



CT10A0013

Ohjelmointi Pythonilla

L05: Ohjelman rakenne

Uolevi Nikula



Päivän asiat

- Teoria, rakenteinen ohjelmointi
- Käytäntö
 - Koodilohkon nimeäminen
 - Aliohjelmien käsitteitä ja rakenne
 - Tiedonsiirto aliohjelmien välillä
 - Tunnukset ja näkyvyys
 - Miksi tehdä aliohjelmia
- Koodiesimerkkejä
- Lopuksi
- Täydennyksiä lukuun 5



Teoria

Rakenteinen ohjelmointi
Aliohjelmat ja niiden käyttö



Rakenteinen ohjelmointi

- Ohjelmat voidaan kirjoittaa “yhteen pötköön”
 - Lopputulos on usein spagettikoodia, jota on vaikea ymmärtää, seurata ja muokata
- 1960-luvulla ruvettiin puhumaan rakenteisesta ohjelmoinnista, jossa ohjelman perusrakenteiksi muodostuivat
 - Valintarakenteet
 - Toistorakenteet
 - Koodilohkot
 - **Nimetyt koodilohkot eli aliohjelmat**
- Ohjelman rakenteen kannalta keskeisiä asioita ovat
 - Pääohjelma
 - Aliohjelmat
 - Tiedonvälitys em. ohjelmien välillä parametreilla ja paluuarvoilla



Aliohjelmat ja niiden käyttö

- Olet käyttänyt aliohjelmia tämän kurssin kaikissa tehtävissä, esim.
 - **print**-aliohjelma tulostaa parametrinsa näytölle
 - **input**-aliohjelma tulostaa parametrina olevan tekstin näytölle ja palauttaa käyttäjän antaman merkkijonon
 - **len**-aliohjelma laskee parametrin pituuden ja palauttaa sen
 - **int**, **float** ja **str** –aliohjelmat muuttavat parametrinsa halutun tyyppiseksi ja palauttavat sen
- Aliohjelmiin liittyy monia asioita, joita käydään läpi seuraavilla kalvoilla
 - Tämän viikon tehtävät ovat perustehtäviä, jotka ovat vaikeudeltaan vastaavia kuin em. aliohjelmien käyttö
 - Mikäli teoria tuntuu vaikealta, tai sitä on liian paljon, kannattaa keskittyä tehtävien tekoon ja käyttää apuna ohjelmointiopasta ja -videoita
 - Perustehtävien tekeminen on usein helpompaa kuin kattava teorian opiskelu



Käytäntö

Koodilohko

Käsitteitä ja rakenne

Tiedonsiirto

Tunnukset ja näkyvyys

Miksi tehdä aliohjelmia



Koodilohkon nimeäminen

Nimetty koodilohko
Excel-makro



Koodilohkosta nimettyyn koodilohkoon

- Koodilohko tuttu asia: kaksoispiste + sisennys
- Koodilohko on yksi kokonaisuus
 - valintarakenteessa
 - toistorakenteessa
- Koodilohkon laajennus on tehdä siitä **nimetty** koodilohko, jonka jälkeen koodilohko voidaan suorittaa tuota nimeä **kutsumalla**
- Pythonissa nimettyä koodilohkoa kutsutaan **aliohjelmaksi**

Koodilohko



```
Haluan = input("Haluatko laskea keskiarvon (k/e): ")
if (Haluan == 'k'):
    Summa = 0
    Lkm = int(input("Montako lukua huomioidaan: "))
    for i in range(Lkm):
        Summa = Summa + int(input("Anna luku: "))
    print("Keskiarvo on", Summa / Lkm)
print("Kiitos ohjelman käytöstä.")
```

Nimetty koodilohko keskiarvo



```
def keskiarvo():  
    Summa = 0  
    Lkm = int(input("Montako lukua huomioidaan: "))  
    for i in range(Lkm):  
        Summa = Summa + int(input("Anna luku: "))  
    print("Keskiarvo on", Summa / Lkm)  
    return None
```

```
Haluan = input("Haluatko laskea keskiarvon (k/e): ")  
if (Haluan == 'k'):  
    keskiarvo()  
print("Kiitos ohjelman käytöstä.")
```

Käskysekvenssin nimeämisestä yleisesti



- Nimetyt käskysekvenssit ja niiden uudelleenkäyttö on yleinen tapa automatisoida rutiineja ja tehostaa työtä esim. ohjelmistotestauksessa (vrt. CodeGrade), CAD-työkaluissa ja Excelissä
- Nimettyjä käskysarjoja hyödynnetään myös yleisemmin
 - Esim. LUTin opiskelun sivuilta löytyy paljon toimintaohjeita erilaisiin tilanteisiin
 - Opetuksen tutkintosääntö
 - Ohjeita diplomityötä suunnittelevalle opiskelijalle
 - Kuulusteluja koskevat ohjeet opiskelijoille
 - Opintojen keskeyttäminen
 - Toimenpideohjeet vilppitapauksissa
 - ...
 - Samoin toimintaohjeita on esim. tulipalon varalle jne.
- Pythonissa käskysekvenssejä voi nimetä ja tällöin muodostuu nimetty koodilohko eli **aliohjelma**



Nimetty käskysekvenssi Excelissä eli makro

- Excelissä voidaan nauhoittaa makroja, joilla voidaan automatisoida rutiinitehtäviä
 - Nauhoituksen aluksi makrolle annetaan nimi
 - Kun käskysarja halutaan suorittaa, valitaan makro-työkalusta halutun makron nimi ja suorita-käsky
 - Excel tekee täsmälleen samat toimenpiteet kuin makroa nauhoitettaessa tehtiin



Aliohjelmien käsitteitä ja rakenne

Tunnusten nimeäminen

Aliohjelmiin liittyviä termejä

Aliohjelman ja tiedoston rakenne



L01: Muuttujien nimeämisestä

- Aliohjelmien ja muuttujien nimet ovat **tunnuksia**, joiden käytölle on selkeät ja yhteiset säännöt
 - Alettava kirjaimella tai alaviivalla '_'
 - Nimessä voi olla kirjaimia (isoja ja pieniä) sekä numeroita (0-9)
 - Ei ääkkösiä eli skandinaavisia merkkejä eikä erikoismerkkejä
 - (Python hyväksyy, muut kielet ei hyväksy → älä käytä)
 - Isot ja pienet kirjaimet eri asia
 - Pythonissa tunnukset "Talo" ja "talo" ovat kaksi eri asiaa kuten puhekielessä "talo" ja "valo"



Aliohjelmiin liittyviä termejä 1

- **Pääohjelma** – ensisijaisesti suoritettava koodilohko eli ohjelma
 - C-pohjaisissa kielissä käyttöjärjestelmä kutsuu `main()` –ohjelmaa, ts. pakollinen
 - Pythonissa päätason koodi (ei sisennetty) toimii pääohjelmana
- **Aliohjelma** – nimetty koodilohko, jota voi kutsua muista ohjelmista
 - Aliohjelmia voi olla vapaavalintainen määrä, 0-N
 - Aliohjelmia käytetään pilkkomaan koodi pienempiin ja paremmin hallittaviin osiin
- **Funktio** – aliohjelma, joka palauttaa arvon
 - On olemassa myös aliohjelma, joka ei palauta arvoa – tämä on *proseduuri*
 - Käytännössä tällä kurssilla aliohjelma ja funktio ovat synonyymeja eikä proseduureista puhuta tätä mainintaa enempää



Aliohjelmiin liittyviä termejä 2

- **Paluuarvo** – aliohjelman tapa palauttaa pääohjelmalle tietoa (esim. `return Nimi`)
- **Parametri** – aliohjelman välitettyjen arvojen nimitys aliohjelman määrittelyssä ja sisällä (esim. `def Laske(Luku1, Luku2) :`)
- **Argumentti** – aliohjelmalle välitettävien arvojen nimitys, ts. aliohjelmaa kutsutaan argumenteilla
 - Argumentti-käsitettä käytetään harvoin, lähinnä kun halutaan korostaa eroa aliohjelmakutsun ja aliohjelman koodin välillä



Aliohjelman rakenne

- Aliohjelman rakenneosat ovat seuraavat

Varattu sana Ohjelman nimi Sulut Kaksoispiste

def

Aliohjelma

()

:

print("suoritettava käsky 1")

print("suoritettava käsky 2")

print("suoritettava käsky 3")

return None

Sisennetyt käskyt

**Aliohjelman loppu eli paluu kutsuvaan
(ali)ohjelmaan**



Lähdekooditiedoston rakenne

- Kun ohjelma koostuu useista ohjelmalohkoista, kannattaa ohjelmalle muodostaa ”pääohjelma”, joka vastaa varsinaisen tehtävän koordinoinnista
 - Noudattaa C-kielen filosofiaa eli ohjelman suoritus alkaa main-ohjelmasta
 - Tiedostoon laitetaan päätasolle eli ei-sisennetylle tasolle vain tämän pääohjelman kutsu
 - **Tällä kurssilla ohjelman nimen tulee olla aina `paaohjelma()`**
- Muut aliohjelmat tulee sijoittaa tiedostossa ennen pääohjelmaa, koska kutsuttava ohjelma on määriteltävä ennen sen kutsua



Esimerkki paaohjelma():sta

```
def paaohjelma(): # Määritelty ennen kutsua
    print("Yksinkertainen pääohjelma eli paaohjelma()")
    print("Kiitos ohjelman käytöstä.")
    return None
```

```
paaohjelma() # Päätasolla on vain kutsu paaohjelmaan
```



Tiedonsiirto ohjelmien välillä

Parametrit

Paluuarvot



Ohjelmien välinen tiedonsiirto

- Aliohjelmien **parametrit**
 - Tietoa viedään aliohjelmiin parametreilla
 - Oletusarvoisesti aliohjelmakutsussa olevat argumentit sijoitetaan aliohjelman parametreiksi järjestyksen perusteella
 - Parametrien järjestystä voidaan muuttaa käyttämällä avainsanoja
 - Parametreille voidaan antaa oletusarvoja
 - Tässä vaiheessa kaikki Pythonin parametrit ovat arvoparametreja, joiden arvon muutos ei näy kutsuvassa ohjelmassa. Katso tarkemmin Arvoparametrit-kalvolta
- Aliohjelmista palautetaan tietoa **paluuarvoilla**



Parametrien järjestys

- Aliohjelmia kutsuttaessa annetut argumentit ja aliohjelman parametrit yhdistetään niiden järjestyksen perusteella
- Jos parametreja on monta, ne erotetaan toisistaan pilkuilla

```
def Tulosta(Eka, Toka, Kolmas):  
    print(Eka, Toka, Kolmas)
```

```
a = "eka"  
b = 1  
c = 2.34
```

```
Tulosta(a, b, c) # kokeile eri järjestyksiä
```



Järjestyksellä on väliä – ei nimillä!



Parametrien nimeäminen

- Pythonissa aliohjelman parametrien järjestystä voi muuttaa, jos aliohjelmakutsussa kerrotaan, minkä nimiselle parametrille arvo halutaan antaa, esim.

```
def Tulosta(Nimi, Ika):  
    print(Nimi, Ika)
```

```
Nimi1 = "Ville"
```

```
Vuosia = 8
```

```
Tulosta(Ika=vuosia, Nimi=nimi1)
```

- Oleellista parametrien välityksessä ja ohjelmoinnissa yleensäkin on **yksikäsitteisyys eli tarkoituksen on oltava selvä**
- Huom. print-käskyn yhteydessä on jo käytetty nimettyjä parametreja *end* ja *sep*

Arvoparametrit



- Pythonissa aliohjelman parametrit ovat kopioita kutsun argumenteista
 - Koskee tähän mennessä läpikäytyjä yksinkertaisia tietotyyppisiä eli numeroita ja merkkijonoja; asia tarkentuu myöhemmin
- Aliohjelmassa muuttujan kopioon tehdyt muutokset eivät muuta alkuperäistä muuttujaa

```
def tulosta(Etunimi, Ika):  
    print(Etunimi, Ika)  
    Etunimi = "Kalle"  
    Ika = Ika + 1  
    print(Etunimi, Ika)
```

```
Nimi1 = "Ville"  
Vuosia = 9  
print(Nimi1, Vuosia)  
tulosta(Nimi1, Vuosia)  
print(Nimi1, Vuosia)
```




Tunnukset ja näkyvyys

Tunnukset

Nimiavaruus

Näkyvyys

Tunnuksista



- Tähän asti **tunnus** on tarkoittanut **muuttujan nimeä**
 - Kyseessä on tunnus eli nimi säiliölle, johon on tallennettu arvo, ja johon halutaan päästä käsiksi myöhemmin
- Nimetty koodilohko on **aliohjelma**, jonka nimi on myös tunnus
 - Aliohjelma on koodiosio, jota halutaan pystyä kutsumaan myöhemmin nimellä
- Yksi muuttujien rooli on **kiintoarvo**
 - Kiintoarvo tarkoittaa sitä, ettei muuttujan arvo muutu
 - Monissa muissa ohjelmointikielissä kiintoarvo-roolin toiminta on varmistettu erillisellä tunnukseella **vakio**. Tällöin kääntäjä/tulkki estää vakion arvon muuttumisen ohjelman suorituksen aikana
 - Pythonissa ei ole vakioita, joten on tultava toimeen kiintoarvo-muuttujilla ja ohjelmoijan on huolehdittava, ettei niiden arvo muutu ohjelman suorituksen aikana

Nimiavaruudet



- Aliohjelma on itsenäinen ohjelma ja sillä on oma *nimiavaruus*. Tämä tarkoittaa sitä, että eri aliohjelmissa voi olla saman nimisiä muuttujia, koska yksi aliohjelma näkee vain omassa koodilohkossa (nimiavaruudessa) olevat muuttujat
 - Vrt. esim. huone – eri huoneissa voi olla lamppu, pöytä ja kaappi, ja huone tekee esineistä yksikäsitteisiä, esim. *olohuoneen* pöytä tai *keittiön* pöytä
 - Jos samassa huoneessa on kaksi pöytää, ne nimetään yleensä yksikäsitteisesti eli esim. *tiskipöytä* ja *ruokapöytä* jne.
- Yhdessä aliohjelmassa käytettävien muuttujien nimien on oltava yksikäsitteisiä
 - Eri aliohjelmat eivät ”näe” toisten aliohjelmien muuttujia (nimiavaruutta)
 - Eri aliohjelmien välillä voidaan välittää tietoa parametreilla ja paluuarvoilla



Nimiavaruus, tunnukset ja näkyvyys

Tunnus	Näkyvyys
Muuttuja	Lokaali
Kiintoarvo (Vakio)	Lokaali tai Globaali
Aliohjelma	Globaali

- **Lokaali** – paikallinen, näkyy yhden aliohjelman sisällä
- **Globaali** – näkyy koko tiedostossa, *määritelty tiedoston päätasolla*
- Hyvään virheitä välttävään ohjelmointityyliin kuuluu, *ettei globaaleja muuttujia käytetä*
- Jotkut kirjastot perustuvat globaaleihin muuttujiin, jolloin niiden käyttöä ei voi välttää ja tällöin se on hyväksyttävää
- **Tällä kurssilla globaalien muuttujien käyttö on kielletty ellei erikseen käsketä käyttämään niitä**



Miksi tehdä aliohjelmia?



Miksi tehdä aliohjelmiä 1/2

- Koodin rakenteen selkeyttäminen
 - Eri asioita tehdään selkeästi eri paikoissa, esim. tiedoston luku ja kirjoitus
- Aliohjelmat hävittävät toteutuksen yksityiskohdat
 - Määrittely kertoo aliohjelman nimen ja parametrit (print, input, len, ...)
 - Ohjelmoija voi katsoa dokumentaatiosta aliohjelman toiminnallisuuden
- Aliohjelmat tarjoavat yhden luonnollisen lähestymistavan ohjelmointityön jakamiseksi
 - Eri henkilöt ohjelmoivat eri aliohjelmiä
 - Rajapinnat sovitaan yhteisesti



Miksi tehdä aliohjelmia 2/2

- Uudelleenkäyttö
 - Tarvitaan samaa toimintoa toistuvasti, esim. yksikkömuunnoksia (valuutta, lämpötila, paino, pituus, jne.)
 - Tarve voi olla saman ohjelman sisällä tai laajemmassa käyttäjäkunnassa
 - Esim. kaikissa ohjelmissa ei ole omaa pdf-tiedostojen katselutoimintoa vaan tähän käytetään yleisesti Adobe Acrobat Reader -ohjelmaa
 - Nimettyjä aliohjelmia on helppo kutsua suorittamaan tehtäviä eri ohjelmista sekä Pythonin komentotulkista
 - Aliohjelmat ovat yksi tärkeimmistä uudelleenkäytön muodoista



Koodiesimerkkejä



Esimerkki ohjelman rakenteesta

```
def tulosta(Lkm):  
    print(Lkm)  
    Lkm = Lkm + 1  
    print(Lkm)  
    return None
```

```
def paaohjelma():  
    Luku = 1  
    print(Luku)  
    tulosta(Luku)  
    print(Luku)  
    return None
```

```
paaohjelma()
```

Valikkopohjainen ohjelma aliohjelmina

```
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()
```

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

paaohjelma on tässä aliohjelmien alla/jälkeen



Aliohjelman suoritus – demo videolla

<http://pythontutor.com/>

```
def aliohjelma():  
    Nimi = input("Anna nimi: ")  
    print("Nimi on", Nimi)  
    Pituus = len(Nimi)  
    print("Nimi on " + str(Pituus) + " merkkiä pitkä.")  
    return None  
  
aliohjelma()  
aliohjelma()  
aliohjelma()  
print("Kiitos ohjelman käytöstä.")
```



Lopuksi

Osaamistavoitteet



Osaamistavoitteet

- Aliohjelmat
 - Miksi tehdä
 - Miten tehdä
 - Miten käyttää
 - paaohjelma():n ja def-return:n käyttö
- Tiedon välitys aliohjelmien välillä
 - Parametrit ja paluuarvo
- Nimiavaruus, tunnukset ja näkyvyys
 - Aliohjelmat, kiintoarvot, muuttujat
 - Lokaali vs. globaali



Täydennyksiä oppaan lukuun 5

Tyyliohjeita pienille Python-ohjelmille
Oppaan esimerkit ja käsitellyt asiat



Pienen Python-ohjelman tyyliohjeet 1

- **Nimiavaruus**
 - **Muuttujat aina lokaaleja.** Muuttujat tulee määritellä aina aliohjelman sisällä, jolloin ne näkyvät vain ko. aliohjelman sisällä
 - **Aliohjelmat aina globaaleja.** Aliohjelmat tulee määritellä aina päätasolla eli globaaleiksi, jotta niitä voi kutsua mistä tahansa
 - **Kiintoarvot tyypillisesti globaaleja.** Kiintoarvot ovat tyypillisesti hyödyllisimpiä, kun ne määrittelee globaaleiksi eli näkymään kaikkialla ohjelmassa. Kiintoarvon arvo asetetaan ohjelman alussa eikä se saa muuttua ohjelman suorituksen aikana



Pienen Python-ohjelman tyyliohjeet 2

- Aliohjelmat
 - Tällä kurssilla isoissa ohjelmissa tulee olla pääohjelma `paaohjelma()`, jolloin päätasolla ei ole muita käskyjä tämän `paaohjelma()` -kutsun lisäksi. Päätasolla on myös aliohjelmien ja kiintoarvojen *määrittelyt*
 - Tiedostossa määritellään ensin/ylimmäisenä kiintoarvot, sitten aliohjelmat ja viimeisenä `paaohjelma()`
 - Aliohjelma alkaa `def`-sanalla ja se päättyy `return`-käskyyn
 - Tiedonvälitys aliohjelmaan ja sieltä takaisin tapahtuu parametreja ja paluuarvoa käyttäen
 - Mikäli aliohjelma palauttaa jonkun arvon, se tulee `return` -käskyn perään ja jos aliohjelma ei palauta arvoa, käytetään `return None` -käskyä



ASPA – staattinen analysaattori

- Ohjelmoinnin perusteet –kurssin tehtävien tekemisen tueksi on oma työkalu, ASPA
- ASPA lukee valmiin Python-ohjelman ja analysoi kurssilla läpikäytyjen ohjeiden noudattamisen, esim. L05 pohjalta ASPA etsii aliohjelmiin ja nimiavaruuksiin liittyviä virheitä sekä varoittaa niistä
- ASPA ei suorita ohjelmaa ja on siten *staattinen analysaattori*
- ASPA on saatavilla Moodlessa ja sitä saa käyttää omien ohjelmien tarkistamiseen ja korjaamisen
 - Käyttö on vapaaehtoista
 - Työkalusta kerätään palautetta sen kehitystyötä varten
 - ASPA on yhdessä tiedostossa oleva yksi Python-ohjelma
- Henkilökunta käyttää ASPAa harjoitustöiden ja tenttivastausten arvioinnin lähtökohtana. Käytännössä ASPAn tarkastuksesta ilman huomautuksia läpi menneen ohjelman pitäisi mennä läpi myös henkilökunnan arvioinnista



ASPAn tarkistukset L05

- Ohjelman rakenteeseen, aliohjelmiin ja näkyvyyteen liittyen ASPA tarkistaa ohjelmasta seuraavat asiat
 - Kiintoarvot on määrittely globaaleiksi tiedoston alussa
 - Aliohjelmat on määritelty globaaleiksi kiintoarvojen jälkeen
 - `paaohjelma()` on määritelty aliohjelmien jälkeen
 - Tiedoston lopussa päätasolla on vain yksi ohjelmakutsu `paaohjelma()`
 - Jokaisen aliohjelman lopussa on `return` ja yksi paluuarvo, tarvittaessa `None`
 - Paluuarvona ei ole vakiota, esim. `return "hei"`, koska tyypillisesti paluuarvo on muuttuja
 - Aliohjelmakutsussa on sama määrä parametreja kuin aliohjelman määrittelyssä
 - Aliohjelma ei kutsu itseään suoraan ja epäsuorasta eli siinä ei saa olla rekursiota. Tällä kurssilla tulee käyttää `for` ja `while` –toistorakenteita paitsi erikseen mainituissa rekursio-tehtävissä ja esimerkeissä. Rekursioon palataan myöhemmin

ASPA:n tarkistukset liittyen luentoihin 1-4



- ASPA auttaa hyvän ohjelmointityylin noudattamisessa ja varoittaa, jos se epäilee, ettei seuraavia ohjeita ole noudatettu
 - Tunnusten nimissä ei tule olla ääkkösiä
 - Ohjelmassa ei saa olla koodia, jota ei voi koskaan suorittaa, esim. valintarakenteessa
 - Ohjelmassa ei saa olla ikisilmukoita, joiden pysäyttäminen ei onnistu



Käsitellyt asiat oppaan luvussa 5

- Funktio: Esimerkki 5.1, 5.7
- Parametrit: Esimerkki 5.2, 5.7
- Paluuarvo: Esimerkki 5.3, 5.4, 5.7
- Nimiavaruus: Esimerkki 5.5, Taulukko 5.1
- Ison ohjelman rakenne: Esimerkki 5.6, 5.7 – Huom. tällä kurssilla “iso”
- Funktioiden dokumentaatorivi
- Pythonin tarjoamia hyödyllisiä funktioita