LUT Python ohjelmointiopas 2023 - osa 5
Sisällysluettelo

Luku 5: Ohjelman rakenne

Olemme tähän mennessä kirjoittaneet ohjelmia, joissa kaikki koodi on päätasolla lukuun ottamatta valinta- ja toistorakenteisiin liittyviä sisennettyjä koodilohkoja. Tämä on yksi Pythonin vahvuuksia eli ohjelmia voi kirjoittaa nopeasti ilman kattavaa suunnittelua. Ohjelmien koon kasvaessa tilanne kuitenkin muuttuu ja ohjelmien rakenteen suunnittelu tulee tärkeäksi osaksi ohjelman tekemistä. Tilanne vastaa huoneiston suunnittelua eli yksiössä on lähtökohtaisesti yksi huone, jossa kaikki huoneistoon liittyvät toiminnot eli oleskelu, nukkuminen, opiskelu/työ ja ruuanlaitto sekä syöminen tehdään samassa tilassa. Tämä toimii noin 20m2 huoneessa, vaikka niissäkin on tyypillisesti pesuhuone/WC, jota ei lasketa huoneeksi. Mutta huoneiston koon moninkertaistuessa ruvetaan toimintoja eriyttämään eri tiloihin ja suunnittelemaan erilaisia huoneita. Tähän asti tekemämme ohjelmat ovat olleet pieniä ja nyt kun ohjelmat rupeavat kasvamaan, pitää meidän ruveta suunnittelemaan niiden rakennetta tarkemmin.

Python-ohjelmoinnin keveyttä ja ketteryyttä demonstroi hyvin se, että minimaalinen ohjelma voi olla yksi `print`-lause. Pythonin joustavuus mahdollistaa sen opiskelun vaiheittain ja höydyllisten ohjelmien tekemisen minimaalisella koodimäärällä. Ohjelmien kasvaessa niiden luettavuus, selkeys ja ylläpidettävyys tulevat tärkeämmäksi kuin nopea toteutus, ja kun ohjelmointikieli ei pakota näitä ominaisuuksia, jää niiden toteutus ohjelmoijan vastuulle (vrt. vakio-kiintoarvo –keskustelu aiemmin). Tärkeimpiä ominaisuuksia luettavuuden ja ymmärrettävyyden kannalta on selkeä ohjelmarakenne, jota tavoitellessa keskeinen asia on jakaa ohjelma pieniin selkeisiin kokonaisuuksiin eli aliohjelmiin. Kun aliohjelman toimintaperiaatteet ymmärtää, ei sen sisäisestä toteutuksesta tarvitse välittää vaan voi keskittyä sen käyttämiseen, niin kuin olemme tehneet mm. aliohjelmien `print`, `input` ja `len` kanssa. Toisaalta Python tulkitsee päätasolle kirjoitetun eli sisentämättömän koodin pääohjelmaksi, joten sekä pääohjelma että aliohjelmat ovat meille tuttuja käyttäjän näkökulmasta. Tässä luvussa tutustumme tarkemmin ohjelman rakenteeseen ja pääohjelman sekä aliohjelmien tekemiseen.

Pääohjelma

Yksi Pythonin keino nopeuttaa ohjelmien tekemistä on tulkita päätasolla oleva koodi suoraan monissa muissa ohjelmointikielissä olevana pääohjelma-rakenteena. Esimerkiksi C-kielessä on `main()`-ohjelma, joka on ainoa pakollinen funktio ohjelmassa ja jota käyttöjärjestelmä kutsuu ohjelmaa käynnistettäessä. Pythonin ratkaisu toimii hyvin niin kauan kuin päätasolla oleva koodin määrä vastaa tyypillistä aliohjelmaa eli kun se mahtuu yhdellä kertaa näytölle. Tätä isompien ohjelmien kohdalla riski virheisiin kasvaa ja siksi suositaan lyhyitä ohjelmia, joka tarkoittaa ohjelman muodostamista pääohjelmasta ja sitä tukevista aliohjelmista.

Esimerkissä 5.1 näkyy tähän asti tyypillinen ohjelma, jossa kaikki koodi on päätasolla. Tässä 5 rivin ohjelmassa tämä ratkaisu toimii hyvin ja tämä ohjelma kannattaa toteuttaa näin.

Esimerkki 5.1. Ohjelman rakenne oppaan luvuissa 1-4, kaikki koodi päätasolla

```
print("Ohjelma yhdessä putkessa.")          # Tulosteita
Vuosi = int(input("Mikä vuosi nyt on: "))  # Toiminnallinen osuus
print("Nyt on vuosi", Vuosi)                # Tulosteita
print("Kiitos ohjelman käytöstä.")          # Lopetus
print()
```

Tuloste

```
Ohjelma yhdessä putkessa.
Mikä vuosi nyt on: 2023
Nyt on vuosi 2023
Kiitos ohjelman käytöstä.
```

Esimerkissä 5.2 on esimerkin 5.1 kaikki koodirivit sisennetty ja tälle koodilohkolle on annettu nimeksi `paaohjelma`. Nimetty koodilohko alkaa `def`-sanalla, sitä seuraa koodilohkolle annettu nimi, nyt `paaohjelma`, jonka jälkeen on sulut sekä kaksoispiste. Normaalin koodilohkon tyyliin kaikki siihen kuuluvat rivit on sisennetty samalle tasolle ja nimetyn koodilohkon viimeisenä rivinä on `return`-käsky. Tällä kertaa `return`-käskyä seuraa `None`-avainsana, joka tarkoittaa sitä, ettei tämä ohjelma palauta mitään paluuarvoa. Vastaavasti `def`-rivillä oli tyhjät sulut, jotka kertovat, ettei tähän ohjelmaan välitetty tietoa eli se ei saanut parametrejä. Nämä tiedonvälitykseen liittyvät parametrit ja paluuarvo käsitellään myöhemmin ja nyt keskitymme nimettyyn koodilohkoon. Nimetty koodilohko tarkoittaa edellä määriteltyä koodilohkoa, joka alkoi sanalla `def` (define), ja jos tämä koodilohko halutaan suorittaa, pitää sitä kutsua erikseen. Nimetty koodilohko suoritetaan kirjoittamalla sen nimi koodiin ja tässä tapauksessa nimi `paaohjelma()` kirjoitetaan päätasolle.
Esimerkin 5.2 suorittaminen tarkoittaa sitä, että tulkki käy tiedoston koodirivit läpi alkaen rivistä 1 ja toteaa, että siinä on koodilohkon määrittely ja merkitsee sen omaan kirjanpitoon. Seuraavaksi tulkki löytää `paaohjelma()`-kutsun ja katsoo kirjanpidostaan missä se on, sekä suorittaa yllä määritellyn koodilohkon. Ohjelman suoritus siirtyy tässä vaiheessa `def paaohjelma():` -riville, jonka jälkeen tähän koodilohkoon kuuluvat käskyt suoritetaan normaalisti rivi kerrallaan `return`-käskyyn asti. `return`-käskyn kohdalla hypätään takaisin koodilohkon kutsuun eli päätasolle. Koska `paaohjelma()`-kutsu oli ohjelman viimeinen rivi, loppuu ohjelman suoritus siihen.

Esimerkki 5.2. Nimetty koodilohko paaohjelma, alkaa def ja loppuu return

```
def paaohjelma():
    print("Koko ohjelma muutettuna pääohjelmaksi.")
    Vuosi = int(input("Mikä vuosi nyt on: "))
    print("Nyt on vuosi", Vuosi)
    print("Kiitos ohjelman käytöstä.")
    print()
    return None

paaohjelma()
```

Tuloste

```
Koko ohjelma muutettuna pääohjelmaksi.
Mikä vuosi nyt on: 2023
Nyt on vuosi 2023
Kiitos ohjelman käytöstä.
```

Tässä oppaassa pääohjelman nimenä on aina `paaohjelma()` C-kielen `main`-ohjelman mukaisesti. Kuten esimerkeistä 5.1 ja 5.2 näkyy, pääohjelman muodostaminen toimivasta ohjelmasta on helppoa, sillä kaikki koodi sisennetään, koodilohkon alkuun kirjoitetaan ohjelman nimi `def`-käskyn perään ja koodilohkon viimeiseksi riviksi tulee `return None` -käsky. Tämän jälkeen tarvitaan vielä ohjelman kutsu eli päätasolle koodilohkon nimi. Vaikka tässä nimenomaisessa tapauksessa tästä muutoksesta ei ole mitään hyötyä, olemme nyt saaneet määriteltyä nimetyn koodilohkon ja kutsuttua sitä eli tämä rakenne mahdollistaa nimettyjen koodilohkojen luomisen ja kutsumisen.

Aliohjelma

Edellä tutustuimme pääohjelmaan, joten seuraava vaihe on tutusta aliohjelmiin. Pääohjelman lähtökohtana oli se, että käyttöjärjestelmä kutsuu sitä ja sen jälkeen pääohjelma kutsuu ohjelman muita nimettyjä koodilohkoja eli aliohjelmia. Aliohjelmia voi olla useita, kun taas pääohjelmia voi olla vain yksi. Tähän mennessä usein käytettyjä aliohjelmat ovat esim. `input`, `print`, `int`, `float`, `str` ja `len`. Tässä luvussa käymme läpi, miten aliohjelmia tehdään ja jätämme toisen keskeisen asian eli tiedonsiirron aliohjelmaan ja sieltä takaisin seuraavaan lukuun.

Esimerkin 5.3 Osassa 1 näkyy tähän asti tyypillinen ohjelma, joka laskee keskiarvon käyttäjän niin halutessa. Lisätään tähän ohjelmaan pää-/aliohjelmarakenne tunnistamalla näihin koodilohkoihin kuuluvat asiat ja määrittelemällä niitä vastaavat nimetyt koodilohkot. Osan 1 koodia katsoessa siitä voi tunnistaa kahteen eri asiaan liittyvää koodia: ensinnäkin käyttäjäinteraktioita eli ensimmäisellä rivillä kysytään, haluaako käyttäjä laskea keskiarvon ja viimeisellä rivillä kiitetään käyttäjää ohjelman käytöstä. Toinen osa koodia on ehdollinen koodilohko, jossa lasketaan keskiarvo, jos käyttäjä niin halusi. Tältä pohjalta ohjelmassa on siis pääohjelma `paaohjelma`, joka hoitaa käyttäjäinteraktion ja jos käyttäjä niin haluaa, se kutsuu keskiarvon laskevaa aliohjelmaa. Ja toinen nimetty koodilohko ohjelmassa on aliohjelma `keskiarvo`, joka laskee keskiarvon sekä tulostaa tuloksen näytölle. Tällä tavoin muodostettu ohjelma näkyy Osassa 2, jossa on ensin aliohjelma `keskiarvo`, sitten pääohjelma `paaohjelma` ja viimeisenä rivinä pääohjelman kutsu, joka käynnistää ohjelman.

Esimerkki 5.3. Aliohjelmarakenteen muodostaminen koodista

```
# Osa 1. Koodia valinta- ja toistorakenteilla
Haluan = input("Haluatko laskea keskiarvon (k/e): ")
if ((Haluan == "k") or (Haluan == "K")):
    Summa = 0
    Lukumaara = int(input("Montako lukua annat: "))
    for i in range(Lukumaara):
        Summa = Summa + int(input("Anna luku: "))
    print("Keskiarvo on", Summa / Lukumaara)
print("Kiitos ohjelman käytöstä.")

# Osa 2. Pääohjelma-aliohjelma -rakenne ja pääohjelman kutsu
def keskiarvo(): # Keskiarvon laskentaan keskittyvä koodi
    Summa = 0
    Lukumaara = int(input("Montako lukua huomioidaan: "))
    for i in range(Lukumaara):
        Summa = Summa + int(input("Anna luku: "))
    print("Keskiarvo on", Summa / Lukumaara)
    return None

def paaohjelma(): # Ohjelman suorituksen hallintaan keskittyvä koodi
    Haluan = input("Haluatko laskea keskiarvon (k/e): ")
    if ((Haluan == "k") or (Haluan == "K")):
        keskiarvo()
    print("Kiitos ohjelman käytöstä.")
    return None

paaohjelma() # Päätasolla oleva pääohjelman kutsu käynnistää ohjelman
```

Tuloste

```
Haluatko laskea keskiarvon (k/e): k
Montako lukua annat: 3
Anna luku: 1
Anna luku: 2
Anna luku: 3
Keskiarvo on 2.0
Kiitos ohjelman käytöstä.
Haluatko laskea keskiarvon (k/e): e
Kiitos ohjelman käytöstä.
```

Hyvän ohjelmointityylin mukaisessa isommassa Python-ohjelmassa kaikki koodi kirjoitetaan siis erillisiin aliohjelmiin ja päätasolla on vain kutsu pääohjelmaksi nimettyyn funktioon. Päätasolla on `paaohjelma()`-kutsun lisäksi kaikkien aliohjelmien määrittelyt (`def …():`) sekä kiintoarvojen määrittelyt sekä muita myöhemmin oppaassa mainittavia globaalisti määriteltäviä rakenteita.

Alla on esimerkki 5.6 hyvän ohjelmointityylin mukaisesta minimaalisesta ohjelmatiedoston ja ohjelman rakenteesta. Huomaa, että merkittävä osa ohjelmasta liittyy aliohjelmien määrittelyihin ja kirjoittamalla vain itse toiminnallisuuden päätasolle ohjelma olisi lyhempi ja nopeampi tehdä. Siksi tämä ohjelmarakenne sopii isompiin, kasvaviin ohjelmiin, joissa halutaan varmistaa ohjelman ymmärrettävyys ja ylläpidettävyys muutoksista ja pitkästä käyttöiästä huolimatta. Näin ollen tämänkin oppaan tulevissa esimerkeissä palataan tyypillisesti lyhyeen päätasolle kirjoitettavaan koodiin, jotta esimerkeissä voidaan keskittyä uuteen asiaan ja tarpeen tullen tämän isoon ohjelmaan sopivan ohjelmarakenteen voi tarkistaa täältä.

Pää- ja aliohjelmiin perustuva ohjelmarakenne

Nimetty koodilohko mahdollistaa ohjelman jakamisen itsenäisiin osiin. Näitä osia voidaan kutsua moduuleiksi tai komponenteiksi, ja ohjelma voidaan muodostaa näitä yhdistelemällä vastaavalla tavalla kuin Lego-palikoita. Ohjelmoinnissa näitä moduuleja kutsutaan pää- ja aliohjelmiksi. Pääohjelma on aina ohjelmakohtainen ja se keskittyy aliohjelmien kutsumiseen sopivassa järjestyksessä. Aliohjelmia muodostetaan tarpeen mukaan ja keskeinen tavoite on, että yksi aliohjelma keskittyy yhteen loogiseen kokonaisuuteen. Aliohjelman keskeinen toiminnallisuus tulee näkyä sen nimessä. Tästä hyviä esimerkkejä ovat aliohjelmat `input`, `print`, `int`, `float`, `str` ja `len`, joiden toiminnallisuus on tässä vaiheessa meille tuttua ja nimi kertoo sen.

Keskeinen syy aliohjelmien muodostamiseen on niiden uudelleenkäyttö. Käytännössä tavoite on, että yksi aliohjelma tekee yhden asian hyvin, jolloin sitä voi käyttää sekä samassa että muissa ohjelmissa samaan tarkoitukseen ja näin säästää aikaa ja vaivaa. Toinen syy aliohjelmien muodostamiseen on ohjelman kasvaminen niin pitkäksi, että sen ymmärrettävyys kärsii. Tällöin sopivia toiminnallisuuksia voidaan erottaa omiksi kokonaisuuksiksi aliohjelmiin. Aliohjelmien pituus ei ole itseisarvoa vaan usein muutaman rivin koodilohko voidaan tehdä omana aliohjelma, jos se selkiyttää kutsuvaa ohjelmaa. Lisäksi aliohjelmat kasvavat usein ajan myötä, kun niihin lisätään uusia toiminnallisuuksia kuten virheenkäsittely. Käytännössä suuremmat ohjelmat suunnitellaan aliohjelmiin perustuen eli tunnistetaan tarvittavia toiminnallisuuksia ja ohjelman runko muodostuu niistä. Tässä oppaassa tyypillinen ohjelma sisältää tiedon kysymistä käyttäjältä, tietojen analyysiä ja tulosten tulostamista käyttäjälle, jolloin pääohjelman lisäksi tällaisessa ohjelmassa voisi olla kolme edellä mainittuihin asioihin keskittyvää aliohjelmaa.

Pää- ja aliohjelmista muodostuva ohjelma kirjoitetaan tiedostoon Taulukon 5.1 mukaisessa järjestyksessä. Määrittelemällä kiintoarvot ja aliohjelmat tiedoston alussa, ne ovat

käytettävissä kaikissa ohjelman osissa. Jokainen ali-/pääohjelma on itsenäinen ohjelma eli alussa määritellään ohjelmassa tarvittavat muuttujat ja tehdään tarvittavat operaatiot ennen ohjelman loppumista `return`-käskyyn. Huomaa, että `return` siirtää ohjelman suorituksen takaisin kutsuvaan ohjelmaan, joten sen jälkeen olevia sisennettyjä käskyjä ei suoriteta, vaan jokainen ohjelma loppuu yhteen `return`-käskyyn.

Taulukko 5.1. Pää- ja aliohjelmat tiedostossa.

1. Kiintoarvojen määrittelyt
2. Aliohjelmat kutsujärjestyksessä
3. Pääohjelma
4. Pääohjelman kutsu

Nimettyä koodilohkoa voi kutsua Pythonin komentorivitulkissa. Kun olet tehnyt nimetyn koodilohkon ja suorittanut sen kerran IDLEssä, komentorivitulkki tietää, missä ko. nimetty koodilohko on. Näin ollen voit kirjoittaa IDLEn interaktiiviseen ikkunaan tekemäsi aliohjelman nimen ja suorittaa sen käymättä lähdekooditiedostossa. Tämä on kätevä ominaisuus, mikäli olet tottunut käyttämään interaktiivisen ikkunan komentorivitulkkia.

Kuten aiemmin oli puhetta, pääohjelman kutsu on ainoa päätasolla oleva ohjelmakäsky. Kääntäen tämä tarkoittaa sitä, että kaikki toiminnallinen koodi kirjoitetaan pää- ja aliohjelmien sisään eli def ja return -käskyjen väliin. Tällä tavoin ohjelmista muodostuu itsenäisiä kokonaisuuksia, joita ohjelmoijat voivat tehdä itsenäisesti ja tarvittaessa tällainen moduuli on helppo vaihtaa uuteen. Tähän asti katsomamme ohjelmat ovat toimineet näin, mutta käytännössä aliohjelmiin pitää saada tietoa ohjelman lähtökohdaksi ja lopputulokset pitää saada palautettua kutsuvaan ohjelmaan, jotta ohjelma voi toimia yhteistyössä muun ohjelman kanssa. On siis aika siirtyä tiedonvälitykseen ohjelmien välillä eli parametreihin ja paluuarvoihin.

Tiedonvälitys pää- ja aliohjelmien välillä

Edellä kävimme läpi pää- ja aliohjelmarakenteen eli miten ohjelmasta voidaan tunnistaa koodilohkoja ja nimetä sekä kutsua niitä tarpeen mukaan. Aiemmissa esimerkeissä koodilohkot olivat niin itsenäisiä, ettei niissä tarvinnut edes siirtää tietoa ohjelmasta toiseen. Käytännössä tämä on poikkeustilanne ja normaalisti aliohjelmakutsuihin liittyy tiedon siirtoa aliohjelmaan ja takaisin sieltä. Esimerkiksi `input`-aliohjelma tulostaa suluissa olevan tekstin näytölle ja palauttaa käyttäjän antaman merkkijonon. Tietoa välitetään aliohjelmille nimen perässä olevissa suluissa ja näitä tietoalkioita kutsutaan parametreiksi, kun taas tietoa palautetaan aliohjelmasta `return`-käskyllä ja tätä kutsutaan paluuarvoksi.

Esimerkissä 5.4 on useita aliohjelmia, joissa näkyy parametrien ja paluuarvojen normaalit vaihtoehdot. Aiemmissa esimerkeissä tiedonsiirtoa ei ollut ja tämä vaihtoehto näkyy Osassa 1 eli `tulostaOhjeet()` aliohjelman nimen perässä on tyhjät sulut, ts. ei parametreja, ja aliohjelma päättyy `return None` -käskyyn eli paluuarvon tilalla on sana `None`, ts. paluuarvoa ei ole. Osa 2 aliohjelma `tulostaTervehdys(Nimi, Osasto, Vuosikurssi)` saa 3 parametriä, jotka ovat `Nimi`, `Osasto` ja `Vuosikurssi`, eikä tämäkään aliohjelma palauta mitään. Osa 3 aliohjelma `kysyVuosi()` ei saa parametreja, mutta se kysyy käyttäjältä vuosiluvun ja palauttaa sen kokonaislukuna, `return Vuosi`. Osa 4 aliohjelma `parillinenVuosi(Vuosi)` saa parametrina vuosiluvun, josta se selvittää parillisuuden ja palauttaa sen `return Parillinen`. Osa 5 aliohjelma `tulostaTulokset(Vuosi, Parillinen)` saa parametreina vuoden ja parillisuustiedon, jotka se tulostaa käyttäjälle eikä palauta mitään kutsuvaan ohjelmaan. Ohjelman lopuksi Osa 6 sisältää pääohjelman, jossa edellä mainittuja aliohjelmia kutsutaan,

välitetään niihin tietoa, otetaan talteen niiden paluuarvoja ja kiitetään lopuksi käyttäjää ohjelman käytöstä. Osa 7 on ainoa päätasolla oleva ohjelmakäsky eli pääohjelman kutsu.

Esimerkki 5.4. Tiedonvälitys pääohjelman ja erilaisten aliohjelmien välillä

```
# Osa 1. tulosta-aliohjelma ilman tiedonvälitystä
def tulostaOhjeet():
    print("Tämä ohjelma esittelee pää- ja aliohjelmarakenteet.")
    return None

# Osa 2. tulosta-aliohjelma saa tietoa parametreillä
def tulostaTervehdys(Nimi, Osasto, Vuosikurssi):
    print("Terve vaan " + Nimi + "!")
    print("Sanoit olevasi osastolla " + Osasto + ".")
    print("Ja että meneillään on " + Vuosikurssi + ". vuosi.")
    return None

# Osa 3. kysy-aliohjelma palauttaa tietoa return'lla
def kysyVuosi():
    Vuosi = int(input("Anna vuosiluku: "))
    return Vuosi

# Osa 4. Selvitetään vuoden parillisuus ja palautetaan tieto return'lla
def parillinenVuosi(Vuosi):
    Parillinen = False
    if ((Vuosi % 2) == 0):
        Parillinen = True
    return Parillinen

# Osa 5. tulosta tietoja -aliohjelma saa tietoa parametreillä
def tulostaTulokset(Vuosi, Parillinen):
    if (Parillinen == True):
        Tulosta = "parillinen"
    else:
        Tulosta = "pariton"
    print("Vuosi on "+str(Vuosi)+", joka on " + Tulosta + " vuosi.")
    return None

# Osa 6. Pääohjelma ohjaa toimintaa kutsumalla aliohjelmia ja
välittämällä tietoa
def paaohjelma():
    tulostaOhjeet()
    tulostaTervehdys("Brian", "Tite", "4" )
    Vuosiluku = kysyVuosi()
    ParillinenVuosi = parillinenVuosi(Vuosiluku)
    tulostaTulokset(Vuosiluku, ParillinenVuosi)
    print("Kiitos ohjelman käytöstä.")
    return None

# Osa 7. Päätasolla vain paaohjelma()-kutsu
paaohjelma()
# eof
```

Tuloste

```
Tämä ohjelma esittelee pää- ja aliohjelmarakenteet.
Terve vaan Brian!
Sanoit olevasi osastolla Tite.
Ja että meneillään on 4. vuosi.
Anna vuosiluku: 2023
Vuosi on 2023, joka on pariton vuosi.
Kiitos ohjelman käytöstä.
```

Esimerkin 5.4 aliohjelmissa näkyy tyypillinen aliohjelmiin liittyvä roolijako eli osa aliohjelmista keskittyy käyttäjäinteraktioon ja osa laskentaan/analyysiin. Käyttäjäinteraktioon eli käyttöliittymään liittyy usein paljon koodia ja tämä näkyy hyvin myöhemmin tässä oppaassa etenkin sitten, kun katsomme graafisen käyttöliittymän toteutusta. Mutta jo tässä vaiheessa olemme nähneet, että tietojen kysyminen käyttäjältä edellyttää usein erilaisia virheentarkistuksia, vaihtoehtojen läpikäyntiä yms., jolloin koodia tahtoo tulla paljon. Tiivistäen käyttöliittymään liittyvät aliohjelmat ovat iso osa ohjelmointia ja esimerkin 5.4 aliohjelmista tämä näkyy Osissa 1, 2, 3 ja 5. Näissä on tyypillisesti tulosteita, merkkijonojen muodostamista, syötteiden muuttamista sopivaan muotoon ja virheentarkistuksia. Esimerkissä 5.4 on vain yksi laskentaan keskittyvä aliohjelma Osassa 4, jossa aliohjelma saa parametrina tietoa, josta se selvittää asioita ja palauttaa lopuksi tietoa kutsuvalle ohjelmalle. Suuri osa aiemmin käyttämistämme aliohjelmista on vastaavia yksinkertaiseen muutokseen keskittyviä aliohjelmia, kuten `int`, `float`, `str` ja `len`. Nämä aliohjelmat eivät tyypillisesti sisällä tulosteita, vaan ne tekevät jotain syötteelle eli parametrille ja palauttavat arvon.

Tietoa aliohjelmaan parametreilla

Tyypillinen tapa yksinkertaistaa aliohjelmia on välittää niihin tarvittavat tiedot parametreina, jolloin aliohjelmassa voidaan keskittyä tiedon käsittelyyn halutulla tavalla. Esimerkissä 5.4 tietoa välitettiin parametreilla aliohjelmiin Osissa 2, 4 ja 5: `tulostaTervehdys(Nimi, Osasto, Vuosikurssi)`, `parillinenVuosi(Vuosi)` ja `tulostaTulokset(Vuosi, Parillinen)`. Kuten aliohjelmien nimistä näkyy, kaksi niistä keskittyi tulostamiseen ja niiden osalta keskeiset operaatiot aliohjelmassa ovat merkkijonojen muodostaminen sekä tulostaminen näytölle. Osan 4 parillisuuden selvittävä aliohjelma taas palautta Tosi/Epätosi -tiedon kutsuvaan ohjelmaan, jotta sen avulla voidaan tulostaa sopiva viesti käyttäjälle.

Aliohjelmakutsussa sille välitettävät tiedot laitetaan nimen perässä oleviin sulkuihin, jonka jälkeen tiedot siirtyvät aliohjelman määrittelyrivillä suluissa oleviin parametreihin samassa järjestyksessä. Tämän jälkeen aliohjelmassa voidaan käyttää parametreja normaalin muuttujan tavoin. Parametrit toimivat samalla tavoin kuin normaalit aliohjelman sisäiset muuttujat, mutta koska ne on määritelty aliohjelman nimen yhteydessä, saavat ne arvonsa aliohjelmakutsusta ja siksi niitä kutsutaan parametreiksi.

Aliohjelmassa voi olla useita parametreja pilkuilla erotettuina. Tämä on meille tuttua `print`-lauseesta, jolla voi tulostaa monta arvona näytölle tyyliin `print("Moi", "Ville", "5v")`. Jotta aliohjelmakutsussa ja aliohjelman määrittelyrivillä olevat tiedot voi erottaa yksikäsitteisesti, kutsutaan aliohjelmakutsussa olevia tietoja argumenteiksi ja määrittelyrivillä olevia tietoja parametreiksi. Argumentteja ja parametreja tulee olla yhtä monta ja argumentit sijoitetaan parametreille arvoiksi niiden järjestyksen perusteella. Jatkossa argumentti-termiä käytetään vain tarvittaessa ja tyypillisesti tässä oppaassa sekä argumentteihin että parametreihin viitataan termillä parametri.

Parametrien käytön periaatteet on esitetty edellä tiivistetysti ja palataan parametrien keskeisiin ominaisuuksiin kohta esimerkkien kanssa. Mutta ennen sitä tutustutaan paluuarvojen periaatteisiin.

Tietoa aliohjelmasta paluuarvolla ja funktio

Aliohjelma päättyy `return`-käskyyn, jonka perässä voi olla paluuarvo tai sen puuttuessa `None`. Kuten esimerkissä 5.4 näkyy, kaikki aliohjelmat eivät käytä paluuarvoa. Python tulkki lisää tarvittaessa `None`:n tai `return None` -käskyn aliohjelman loppuun, mutta hyvin ohjelmointityylin nimissä ohjelmoijan on syytä laittaa ne näkyville koodiin. Tällöin aliohjelman loppukohdasta tai sen paluuarvosta ei tule epäselvyyttä ja vältetään turhia

virheitä. `return`-käsky siirtää ohjelman kulun takaisin kutsuvaan ohjelmaan, jolloin aliohjelmassa oleva `return` on aliohjelman viimeinen suoritettava käsky eikä sen jälkeen saa olla muita käskyjä.

Aliohjelman päättyessä se palauttaa paluuarvon kutsuvaan ohjelmaan aliohjelmakutsun paikalle. Tämä mahdollistaa paluuarvon sijoittamisen muuttujaan esimerkin 5.4 Osa 6 pääohjelman mukaisesti. Jos arvoa halutaan käyttää vain yhden kerran, voidaan se ottaa huomioon laskukaavassa operandina, antaa parametrinä esim. `print`-käskylle ja tulostaa näytölle tms. Turvallisinta on sijoittaa paluuarvo muuttujan arvoksi.

Paluuarvoon liittyen on syytä muistaa muutama asia. Hyvän ohjelmointityylin nimissä ohjelmassa tulee olla vain yksi `return`-käsky, eikä esim. jokaisessa valintarakenteen haarassa oma `return`. Käytännössä yhteen `return`-käskyyn päästään suunnittelemalla ohjelma hyvin ja käyttämällä tarvittaessa muuttujaa, johon paluuarvo selvitetään ohjelman kuluessa ja palauttamalla se viimeisenä olevalla `return`-käskyllä. Toiseksi Python sallii useiden paluuarvojen käytön, mutta selkeyden nimissä tässä oppaassa käytetään vain yhtä paluuarvoa. Tutustumme myöhemmin rakenteisiin tietorakenteisiin, jotka voivat sisältää useita tietoalkioita ja sopivat useiden tietoalkioien palauttamiseen aliohjelmasta.

Lopuksi kannattaa huomata käsite funktio, joka tarkoittaa arvon palauttavaa aliohjelmaa. Hyvä esimerkki funktiosta on `len`-aliohjelma, joka saa parametrinä esim. merkkijonon ja palauttaa sen pituuden. Funktiot eli paluuarvon palauttavat aliohjelmat ovat niin yleisiä, että tässä oppaassa funktio- ja aliohjelma-käsitteitä käytetään usein ristiin. Aliohjelmaa, joka ei palauta paluuarvoa, voi kutsua proseduuriksi, mutta tätä termiä käytetään nykyään harvoin etenkin Python-kielen kanssa. Kuten edellä todettiin, tulkki lisää kaikkiin aliohjelmiin lopuksi `return None` -käskyn, jolloin ne ovat funktioita.

Parametrien keskeisiä ominaisuuksia

Tutustutaan seuraavaksi parametrien keskeisiin ominaisuuksiin. Nämä on dokumentoitu tänne, sillä näihin törmää ajoittain ja tästä voi katsoa, mistä näissä on kysymys. Mutta aliohjelmien kanssa tärkeintä on ruveta tekemään niitä perusasioiden pohjalta ja tutustua erikoistapauksiin tarpeen mukaan. Tässä käymme läpi arvoparametrit, parametrien järjestyksen, nimetyt parametrit sekä parametrien oletusarvot.

Aliohjelman parametrit ovat arvoparametreja eli ne ovat kopioita kutsuvassa ohjelmassa olevista argumenteista (parametreistä). Totesimme aiemmin, että parametreja voi käyttää aliohjelmassa muuttujan tavoin. Tämä mahdollistaa parametrien arvon muuttamisen, jolloin herää kysymys, muuttuuko kutsuvassa ohjelmassa olevien argumenttien arvot samalla kertaa. Lyhyt vastaus tähän on ei, sillä aliohjelman parametrit ovat alkuperäisten argumenttien kopioita. Syy tähän on se, että argumenttina voi olla esim. merkkijono ”Ville” tai luku 5, eikä näitä vakioita voi muuttaa kutsuvassa ohjelmassa ja siksi aliohjelmissa on parametreina omat muuttujat aliohjelman sisäistä käyttöä varten. Käytännössä tämä tarkoittaa sitä, että muuttunut tieto pitää palauttaa aliohjelmasta kutsuvaan ohjelmaan paluuarvona. Esimerkissä 5.5 pääohjelmasta kutsutaan `tulosta`-aliohjelmaa ja parametreja muutetaan aliohjelmassa. Ja kun tullaan takaisin pääohjelmaan, pääohjelman muuttujien (argumenttien) arvot ovat pysyneet muuttumattomina.

Sekaannusten välttämiseksi muuttujille kannattaa antaa eri nimet kutsuvassa ohjelmassa ja aliohjelmassa. Kun muuttujilla on eri nimet, riski niiden ajatteluun samana muuttujana on pienempi ja tulee vähemmän virheitä. Palaamme arvoparametreihin myöhemmin oppaassa rakenteisten tietorakenteiden yhteydessä.

Esimerkki 5.5. Arvoparametri on kopio, eikä sen arvon muutos näy kutsuvassa ohjelmassa

```
def tulosta(Etunimi, Ika):
    print("Aliohjelman saamat parametrit:", Etunimi, Ika)
    Etunimi = "Kalle"
    Ika = Ika + 1
    print("Aliohjelman muutetut parametrit:", Etunimi, Ika)
    return None

def paaohjelma():
    Nimi = "Ville"
    Vuosia = 9
    print("Pääohjelman muuttujat ennen aliohjelmaa:", Nimi, Vuosia)
    tulosta(Nimi, Vuosia)
    print("Pääohjelman muuttujat aliohjelma jälkeen:", Nimi, Vuosia)
    return None

paaohjelma()
```

Tuloste

```
Pääohjelman muuttujat ennen aliohjelmaa: Ville 9
Aliohjelman saamat parametrit: Ville 9
Aliohjelman muutetut parametrit: Kalle 10
Pääohjelman muuttujat aliohjelma jälkeen: Ville 9
```

Aliohjelman parametrit saavat arvonsa oletusarvoisesti niiden järjestyksen perusteella. Esimerkissä 5.6 näkyy, miten pääohjelmasta kutsutaan tulosta-aliohjelmaa erilaisilla argumenteilla ja miten ne sijoitetaan aliohjelman parametreiksi järjestyksen perusteella. Vaikka argumenteilla ja parametreilla olisi samat nimet, sijoitetaan arvot parametreille järjestyksen perusteella. Siksi selkeintä on antaa argumenteille ja parametreille eli nimet, jolloin arvojen sijoitus menee luontevasti järjestyksen mukaan virheitä välttäen. Esimerkin 5.6 muuttujien kierrätys ei onnistu vahvasti tyypitetyissä ohjelmointikielissä kuten Java ja C, sillä niissä parametrien tietotyypit on nimetty ja kääntäjän varmistaa, että tietotyypit ovat oikein. Pythonin dynaaminen tietotyyppien käsittely mahdollistaa merkkijonon, kokonaisluvun ja desimaaliluvun vapaan vaihtamisen esimerkin 5.6 mukaisesti.

Esimerkki 5.6. Parametrit saavat arvonsa paikan perusteella suluissa

```
def tulosta(Eka, Toka, Kolmas):
    print(Eka, Toka, Kolmas)
    return None

def paaohjelma():
    Muuttuja1 = "Eka"
    Muuttuja2 = 1
    Muuttuja3 = 2.34
    tulosta(Muuttuja1, Muuttuja2, Muuttuja3)
    tulosta(Muuttuja2, Muuttuja3, Muuttuja1)
    tulosta(Muuttuja3, Muuttuja1, Muuttuja3)
    return None

paaohjelma()
```

Tuloste

```
Eka 1 2.34
1 2.34 Eka
2.34 Eka 2.34
```

Parametrien oletusjärjestystä voi muuttaa nimetyillä parametreilla esimerkin 5.7 mukaisesti. Pääohjelman `tulosta`-aliohjelmakutsussa `Ika`-parametrille annetaan muuttujan `Vuosia` arvo, ja `Etunimi`-parametrille annettaan muuttujan `Nimi` arvo. Argumenttien järjestyksellä ei ole väliä, koska parametrien nimien käyttö tekee sijoituksista yksikäsitteisiä. Huomaa, että käytämme nimettyjä parametreja `print`-käskyn yhteydessä, kun haluamme antaa `sep` tai `end` -parametreille uudet arvot. Yleisesti ottaen on selkeämpää rakentaa ohjelmien tiedonvälitys parametrien järjestykseen perustuen. `print` ja `sep`/`end` on kuitenkin hyvä esimerkki normaalista poikkeavasta, mutta toimivasta ratkaisusta.

Esimerkki 5.7. Parametrien nimeäminen

```
def tulosta(Etunimi, Ika):
    print(Etunimi, Ika)
    return None

def paaohjelma():
    Nimi = "Ville"
    Vuosia = 9
    tulosta(Ika=Vuosia, Etunimi=Nimi)
    return None

paaohjelma()
```

Tuloste

```
Ville 9
```

Viimeisenä asiana parametreihin liittyen katsomme niiden oletusarvot esimerkin 5.8 mukaisesti. Aliohjelman parametreille voi antaa oletusarvot määrittelyrivillä tyyliin `def tulosta(Etunimi="Sanna", Ika=50)`. Käytännössä tämä tarkoittaa sitä, että aliohjelmaa `tulosta` voi kutsua ilman parametreja, jolloin `Etunimi`-parametrilla saa arvon `"Sanna"` ja `Ika`-parametri arvon `50`. Esimerkin 5.8 mukaisesti ohjelma toimii oikein, kun kaikki parametrit korvataan oletusarvoilla, tai oletusarvoja käytetään järjestyksessä viimeisten parametrien kohdalla. Argumentit sijoitetaan parametreiksi järjestyksen mukaan (Esimerkki 5.6), jolloin ensimmäinen argumentti sijoitetaan aina ensimmäiseen parametriin ja esimerkin 5.8 `tulosta(Vuosia)` -aliohjelmakutsu ei ole mielekäs. Parametrien oletusarvot toimivat hyvin `print`-aliohjelman `end` ja `sep` parametrien kanssa, joiden oletusarvot ovat `end="\n"` ja `sep=" "`.

Esimerkki 5.8. Parametrien oletusarvot

```
def tulosta(Etunimi="Sanna", Ika=50):
    print(Etunimi, Ika)
    return None

def paaohjelma():
    Nimi = "Ville"
    Vuosia = 9
    tulosta()
    tulosta(Nimi)
    tulosta(Nimi, Vuosia)
    tulosta(Vuosia) # Lopputulos ei ole toivottu!!
    return None

paaohjelma()
```

Tuloste

```
Sanna 50
Ville 50
Ville 9
9 50
```

Tiedonvälityksessä eli parametrien ja paluuarvojen kanssa oleellista on yksikäsitteisyys eli tarkoituksen on oltava selvä. Toimivan ohjelmointikielen toteutus sisältää aina monia teknisiä yksityiskohtia, jotka voivat tuntua oudoilta esimerkin 5.8 mukaisesti. Mutta kun ohjelmoi enemmän ja näkee erilaisia tietorakenteita ja algoritmeja, saattavat oudoltakin tuntuvat ratkaisut tuntua loogisemmilta. Esimerkiksi parametrien ominaisuuksiin kannattaa palata tarpeen tullen, eikä niiden ulkoa opettelusta ole tyypillisesti hyötyä ennen kuin niille on luontevat käyttökohteen `print`-käskyn tyyliin.

Nimiavaruus ja tunnukset

Yksi keskeinen aliohjelmien tavoite on jakaa ohjelma pienempiin helpommin hallittaviin itsenäisiin kokonaisuuksiin. Kuten tämän luvun alussa oli puhetta, iso talo jaetaan eri huoneisiin, joissa kussakin keskitytään lähtökohtaisesti yhteen toiminnallisuuteen kuten ruuanlaittoon, peseytymiseen, nukkumiseen, oleskeluun tms. Vastaavasti aliohjelmat ovat omia kokonaisuuksia, joissa pyritään tekemään aina yksi asia ja keskittymään siihen. Aliohjelmaan tullaan aliohjelmakutsulla parametrien kanssa, ja kun aliohjelma on tehnyt tehtävänsä, palautetaan tulos paluuarvona kutsuvalle ohjelmalle tai esim. tulostetaan se näytölle. Mutta `def` ja `return` -käskyjen välissä aliohjelma on oma kokonaisuus, jossa voidaan keskittyä yhteen asiaan ja sen vuoksi jokaisella pää- ja aliohjelmalla on myös oma nimiavaruus.

Käytännössä Pythonin nimiavaruus-konsepti tarkoittaa sitä, että yhdessä nimiavaruudessa olevat nimet/tunnukset eivät näy muualla eivät muualla määritellyt tunnukset näy sinne. Jokaisella pää- ja aliohjelmalla on siis oma nimiavaruus ja vaikka toisessa nimiavaruudessa olisi samoja tunnuksia, ne eivät näy muualle. Tämä konsepti mahdollistaa pää- ja aliohjelmien kehittämisen toisistaan riippumatta, kunhan rajapinnat eli parametrit, paluuarvot ja aliohjelmanimet on sovittu yhdessä.

Esimerkissä 5.9 on pääohjelman lisäksi 2 aliohjelmaa eli siinä on 3 erillistä nimiavaruutta. Kaikissa pää- ja aliohjelmissa voi olla samannimisiä muuttujia, mutta ne eivät näy oman nimiavaruuden ulkopuolelle, vaan ne ovat toisistaan täysin riippumattomia muuttujia. Ohjelman tulosteet demonstroivat tämän eli kaikissa ohjelmissa tulostetaan `Muuttuja`-nimisen muuttujan arvo, mutta ne ovat toisistaan riippumattomia.

Esimerkki 5.9. Nimiavaruus

```
def aliohjelma1():
    Muuttuja = 1  #  nimiavaruus 2
    print("aliohjelma1:", Muuttuja)
    return None

def aliohjelma2():
    Muuttuja = 2.34  # nimiavaruus 3
    print("aliohjelma2:", Muuttuja)
    return None

def paaohjelma():
    Muuttuja = "Hei" # nimiavaruus 1
    print("Pääohjelma alussa:", Muuttuja)
    aliohjelma1()
    aliohjelma2()
    print("Pääohjelma lopussa:", Muuttuja)
    return None

paaohjelma()
```

Tuloste

```
Pääohjelma alussa: Hei
aliohjelma1: 1
aliohjelma2: 2.34
Pääohjelma lopussa: Hei
```

Tunnus eli muuttuja, aliohjelma tai kiintoarvo

Luvun 1 mukaan muuttujien nimet ovat tunnuksia ja tunnusten nimeämiselle oli sääntöjä. Myös aliohjelmien nimet ovat tunnuksia ja niiden nimeämistä koskevat samat säännöt kuin muuttujia. Monissa muissa ohjelmointikielissä on muuttujien lisäksi olemassa vakio-tunnuksia, joiden arvoa ei voi muuttaa ja ohjelmointiympäristö varmistaa, ettei niitä muuteta. Pythonissa tällaisia vakioita ei ole. Vakioilla voidaan kuitenkin lisätä ohjelman ymmärrettävyyttä ja ylläpidettävyyttä merkittävästi, sillä niiden avulla voidaan esimerkiksi määritellä merkkijonon pituudelle maksimipituus `MAX_PITUUS = 80` tai minimi ikäraja `IKA = 18`, jotka ovat käytössä kaikkialla ohjelmassa muuttumattomina. Näin ollen Pythonissa tämä asia pitää hoitaa luvussa 1 mainitun muuttujan roolin *kiintoarvo* avulla. Käytännössä tämä tarkoittaa sitä, että ohjelmoijan on oltava tarkkana ja katsottava, ettei kiintoarvon arvo muutu ohjelman suorituksen aikana. Koska vakio-konsepti on keskeinen ohjelmointiin liittyvä peruskonsepti, käytetään tässä oppaassa kiintoarvoja, joka tarkoittaa sitä, että (1) lähtökohtaisesti kiintoarvot näkyvät kaikkialla ohjelmassa, (2) Pythonissa ohjelmoija vastaa siitä, ettei kiintoarvon arvo muutu ohjelman aikana ja (3) muissa ohjelmointikielissä käytettäisiin tyypillisesti vakio-tunnuksia.

Tunnusten näkyvyys

Tässä oppaassa lähtökohtana on, että muuttujat ovat aina paikallisia omassa nimiavaruudessaan yhden aliohjelman sisällä. Toisaalta aliohjelmat tulee aina tehdä päätasolle siten, että ne näkyvät ja ovat käytettävissä kaikkialla tehdyssä ohjelmassa. Kiintoarvot, eli vakiot, voivat olla paikallisia nimiavaruuteen liittyen, mutta lähtökohtaisesti niistä on eniten hyötyä silloin kun ne näkyvät kaikissa pää- ja aliohjelmissa ja tieto on käytettävissä kaikkialla yhdessä ohjelmassa eli kun ne on määritelty ohjelman päätasolla globaaleina kiintoarvoina.

Edellä olevien periaatteiden seurauksena tässä oppaassa **yhteiset eli globaalit muuttujat ovat kiellettyjä**. Alla oleva Taulukko 5.2 näyttää tässä oppaassa tähän mennessä esitellyt tunnukset sekä niille sopiva näkyvyys ohjelmassa. Käytännössä globaaleja muuttujia

joudutaan joskus käyttämään teknisen toteutuksen takia, joten käytämme niitä muutamassa poikkeustapauksessa oppaan lopussa ja asia mainitaan silloin aina erikseen.

Taulukko 5.2. Tunnusten näkyvyys

Tunnus	Näkyvyys
Muuttuja	Paikallinen
Kiintoarvo (Vakio)	Paikallinen tai globaali
Aliohjelma	Globaali

Esimerkissä 5.10 näkyy globaalin kiintoarvon määrittely päätasolla ohjelman alussa sekä käyttö pää- ja aliohjelmissa. Tässä esimerkissä kiintoarvo määrittää merkkien lukumäärän ja sitä käytetään toistorakenteessa lopetusehtona. Tämä on tyypillinen esimerkki globaalin kiintoarvon käytöstä.

Esimerkki 5.10. Globaalin kiintoarvon käyttö

```
MERKKEJA = 5

def kysyMerkkeja():
    i = 0
    Merkit = ""
    print("Anna "+str(MERKKEJA)+" merkkiä yksi kerrallaan.")
    while (i < MERKKEJA):
        Merkki = input("Anna "+str(i+1)+". merkki: ")
        Merkit = Merkit + Merkki
        i = i + 1
    return Merkit

def paaohjelma():
    Merkkijono = kysyMerkkeja()
    for i in range(MERKKEJA):
        print(Merkkijono[i], end=" ")
    print()
    return None

paaohjelma()
```

Tuloste

```
Anna 5 merkkiä yksi kerrallaan.
Anna 1. merkki: E
Anna 2. merkki: e
Anna 3. merkki: m
Anna 4. merkki: i
Anna 5. merkki: l
E e m i l
```

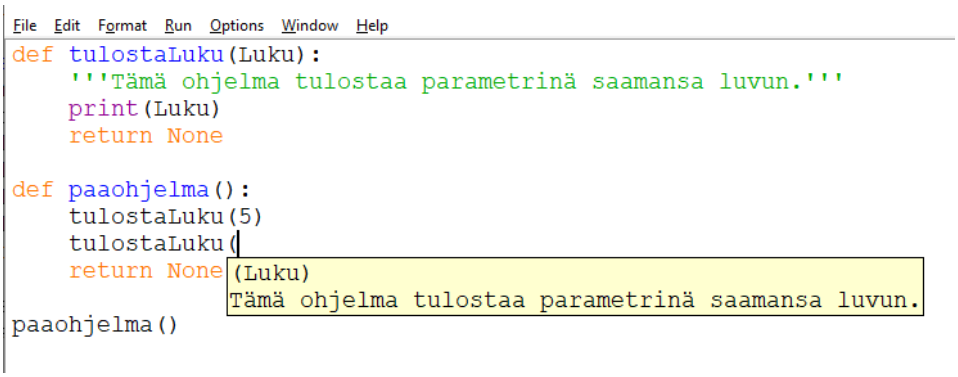
Aliohjelmiin liittyviä muita huomioita

Aliohjelmiin liittyy paljon asioita, mutta edellä oleville periaatteilla pääsee hyvään vauhtiin. Tässä on lisäksi muutama muu huomionarvoinen asia, joihin kannattaa tutustua tarkemmin, kun tarve ilmenee tai mielenkiintoa riittää. Näitä asioita ovat aliohjelmien DocString eli dokumentaatiorivi, tyypillisiä aliohjelmien käyttötapoja ja valikkopohjainen ohjelma.

Olet varmaan huomannut, että kun kirjoitat IDLEssä aliohjelmia niin aukeavan sulun kirjoittaminen tuo aliohjelmaan liittyvän vinkin näkyville kuvan 5.1 mukaisesti. Käytännössä tämä Tool-tip antaa vinkkejä ohjelman käyttöön eli ensimmäisellä rivillä näkyy

suluissa aliohjelman parametrit suluissa ja toisella rivillä näkyy merkkijono, joka kertoo ohjelman käyttötarkoituksen. Ohjelman käyttötarkoitus on Pythonin Doc String ja se on aliohjelman ensimmäisellä määrittelyn jälkeisellä rivillä oleva merkkijono, joka on merkitty kolmen heittomerkkiparin sisään eli `'''DocString'''`. Esimerkin 5.1 ohjelma ei ole muutoin kiinnostava kuin, että siitä näkyy tämä DocString/Toop-tip -käyttö. Ohjelma on suoritettava kerran, jotta tulkki lukee koodin ja kirjaa itselleen siellä olevat aliohjelmat sekä niiden tiedot.

Kuva 5.1. IDLE, DocString ja Tool-tip



Esimerkki 5.11. DocString eli aliohjelman dokumentaatiorivi, Tool-tip

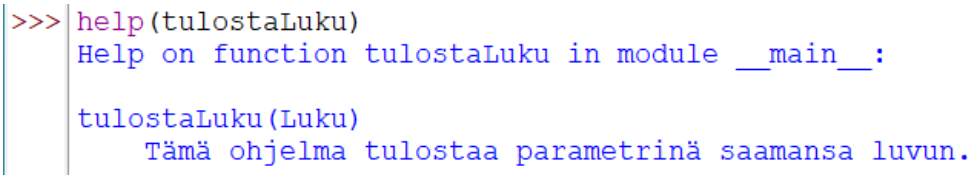
```
def tulostaLuku(Luku):
    '''Tämä ohjelma tulostaa parametrinä saamansa luvun.'''
    print(Luku)
    return None

def paaohjelma():
    tulostaLuku(5)
    return None

paaohjelma()
```

Tuloste

```
5
```

Tool-tipin käyttöä kannattaa harjoitella tehtäviä tehdessä, sillä tyypillinen Python-asennus sisältää dokumentaation ja siten nämä vinkit ovat käytössä vaikkei muuta dokumentaatiota olisikaan saatavilla. IDLEn interaktiivisessa ikkunassa voi antaa myös käskyn `help(tulostaLuku)` kuvan 5.2 mukaisesti. Tulkki esittää kootusti annetun aliohjelman dokumentaatiorivit, joissa on ohjeena, mitä milläkin funktiolla voi tehdä. Jos taas kirjoitat pelkän `help()` tulkkiin, käynnistyy Pythonin sisäänrakennettu apuohjelma. Sieltä voit tulkin kautta selata ohjetietoja eri käskyistä ja moduuleista kuten esim. `print` ja `input`. Helppi lopetetaan jättämällä rivi tyhjäksi, kirjoittamalla `quit` ja painamalla enter.

Kuva 5.2. IDLEn interaktiivinen ikkuna ja help-funktio



Esimerkissä 5.12 näkyy kaksi tyypillistä aliohjelmaa, joita voi käyttää monissa eri tarkoituksissa. Vaikka nämä ohjelmat eivät suoraan sopisikaan tarpeeseesi, kannattaa katsoa niiden rakenne. Osassa 1 näkyy `maksimi`-aliohjelma, joka palauttaa kahdesta luvusta suuremman paluuarvona. Käytännössä tämä rakenne on yleiskäyttöinen eli aliohjelman nimi kertoo, mitä ohjelma tekee, parametrit kertovat annettavat syötteet ja paluuarvo toivotun tuloksen. Esimerkiksi `len`-funktio toimii juuri tällä idealla eli merkkijonosta lasketaan siinä olevien merkkien määrä, joka palautetaan paluuarvona.

Vastaavasti Osan 2 `teeKyltti`-aliohjelma saa tekstiä parametrina ja aliohjelma kehystää tekstin sopivalla raamilla. Aliohjelma selvittää ensin merkkijonon pituuden, lisää siihen sopivat alku- ja merkit sekä palauttaa valmiin lopputuloksen. Ohjelman yleiskäyttöisyys tulee siitä, että se tekee vastaavia kehyksiä riippumatta annetun merkkijonon pituudesta.

Huomaa, että `teeKyltti`-aliohjelmaa käytetään esimerkissä kolmella eri tavalla. Ensimmäinen vaihtoehto on kysyä `input`:lla merkkijono, sijoittaa paluuarvo muuttujaan, kutsua `teeKyltti`-aliohjelmaa ja sijoittaa paluuarvo muuttujaan, joka tulostetaan lopuksi `print`:llä. Toisessa vaihtoehdossa `input`:n paluuarvo menee suoraan `teeKyltti`-aliohjelman parametriksi, ja sen paluuarvo sijoitetaan muuttujaan sekä tulostetaan `print`:llä. Kolmannessa vaihtoehdossa `input`:n paluuarvo annetaan `teeKyltti`-aliohjelmalle parametrina ja sen paluuarvo annetaan `print`:lle parametrina. Nämä toteutusvaihtoehdot perustuvat parametrien ja paluuarvojen ymmärrykseen ja niitä voi käyttää sopivissa paikoissa. Paluuarvojen tehokas hyödyntäminen lyhentää koodia ja voi olla toimiva ratkaisu, mutta tässä ratkaisussa mitään tietoa ei ole muuttujissa uudelleenkäyttöä varten ja ymmärrettävyyden kannalta tällainen koodi voi olla riski. Siksi tällaista koodia kannattaa kirjoittaa vain, jos pystyy perustelemaan sen hyvin eikä riskiä väärinymmärryksiin tai virheisiin ole.

Esimerkki 5.12. Tyypillisiä aliohjelmien käyttötapoja

```
# Osa 1. Funktio suuremman luvun selvittämiseen
def maksimi(Luku1, Luku2):
    if (Luku1 > Luku2):
        Isompi = Luku1
    else:
        Isompi = Luku2
    return Isompi

# Osa 2. Funktio tekemään tulostukseen haluttavia asioita
def teeKyltti(Teksti):
    Plakaatti = "*" * (len(Teksti) + 4) + "\n"
    Plakaatti = Plakaatti + "* " + Teksti + " *\n"
    Plakaatti = Plakaatti + "*" * (len(Teksti) + 4) + "\n"
    return Plakaatti

# Osa 3. Pääohjelma
def paaohjelma():
    # Kutsutaan maksimi-funktiota, Osa 1
    Luku1 = int(input("Anna luku 1: "))
    Luku2 = int(input("Anna luku 2: "))
    Suurempi = maksimi(Luku1, Luku2)
    print("Suurempi luku on", Suurempi)
    # Kutsutaan teeKyltti-funktiota, Osa 2
    Syote = input("Anna syöte 1: ")
    Taulu = teeKyltti(Syote)
    print(Taulu)
    Taulu = teeKyltti(input("Anna syöte 2: "))
    print(Taulu)
    print(teeKyltti(input("Anna syöte 3: ")))
    return None
paaohjelma()
```

Tuloste

```
Anna luku 1: 10
Anna luku 2: 1111
Suurempi luku on 1111
Anna syöte 1: Mia
*******
* Mia *
*******

Anna syöte 2: Karri
*********
* Karri *
*********

Anna syöte 3: Sanna-Katariina
******************
* Sanna-Katariina *
******************
```

Luvun 4 kokoava esimerkki 4.8 esitteli valikkopohjaisen ohjelman rungon ja esimerkissä 5.13 on valikkopohjaisen ohjelman aliohjelmista koostuva versio. Ohjelma muodostuu nyt pääohjelmasta, jossa on toisto- ja valintarakenteet ohjelman pyörittämiseen, sekä erillisestä `valikko`-aliohjelmasta, jossa on valikon tulostus ja valinnan kysyminen. `valikko`-aliohjelma palauttaa valinnan kokonaislukuna, jolloin pääohjelman valintarakenteessa on helppo siirtyä haluttuun toimintoon. Tämä valikkopohjainen ohjelma tulee toistumaan jatkossa useita kertoja, joten tämän perusrakenne kannattaa opetella näitä tehdessä.

Esimerkki 5.13. Valikkopohjainen ohjelma

```
def valikko(): # Valikko-aliohjelma
    print("Käytettävissä olevat toiminnot:")
    print("1) Tulosta 'Moi'")
    print("2) Tulosta 'Hoi'")
    print("0) Lopeta")
    valinta = int(input("Valintasi: "))
    return valinta

def paaohjelma(): # Pääohjelma
    Valinta = 1
    while (Valinta != 0):
        Valinta = valikko()
        if (Valinta == 1):
            print("Moi")
        elif (Valinta == 2):
            print("Hoi")
        elif (Valinta == 0):
            print("Lopetetaan.")
        else:
            print("Tuntematon valinta, yritä uudestaan.")
        print()
    print("Kiitos ohjelman käytöstä.")
    return None

paaohjelma()
# eof
```

Tuloste

```
Käytettävissä olevat toiminnot:
1) Tulosta 'Moi'
2) Tulosta 'Hoi'
0) Lopeta
Valintasi: 1
Moi

Käytettävissä olevat toiminnot:
1) Tulosta 'Moi'
2) Tulosta 'Hoi'
0) Lopeta
Valintasi: 0
Lopetetaan.

Kiitos ohjelman käytöstä.
```

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.

- Pääohjelma (E5.1, 5.2)
- Aliohjelma (E5.3)
- Pää- ja aliohjelmat sekä ohjelmarakenne (Taulukko 5.1, E5.10)
- Tiedonvälitys pää- ja aliohjelmien välillä (E5.4)
    - Parametrit, arvoparametri, järjestys, nimeäminen, oletusarvot (E5.4, 5.5, 5.6, 5.7, 5.8)
    - Paluuarvo, funktio (E5.4, 5.12)
- Nimiavaruus (E5.9)
- Tunnukset ja näkyvyys (Taulukko 5.2, E5.10)
- Tool-tip, DocString, tyypillisiä aliohjelmien käyttötapoja (E5.11, 5.12)
- Valikkopohjainen ohjelma (E5.13, 5.14)

Näitä asioita käsitellään pääasiassa tässä luvussa ja palaamme muutamiin asioihin uudestaan myöhemmin, kun tilanne muuttuu uusien tietorakenteiden ja käyttötapojen myötä. Näitä kaikkia asioita ei kannata yrittää opetella ulkoa, vaan ensimmäinen vaihe on opetella tunnistamaan nämä konseptit ja palata niihin aina tarpeen mukaan, kun tekee näitä konsepteja sisältäviä ohjelmia. Tekniset yksityiskohdat voi tarkistaa täältä aina tarpeen mukaan. Kun konsepteja on käyttänyt muutaman kerran, rupeaa ne oppimaan tekemisen kautta.

Pienen Python-perusohjelman tyyliohjeet

Tähän lukuun liittyvät tyyliohjeet on koottu alle. Ohjeita on paljon, koska aliohjelmia voi tehdä monella tavalla ja sen takia niiden kanssa tulee paljon erilaisia virheitä. Alla olevia ohjeita noudattamalla vältät useimmat aliohjelmiin liittyvät virheet.

Pää/aliohjelmien yleisiä ohjeita

1. Aliohjelmien nimet ovat tunnuksia, joten niitä koskevat aiemmat tyyliohjeet, esim.
    a. Tunnusten tulee olla kuvaavia, yksikäsitteisiä, selkeitä ja johdonmukaisia
    b. Älä käytä skandinaavisia merkkejä ja mitäänsanomattomien ja harhaanjohtavien tunnusten käyttö on kielletty
2. Aliohjelman nimen tulee kertoa, mitä aliohjelmassa tapahtuu. Käytä tarvittaessa useita sanoja, esim. `kysyTiedot`, `analysoiTiedot`, `tulostaTiedot`
3. Yhdestä sanasta muodostuvat nimet tulee kirjoittaa kaikki kirjaimet pienellä ja useista sanoista muodostuvien nimien ensimmäiset kirjaimet kirjoitetaan suuraakkosilla ensimmäistä kirjainta lukuun ottamatta, esim. `paaohjelma`, `valikko`, `kysyNimi`, `analysoiTiedot`
4. Samassa nimiavaruudessa olevat tunnukset tulee nimetä eri nimillä, esim. kiintoarvoilla, muuttujilla ja aliohjelmilla on oltava eri nimet
5. Aliohjelmien määrittelyssä kaarisulkuihin (ja) tulee vastaanotettavat parametrit. Jos parametrejä ei ole, sulut jätetään tyhjiksi
6. Ohjelmassa ei tule olla koodirivejä, jotka eivät tee mitään tai joita ei voi koskaan saavuttaa, esim.
    a. Aliohjelmat, jotka on määritelty, mutta joita ei kutsuta
    b. `return` käskyn jälkeen samassa koodilohkossa olevat käskyt

Milloin tehdään uusi aliohjelma

7. Jos sama/vastaava toiminta tehdään ohjelmassa monta kertaa tai ohjelmaan lisätään uusi toiminnallisuus kuten tiedoston luku tai kirjoitus
8. Jos ohjelman toiminnallisuuden lisäys kasvattaa ohjelmaa ja tekee siitä hankalasti ymmärrettävän
9. Uutta aliohjelmaa ei tule tehdä, jos se ei tuo lisäarvoa

Aliohjelmien tiedonvälitys

10. Tiedot aliohjelmiin välitetään parametrien avulla
11. Kaikki aliohjelmat päättyvät `return`-käskyyn ja paluuarvoon. Mikäli ohjelma ei palauta tietoa, tulee käyttää `return None` -käskyä
12. Tässä oppaasta aliohjelmasta palautetaan vain yksi paluuarvo, useiden tietoalkioiden palautukseen palataan myöhemmin

Tunnusten näkyvyyden lähtökohdat

13. Globaalit muuttujat ovat kiellettyjä
14. Muuttujat määritellään lokaaleina pää/aliohjelmissa
15. Kiintoarvot ja aliohjelmat määritellään globaaleina
16. Globaalien kiintoarvojen käyttö on suositeltavaa

Pää/aliohjelmien tyypillinen rakenne

17. Pää/aliohjelma alkaa `def`-tunnisteella, jota seuraa ohjelman nimi ja sulut parametreilla sekä kaksoispiste
18. Muuttujien määrittelyt ja alustukset
19. Tietojen kysyminen käyttäjältä
20. Toiminnallinen osuus, esim. laskenta, lukeminen tai kirjoittaminen
21. Tulosten tulostaminen käyttäjälle
22. Lopetusrutiinit ja käyttäjän informointi pää/aliohjelman loppumisesta
23. Kaikki pää/aliohjelmat päättyvät `return`-käskyyn ja paluuarvoon/`None`:en

Pää/aliohjelmista muodostuvan ohjelmatiedoston rakenne

24. Kiintoarvojen määrittelyt
25. Aliohjelmat kutsujärjestyksessä
26. Pääohjelma
27. Pääohjelman kutsu

Valikkopohjaisen ohjelman pääohjelman rakenne

28. Ohjelma perustuu toisto- ja valintarakenteisiin
29. Toisto toteutetaan `while`-rakenteella, josta poistutaan toistoehdon ollessa epätosi
30. Ohjelman valikon tulostus ja valinnan kysyminen tehdään valikko-aliohjelmassa
31. Valintarakenteena käytetään monihaaraista `if-elif-else`-rakennetta
    a. Valintarakenteen haarat vastaavat valikon valintoja samassa järjestyksessä 1-N, jonka jälkeen on ohjelman normaali lopetus valinnalla 0
    b. Valintarakenteen viimeinen `else`-haara käsittelee tuntemattomat valinnat
32. Valintarakenteen haarassa tulee tehdä kaikki valintaan liittyvät toimenpiteet alusta loppuun asti mahdollisesti kutsuen useita eri aliohjelmia
33. Toistorakenteen viimeinen käsky on valintarakenteen jälkeen tyhjän rivin tulostus
34. Ohjelman kaikki lopetusrutiinit ovat toistorakenteen jälkeen ennen pääohjelman loppua

Valikkopohjaisen ohjelman valikko-aliohjelman rakenne

35. Aliohjelma ei saa parametrejä ja se palauttaa käyttäjän valinnan kokonaislukuna
36. Valikon jokainen rivi tulostetaan omalla `print`-käskyllä
37. Käyttäjän valinnat numeroidaan alkaen luvusta 1 ja viimeisenä on Lopeta-valinta numerolla 0; numeron jälkeen on aina)-merkki ja välilyönti

Valikkopohjaisen ohjelman pääohjelman standardifraasit

38. `"Tuntematon valinta, yritä uudestaan."`
39. `"Lopetetaan."`
40. `"Kiitos ohjelman käytöstä."`

Valikkopohjaisen ohjelman valikko-aliohjelman standardifraasit

41. `"Valitse haluamasi toiminto:"`
42. `"0) Lopeta"`
43. `"Anna valintasi: "`

Luvun asiat kokoava esimerkki

Esimerkissä 5.14 näkyy luvun kokoava esimerkki, joka on muokattu luvun 4 kokoavasta esimerkistä 4.8 lisäämällä siihen pää- ja aliohjelmat. Ohjelman toiminnallisuus ei ole muuttunut vaan edellisestä ohjelmasta on editoitu eli muokattu aliohjelmarakenne.

Esimerkki 5.14. Kokoava esimerkki

```
# Aliohjelmat
def valikko():
    print("Valitse haluamasi toiminto:")
    print("1) Kysy merkkijono")
    print("2) Tulosta merkkijono etuperin")
    print("3) Tulosta merkkijono takaperin")
    print("0) Lopeta")
    Valinta = int(input("Anna valintasi: "))
    return Valinta

def kysyMerkkijono():
    Syote = input("Anna merkkijono: ")
    return Syote

def tulostaEtuperin(Merkit):
    print(Merkit)
    return None

def tulostaTakaperin(Merkit):
    print(Merkit[::-1])
    return None

# Pääohjelma viimeisenä aliohjelmien alla/jälkeen
def paaohjelma():
    Valinta = 1
    while (Valinta != 0):
        Valinta = valikko()

        # Valintarakenne
        if (Valinta == 1):
            Merkkijono = kysyMerkkijono()
        elif (Valinta == 2):
            tulostaEtuperin(Merkkijono)
        elif (Valinta == 3):
            tulostaTakaperin(Merkkijono)
        elif (Valinta == 0):
            print("Lopetetaan.")
        else:
            print("Tuntematon valinta, yritä uudestaan.")
        print()
    print("Kiitos ohjelman käytöstä.")
    return None

# Pääohjelmakutsu päätasolla
paaohjelma()
# eof
```

Tuloste

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

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

```
ittaM-ahuJ

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

Kiitos ohjelman käytöstä.
```