ")
    if ((Syote == "e") or (Syote == "E")):
        Jatka = False
    Lukumaara = Lukumaara + 1
print("Lopetetaan, lukumäärä on", Lukumaara - 1)
```

Tuloste

```
0
1
2
3
Kiitos ohjelman käytöstä.
Anna arvosana (-1 lopettaa): 4
Anna arvosana (-1 lopettaa): 5
Anna arvosana (-1 lopettaa): 6
Anna arvosana (-1 lopettaa): 7
Anna arvosana (-1 lopettaa): 8
Anna arvosana (-1 lopettaa): 9
Anna arvosana (-1 lopettaa): -1
Lopetetaan, keskiarvo on 6.5
Lukumäärä on 1
Haluatko jatkaa (k/e): k
Lukumäärä on 2
Haluatko jatkaa (k/e): e
Lopetetaan, lukumäärä on 2
```

Alkuehtoinen toistorakenne perustuu aiemmin käsiteltyihin ehtolausekkeisiin ja koodilohkoihin, joten ainoa uusi asia on `while`-käsky. `while`-rakenne on joustava eli se mahdollistaa useiden ehtojen arvioinnin lopetusehtona, mutta toisaalta se edellyttää myös tarkkuutta, koska ohjelmoijan on varmistettava lopetusehdon toteutuminen. Jos näin ei käy, on lopputuloksena ikisilmukka eli ohjelma toistaa samaa koodilohkoa eikä pääse eteenpäin. Tällainen ohjelma on virheellinen ja pitää korjata, ja koodia pääsee korjaamaan antamalla syötteeksi CTRL-C eli näppäin Ctrl (tyypillisesti näppäimistön vasen alanurkka) ja C-kirjain samanaikaisesti. Tämä antaa ohjelmalla keskeytyskäskyn, jonka jälkeen ohjelman voi korjata. Huomaa, että osassa 2 ohjelma saattaa kaatua nollalla jakoon, mutta koska nyt keskitytään toistorakenteeseen, on asia huomioitu vain kommentissa.

Askeltava toistorakenne for

Toinen vaihtoehto toistorakenteen toteuttamiseen Pythonissa on `for`-rakenne. `for`-rakenne eroaa `while`-rakenteesta kolmella tavalla. Ensinnäkin `for`-rakenteen kierrosmäärä on tiedettävä suorituksen alkaessa. Toiseksi `for`-rakenteen kanssa ei voi käyttää useita lopetusehtoja. Kolmanneksi `for`-rakennetta voi käyttää monialkioisten rakenteiden kuten listojen läpikäyntiin. Nyt keskitymme `for`-rakenteen käyttämiseen sen perinteisessä muodossa eli silmukoiden tekemiseen ja palaamme muihin tietorakenteisiin myöhemmin.

Esimerkissä 4.2 näkyy kaksi tapaa toteuttaa `for`-rakenne. `for`-rakenne muodostuu neljästä osasta eli avainsana `for`, muuttuja (nyt `i`), avainsana `in` ja lopuksi läpikäytävä sekvenssi. Käytännössä tämä tarkoittaa sitä, että muuttuja `i` käy kaikki sekvenssissä olevat arvot läpi yksi kerrallaan ja toistorakenteen koodilohkossa jokaista arvoa käytetään vuorollaan. Taulukossa 3.1 oli mukana `in`-operaattori ja silloin oli puhetta, että se on joukko-operaattori. Nyt muuttuja `i` käy kaikki seuraavana nimetyn joukon, sekvenssin, arvot vuorotellen.

Osassa 1 näkyvä tapa toteuttaa `for`-rakenne on selkeä, koska kaikki läpikäytävät arvot ovat näkyvillä. Käytännössä tämä on kuitenkin joustamaton tapa toteuttaa toistorakenne, sillä jos läpikäytävä määrä muuttuu esim. kymmenestä sataan, edellyttää se koodin muuttamista ja harva haluaa kirjoittaa sata numeroa koodiin puhumattakaan tilanteesta, jossa toistoja on miljoonia. Siksi osa 2 esittelee `range`-funktion, jolla voidaan luoda `for`-rakenteen tarvitsema sekvenssi joustavasti. Huomaa, että `range`-funktion parametrit ovat samat kuin merkkijonoleikkauksessa: `range(Alku, Loppu, Askel)` vs. `Merkkijono[Alku:Loppu:Askel]`. Esimerkin molemmat osat toimivat samalla tavalla ja tulosteet ovat samat. Osassa 2 näkyvä `range`-funktio luo siis vastaavan listan kuin Osassa 1 näkyy, ts. luvut 1-10.

Esimerkki 4.2. Askeltava toistorakenne for

```
# Osa 1. Askeltava toisto
for i in [1,2,3,4,5,6,7,8,9,10]:
    print(i, end=" ")
print()
print("Kiitos ohjelman käytöstä.")

# Osa 2. Askeltava toistorakenne for range-funktiolla
Alku = 1
Loppu = 11
Askel = 1
for i in range(Alku, Loppu, Askel):
    print(i, end=" ")
print()
print("Kiitos ohjelman käytöstä.")
```

Tuloste

```
1 2 3 4 5 6 7 8 9 10
Kiitos ohjelman käytöstä.
1 2 3 4 5 6 7 8 9 10
Kiitos ohjelman käytöstä.
```

Toistorakenteiden laajennokset

Toistorakenteen idea on toistaa sama koodilohko useita kertoja. Tämä on selkeä lähtökohta, mutta aina kaikki ei mene suunnitellusti ja on pakko muuttaa suunnitelmia. Siksi toistorakenteissakin on mahdollisuus muuttaa suunnitelmia ja se onnistuu kahdella laajennoksella, `break` ja `continue` -käskyillä. Käytännössä nämä ovat hyppykäskyjä, joilla voidaan muuttaa suorituksen kulkua toistorakenteen sisällä eli joko poistua toistorakenteesta kokonaan tai siirtyä seuraavaan toistoon jättämällä loput koodilohkosta suorittamatta. Kannattaa muistaa, että normaali suoritus on tyypillisesti hyvin suunniteltu ja esimerkiksi ohjelman lopetukseen liittyy tietyt rutiinit. Tässä vaiheessa opasta ohjelman lopetusrutiinit muodostuvat lähinnä sopivien rivinvaihtojen ja *Kiitos ohjelman käytöstä.* -tiedotteen tulostuksesta, mutta myöhemmin näitä tarvitaan enemmän. Siksi normaalista suorituksesta poikkeaminen on poikkeus ja poikkeuksesta aiheutuvat muutokset pitää käydä läpi harkiten ongelmien välttämiseksi.

Esimerkissä 4.3 on kaksi toistorakennetta, jotka molemmat toimivat lähtökohtaisesti samalla tavalla. Molemmissa toistorakenteissa on ehdollinen käsky, jonka seurauksena osassa 1 `continue`-käsky siirtää suorituksen seuraavalle kierrokselle, kun taas osassa 2 `break`-käsky johtaa toistorakenteesta poistumiseen kesken sen suorituksen.

Esimerkki 4.3. Toistorakenteiden laajennokset

```
# Osa 1. Toistorakenteiden laajennos continue
for i in range(1, 21):
    if ((i % 10) == 0): # jos luku jaollinen 10:llä, continue
        continue
    print(i, end=" ")
print()
print("Kiitos ohjelman käytöstä.")

# Osa 2. Toistorakenteiden laajennos break
i = 1
while (i < 21):
    if ((i % 10) == 0): # jos jaollinen 10:llä, break
        break
    print(i, end=" ")
    i = i + 1
print()
print("Kiitos ohjelman käytöstä.")
```

Tuloste

```
1 2 3 4 5 6 7 8 9 11 12 13 14 15 16 17 18 19
Kiitos ohjelman käytöstä.
1 2 3 4 5 6 7 8 9
Kiitos ohjelman käytöstä.
```

Tyypillisiä sovelluksia

Toistorakenteita käytetään ohjelmoinnissa jatkuvasti, sillä tietokone ja ohjelmat sopivat hyvin suurten tietomäärien käsittelyyn juuri toistorakenteen asiasta eli samat operaatiot suoritetaan suurelle joukolle tietoja. Tässä osiossa on kolme tyypillistä toistorakenteiden sovellustyyppiä eli näistä löytyy hyvä lähtökohta monien ohjelmointitehtävien ratkaisemiseen.

Ohjelmat käsittelevät usein tietoa, jotka kuuluvat jollain tavalla yhteen eli muodostavat yhden joukon. Esimerkki 4.4 osassa 1 on lyhyt esimerkki `in`-operaattorin käytöstä merkkijonon läpikäynnissä. Kuten aiemmin todettiin, `for`-lauseen muuttuja käy läpi `in`-operaattorilla sekvenssin arvot. Käytännössä muuttuja `Kirjain` saa kaikki sekvenssin eli merkkijonon ”Apina” merkit vuorotellen ja kohdistaa niihin halutun operaation, joka on nyt merkin tulostus riville. Vaikka tätä nimenomaista ratkaisua ei tarvinne usein, tämä rakenne toistuu jatkossa usein ja siksi tähän kannattaa palata myöhemmin aina tarvittaessa.

Esimerkin 4.4 osassa 2 lasketaan arvosanojen tilastotietoja. Koska nyt halutaan monenlaista tietoa, tarvitaan useita muuttujia ja tässä esimerkissä näkyy, että ne kaikki kannattaa alustaa sopivilla arvoilla ennen niiden käyttöä ongelmien välttämiseksi. Esimerkiksi `Lukumaara` ja `Summa` -muuttujia käytetään koodissa suoraan sijoitusmerkin oikealla puolella, jolloin ne tulee alustaa arvolla 0. `Pienin` ja `Suurin` -muuttujat pitää alustaa jollain, mutta etsittäessä suurinta/pienintä arvoa, yleisesti sopivaa alkuarvoa ei ole. Siksi ne on alustettu `None`-arvolla eli niillä ei ole vielä arvoa. Kun käyttäjä on syöttänyt ensimmäisen arvon, voidaan nämä muuttujat alustaa sillä ja vaihtaa sitä tarvittaessa. Tällä tavoin ohjelma löytää varmasti käyttäjän antamista arvoisa pienimmän ja suurimman. Tässä ohjelmassa nollalla jako on estetty, koska ohjelma demonstroi tilastotietojen laskentaa ja silloin syötteiden puute on huomioitava poikkeustapaus.

Esimerkki 4.4. Tyypillinen sovellus: joukon käsittely

```
# Osa 1. Joukon läpikäynti for-in rakenteella
for Kirjain in "Apina":
    print(Kirjain)
print()

# Osa 2. Tilastotietojen selvittäminen
Lukumaara = 0
Summa = 0
Keskiarvo = None
Pienin = None
Suurin = None
Arvosana = int(input("Anna arvosana (-1 lopettaa): "))

while (Arvosana != -1):
    Lukumaara = Lukumaara + 1
    Summa = Summa + Arvosana
    if (Pienin == None):
        Pienin = Arvosana
        Suurin = Arvosana
    elif (Pienin > Arvosana):
        Pienin = Arvosana
    elif (Suurin < Arvosana):
        Suurin = Arvosana
    Arvosana = int(input("Anna arvosana (-1 lopettaa): "))
print()

if (Lukumaara > 0):
    Keskiarvo = Summa / Lukumaara
    print("Lukumäärä on", Lukumaara)
    print("Summa on", Summa)
    print("Keskiarvo on", round(Keskiarvo, 2))
    print("Pienin on", Pienin)
    print("Suurin on", Suurin)
else:
    print("Et antanut yhtään arvosanaa, tilastotietoja ei voi laskea.")
print("Kiitos ohjelman käytöstä.")
```

Tuloste

```
A
p
i
n
a

Anna arvosana (-1 lopettaa): 7
Anna arvosana (-1 lopettaa): 8
Anna arvosana (-1 lopettaa): 7
Anna arvosana (-1 lopettaa): 7
Anna arvosana (-1 lopettaa): -1

Lukumäärä on 4
Summa on 29
Keskiarvo on 7.25
Pienin on 7
Suurin on 8
Kiitos ohjelman käytöstä.
```

Esimerkissä 4.5 näkyy monimutkaisten ehtojen käsittely toistorakenteessa. Esimerkissä etsitään alkulukuja tietyltä lukuväliltä tunnistamalla luvut, jotka saadaan aiempien lukujen tulona. Tämä perustuu jaollisuustestiin, jota käytettiin myös parillisuuden selvittämisessä. Tällaisten monimutkaisten ehtojen kohdalla on usein järkevintä tehdä testit

valintarakenteessa ja hyödyntää `break`-käskyä toistorakenteista poistumiseen. Tiivistetysti esimerkissä on kaksi päällekkäistä toistorakennetta, sisemmästä silmukasta poistutaan tarvittaessa `break`-käskyllä, ja lisäksi käytössä on yksisuuntainen lippu. Lippu alustetaan `False`:ksi joka kierroksella, jonka jälkeen mahdollinen jaollisuus tallennetaan siihen ja lopuksi sillä ohjataan tulostusta.

Esimerkki 4.5. Tyypillinen sovellus: monimutkaiset ehdot

```
MAX = 10
for Luku in range(2, MAX):
    JakajaLoytyi = False
    for Jakaja in range(2, Luku):
        if ((Luku % Jakaja) == 0):
            print(Luku, "ei ole alkuluku, sillä se muodostuu tulosta",
                        Jakaja, "*", Luku // Jakaja)
            JakajaLoytyi = True
            break
    if (JakajaLoytyi == False):
        print(Luku, "on alkuluku")
print()
```

Tuloste

```
2 on alkuluku
3 on alkuluku
4 ei ole alkuluku, sillä se muodostuu tulosta 2 * 2
5 on alkuluku
6 ei ole alkuluku, sillä se muodostuu tulosta 2 * 3
7 on alkuluku
8 ei ole alkuluku, sillä se muodostuu tulosta 2 * 4
9 ei ole alkuluku, sillä se muodostuu tulosta 3 * 3
```

Kolmas tyypillinen toistorakenteiden sovellus on taulukon tulostus, joka on esitetty Esimerkissä 4.6. Taulukko koostuu riveistä ja sarakkeista, joten sen tulostamiseen tarvitaan kaksi sisäkkäistä toistorakennetta. Ulompi toistorakenne käy läpi eri rivit, kun taas sisempi rakenne tulostaa aina yhden rivin sarakkeet kerralla. Huomaa, että tulosteessa pitää olla tarkkana rivinvaihtojen kanssa ja yhden rivin sarakkeiden tulostus on helpointa `print`-käskyn `end`-parametria hyödyntäen. Tähänkin palataan tarkemmin myöhemmin, sillä taulukkolaskentasovellukset perustuvat riveihin ja sarakkeisiin, joten tällä rakenteella voi sekä kirjoittaa että lukea taulukkolaskentaohjelmien käyttämää tietoa.

Esimerkki 4.6. Tyypillinen sovellus: taulukon eli matriisin tulostus riveille ja sarakkeille

```
# Esimerkki 4.
Riveja = int(input("Anna tulostettavien rivien määrä: "))
Sarakkeita = int(input("Anna tulostettavien sarakkeiden määrä: "))

for i in range(1, Riveja+1):
    for j in range(1, Sarakkeita+1):
        print(i*j, end="\t")
    print()
print()
```

Tuloste

```
Anna tulostettavien rivien määrä: 3
Anna tulostettavien sarakkeiden määrä: 5
1     2     3     4     5
2     4     6     8     10
3     6     9     12    15
```

Toistorakenteiden perustyypit

Toistorakenteista on olemassa paljon erilaisia toteutuksia, koska niitä voidaan soveltaa moniin erilaisiin tarkoituksiin. Esimerkissä 4.7 näkyy tyypilliset toistorakenteet, joista yleensä löytyy sopiva lähtökohta kaikkiin tarpeisiin. Valitsemalla suoraan sopivan rakenteen, ohjelma toimii alusta alkaen pääpiirteissään oikealla tavalla ja tarvittavat muokkaukset jäävät usein vähäisiksi. Tyypillisesti toistorakenne on nopein toteuttaa osan 1 askeltavana toistona eli `for`-lause `range`-funktiolla. Sama toiminnallisuus saadaan `while`-rakenteella, mutta siitä tulee tyypillisesti pidempi (Osa 2). Käyttäjän antamaan lopetusmerkkiin perustuva toistorakenne tulee toteuttaa `while`-rakenteella (Osa 3), koska toistojen määrä ei ole etukäteen tiedossa. Samoin useat lopetusehdot on toteutettava `while`-rakenteella (Osa 4).

Esimerkki 4.7. Toistorakenteiden perustyypit

```
# Osa 1.
print("Askeltava toisto for ja range-funktio:")
for i in range(4):
    print(i)
print()

# Osa 2.
print("Alkuehtoinen toisto while, lopetusehtona laskurin tavoitearvo")
i = 0
while (i < 4):
    print(i)
    i = i + 1
print()

# Osa 3.
print("Alkuehtoinen toisto while, lopetusehtona käyttäjän lopetusmerkki")
Luku = int(input("Anna luku (-1 lopettaa): "))
while (Luku != -1):
    print(Luku)
    Luku = int(input("Anna luku (-1 lopettaa): "))
print()

# Osa 4.
print("Alkuehtoinen toisto while, useita lopetusehtoja")
MAX = 2
Lukumaara = 0
Luku = int(input("Anna luku (-1 lopettaa): "))
while ((Luku != -1) and (Lukumaara < MAX)):
    print(Luku)
    Luku = int(input("Anna luku (-1 lopettaa): "))
    Lukumaara = Lukumaara + 1
print("Kiitos ohjelman käytöstä.")
```

Tuloste

```
Askeltava toisto for ja range-funktio:
0
1
2
3

Alkuehtoinen toisto while, lopetusehtona laskurin tavoitearvo
0
1
2
3
```

```
Alkuehtoinen toisto while, lopetusehtona käyttäjän lopetusmerkki
Anna luku (-1 lopettaa): 1
1
Anna luku (-1 lopettaa): 2
2
Anna luku (-1 lopettaa): 3
3
Anna luku (-1 lopettaa): -1

Alkuehtoinen toisto while, useita lopetusehtoja
Anna luku (-1 lopettaa): 1
1
Anna luku (-1 lopettaa): 2
2
Anna luku (-1 lopettaa): 3
Kiitos ohjelman käytöstä.
```

Muuttujien roolit: tuoreimman säilyttäjä, askeltaja, kokooja

Toistorakenteissa monilla muuttujilla on vakiintuneet roolit. Tästä hyvä esimerkki on alkuehtoinen toisto, jossa käyttäjältä kysytään syöte, joka perusteella toistorakenne päättyy. Kuten näimme esimerkin 4.1 osassa 2, tämän `while`-toistorakenteen lopetus perustuu käyttäjän antamaan lopetusmerkkiin eli käyttäjältä pyydettiin arvosana ja kerrottiin, että arvo -1 lopettaa toistorakenteen. Tämä muuttuja on tyypillinen *tuoreimman säilyttäjä*. Olemme käyttäneet roolia jo aiemmin, kun käyttäjältä kysyttiin toistuvasti nimiä tms.

Toinen tyypillinen toistorakenteeseen liittyvä muuttuja on *askeltaja*, sillä askeltava toisto, `for`, perustuu siihen. Askeltaja etenee ja muuttuu etukäteen ilmoitetulla tavalla, josta tyypillisin esimerkki on lukujen 1-N läpikäynti. Pythonissa näiden sekvenssien luomiseen on oma funktio `range`, jolla läpikäytävän joukon/sekvenssin luominen on helppoa, ks. esimerkki 4.2 osa 2.

Tässä luvussa uutena muuttujan roolina oli esimerkin 4.4 osassa 2 käytetty `Summa`-muuttuja, johon laskettiin kaikkien käyttäjän antamien lukujen summa, ts. kyseessä oli *kokooja*-muuttuja. Kokoojaan yhdistetään joukon kaikki arvot, joten tyypillisesti muuttuja alustetaan nollaksi, jotta siihen voidaan lisätä kaikki joukon arvot yksi kerrallaan toistorakenteessa. Usein kokooja-muuttujan arvoa käytetään esimerkiksi keskiarvon laskentaan, niin kuin tässäkin esimerkissä tehtiin.

Yhteenveto

Osaamistavoitteet

Tämän luvun keskeiset asiat on nimetty alla olevassa listassa. Tarkista sen avulla, että tunnistat nämä asiat ja sinulla on käsitys siitä, mitä ne tarkoittavat. Mikäli joku käsite tuntuu oudolta, käy siihen liittyvät asiat uudestaan läpi. Nämä käsitteet ja käskyt on hyvä muistaa ja tunnistaa, sillä seuraavaksi teemme niihin perustuvia ohjelmia.

- Alkuehtoinen toisto `while`: perusrakenne, lopetusmerkki, monta ehtoa (E4.1)
- Askeltava toisto `for`: perusrakenne, `range` (E4.2)
- Toistorakenteiden laajennokset: `break` ja `continue` (E4.3)
- Tyypillisiä sovelluksia
    - Joukon käsittely: `in`, lukumäärä, summa, keskiarvo, pienin/suurin arvo (E4.4)
    - Monimutkaiset ehdot (E4.5)
    - Taulukon eli matriisin tulostus riveille ja sarakkeille (E4.6)

- Toistorakenteiden perustyypit (E4.7)
- Muuttujien roolit: tuoreimman säilyttäjä, askeltaja, kokooja
- Valikkopohjainen ohjelma toistolla (E4.8)

Huomaa, että tilastotietojen laskenta ja pienimmän/suurimman arvon selvittäminen ovat tyypillisiä algoritmeja tällä kurssilla. Näitä tietoja voi selvittää muillakin tavoilla, mutta aina ratkaisut eivät ole yleisesti toimivia vaan toimivat vain tietyillä reunaehdoilla. Edellä käsitellyt ratkaisut/algoritmit on sovitettu tällä kurssilla käytettyihin tietojoukkoihin ja siten suositeltavia tällä kurssilla.

Pienen Python-perusohjelman tyyliohjeet

Tähän lukuun liittyvät tyyliohjeet on koottu alle.

1. Askeltavaa `for`-toistorakennetta tulee käyttää, kun tiedetään läpikäytävä sekvenssi. Sekvenssi luodaan tyypillisesti `range`-funktiolla
2. Alkuehtoista `while`-toistorakennetta tulee käyttää, kun läpikäytävien tietojen lukumäärä ei ole tiedossa etukäteen tai lopetusehtoja on useita
3. Lähtökohtaisesti toistorakenteen tulee päättyä normaalisti, jotta sen jälkeiset lopetusrutiinit voi sijoittaa yhteen koodilohkoon ja suorittaa hallitusti. Normaali lopetus tarkoittaa, että kaikki läpikäytävät arvot on käsitelty
4. Toistorakenteiden askeltaja-muuttuja on usein kirjan `i` – tai `i` ja `j`, jos askeltajia on kaksi. Myös kuvaavat muuttujanimet kuten `Ika` ja `Lukumaara` ovat hyviä
5. Yksi valikkopohjaisen ohjelman standardifraasi muuttuu toistorakenteen myötä:
   a. `"Tuntematon valinta, yritä uudestaan."`
6. Valikkopohjaisen ohjelman valintarakenteen jälkeen toistorakenteen viimeinen käsky on tyhjän rivin tulostus
7. Ohjelmassa käsiteltävät muuttujat tulee määritellä ja alustaa ohjelman alussa. Tyypillisiä alustusarvoja ovat 0 ja `None`. `None` alkuarvosta näkee, ettei muuttujalla ole vielä oikeaa arvoa
8. Etsittäessä datasta tiettyjä arvoja kuten pienin tai suurin, tulee sopivimman säilyttäjä -muuttuja(t) alustaa läpikäytävien tietojen ensimmäisen alkion arvolla
9. Käskyjen `break` ja `continue` jälkeen samassa koodilohkossa ei saa olla käskyjä, koska niitä ei suoriteta
10. Tässä oppaassa ei käytetä toistorakenteiden else-haaraa eikä enumerate()-funktiota

Pythonissa ei ole erillistä loppuehtoista toistorakennetta, vaikka monissa muissa ohjelmointikielissä sellainen on. Tämä pystytään toteuttamaan valintarakenteella ja `break`-käskyllä, joten sen toteutus onnistuu, kuten monimutkaisten ehtojen kohdalla nähtiin. `break`-käskyn käyttöä ei kuitenkaan suositella tällä kurssilla niin kuin edellä lukee.

Luvun asiat kokoava esimerkki

Esimerkissä 4.8 on valikkopohjainen ohjelma eli esimerkin 3.9 valikkopohjaisen ohjelman ideasta laajennettu versio, jossa on valikko- ja valintarakenteen lisäksi toistorakenne. Tällä rakenteella päästään jo pitkälle, mutta jatkossa tämäkin rakenne muuttuu ja kehittyy lisää.

Esimerkki 4.8. Valikkopohjainen ohjelma

```
# 20230922 un E04_8.py (c) LUT
############################################################
# Muuttujien alustukset
Valinta = 1

# Ohjelman toiminnallinen osuus
while (Valinta != 0):
    # Valikko
    print("Valitse haluamasi toiminto:")
    print("1) Anna merkkijono")
    print("2) Tulosta merkkijono etuperin")
    print("3) Tulosta merkkijono takaperin")
    print("0) Lopeta")
    Valinta = int(input("Anna valintasi: "))
    print()

    # Valintarakenne
    if (Valinta == 1):
        Merkkijono = input("Anna merkkijono: ")
    elif (Valinta == 2):
        print(Merkkijono)
    elif (Valinta == 3):
        print(Merkkijono[::-1])
    elif (Valinta == 0):
        print("Lopetetaan.")
    else:
        print("Tuntematon valinta, yritä uudestaan.")
    print()

# Ohjelman lopetus
print("Kiitos ohjelman käytöstä.")
############################################################
# eof
```

Tuloste

```
Valitse haluamasi toiminto:
1) Anna merkkijono
2) Tulosta merkkijono etuperin
3) Tulosta merkkijono takaperin
0) Lopeta
Anna valintasi: 1

Anna merkkijono: Brian on teekkari.

Valitse haluamasi toiminto:
1) Anna merkkijono
2) Tulosta merkkijono etuperin
3) Tulosta merkkijono takaperin
0) Lopeta
Anna valintasi: 3

.irakkeet no nairB

Valitse haluamasi toiminto:
1) Anna merkkijono
2) Tulosta merkkijono etuperin
3) Tulosta merkkijono takaperin
0) Lopeta
Anna valintasi: 0

Lopetetaan.

Kiitos ohjelman käytöstä.
```