LUT Python ohjelmointiopas 2023 - osa 7
Sisällysluettelo

Luku 7: Rakenteiset tietorakenteet

Tietorakenteet ovat yksinkertaisesti sanottuna juurikin itseään tarkoittava asia – eli ne ovat rakenteita, jotka sisältävät tietoa. Olemme tähän mennessä käyttäneet Pythonin ”yksinkertaisia” tietorakenteita kokonaisluku ja desimaaliluku, joissa muuttuja viittaa suoraan koko tietoalkioon ja ohjelmoijan ei tarvitse tietää tai välittää sen sisäisestä rakenteesta. Olemme myös käyttäneet merkkijonoa, jossa muuttujan nimi viittaa lukujen tavoin koko merkkijonoon, mutta lisäksi voidaan viitata muuttujan osiin eli merkkeihin yksitellen ja siitä voidaan erottaa haluttaessa leikkaus eli osa jatkokäsittelyyn.

Pythonissa on myös tietorakenteita, joissa on ohjelmoijan tiedossa oleva sisäinen rakenne eli ne sisältävät useita tietoalkioita, jotka liittyvät toisiinsa ja ovat täten osa samaa kokonaisuutta. Näistä rakenteisista tietorakenteista yleisimpiä ovat lista, luokka/olio, tuple ja sanakirja. Tässä luvussa keskitymme listaan ja luokkaan, mutta esittelemme lyhyesti myös listaa muistuttavan tuplen. Sanakirja on läheistä sukua listalle ja palaamme siihen luvussa 10. Tämän luvun päätteeksi katsomme kokoavassa esimerkissä ohjelmaa, jossa on toteutettu tyypillisen perusohjelman käyttöliittymä ja tiedostonkäsittely käyttäen hyväksi ajonaikaisia tietorakenteita lista ja luokka sekä tiedon pysyvään tallennukseen tekstitiedostoa.

Lista

Lista on Pythonin perustietorakenne, joka sisältää joukon alkioita. Tähän mennessä olemme tutustuneet muuttujiin, jotka ovat pystyneet pitämään sisällään vain yhden alkion kerrallaan, esimerkiksi numeron 42 tai sanan ”suklaakakku”. Lista pystyy pitämään sisällään useita numeroita tai merkkijonoja ja jopa niiden sekoituksia. Pythonin lista vastaa reaalimaailman vastaavaa. Jos kirjoitat paperille listan asioista, esim. ostoslistan, niin todennäköisesti listaat asiat allekkain. Python-listan idea on sama ja erilaisesta teknisestä toteutuksesta huolimatta lopputulos on alkioista koostuva kokonaisuus, jota kutsutaan listaksi.

Listan määrittelyssä listan alku- ja loppukohta merkitään hakasuluilla. Tämä antaa tulkille ilmoituksen siitä, että haluat määritellä listan, ja että annat sille pilkulla eroteltuja alkioita. Kun olet luonut listan, voidaan siihen tämän jälkeen lisätä alkioita, poistaa alkioita, muuttaa järjestystä sekä hakea alkioita. Toisin sanoen listaa voidaan muokata vapaasti toisin kuin esimerkiksi merkkijonoja. Kuten aiemmin tiedostojen ja merkkijonojen kanssa, myös listalla on jäsenfunktioita.

Esimerkissä 7.1 näkyy listan peruskäyttö. Koska listalla on jäsenfunktioita, pitää se määritellä ennen sen käyttöä ja tyhjän listan määrittely tehdään hakasuilla. Listaan lisätään alkioita `append`-jäsenfunktioilla, jonka jälkeen listan alkioita voidaan käydä läpi for rakenteella. Käytön päätyttyä lista tulee tyhjentää eli sen varaama muisti tulee vapauttaa. Nämä kaikki operaatiot näkyvät esimerkissä 7.1.

Esimerkki 7.1. Listan peruskäyttö

```
Tiedot = []           # lista-tyyppisen muuttujan määrittely

for i in range(10):
    Tiedot.append(i)  # tietojen lisäys listaan, append

for Alkio in Tiedot:  # listan alkioiden läpikäynti
    print(Alkio, end=" ")

Tiedot.clear()        # listan tyhjennys
```

Tuloste

```
0 1 2 3 4 5 6 7 8 9
```

Listaan voi tallettaa mitä tahansa tietoa. Käytännössä listaan kannattaa lisätä aina samanlaisia tietoalkioita, jotta haettaessa tietoa listasta on selvää, mitä sieltä tulee. Tässä vaiheessa listaan laitetaan tyypillisesti numeroita eli kokonaislukuja esimerkin 7.1 mukaisesti, desimaalilukuja tai merkkijonoja.

Esimerkissä oli hieman aiemmasta poikkeava `for...in`-toistorakenne listan läpikäymiseen. Lista on eräänlainen sarja (sequence), joten sitä voidaan käyttää yksinään `for`-lauseen määrittelemisessä. Jos annamme `range`-funktion tilalle listan, `for`-lause käy läpi kaikki listan alkiot.

Listan voi tulostaa yksinkertaisesti `print`-funktiolla. Tällöin lista tulostuu sarjamuodossa sulkujen ja heittomerkkien kanssa pilkut alkioiden välissä. Siksi tätä tulee käyttää vain testausvaiheessa ja valmiissa ohjelmassa listan alkiot tulostetaan aina erillisinä alkioina.

Listan käytön loputtua se tulee aina tyhjentää `clear()`-jäsenfunktiolla. Tämä mahdollistaa muuttujan käytön myöhemmin eli uusien alkioiden laittamisen listaan jne., mutta vapauttaa tietoalkioille varatun muistin.

Listan yleisimpiä jäsenfunktioita

Taulukossa 7.1 on yleisiä jäsenfunktioita listan sisällön muokkaamiseen. Näitä kannattaa opetella käyttämään tarpeen mukaan ja hyvään alkuun pääsee `append` ja `clear` -jäsenfunktioilla esimerkin 7.1 mukaisesti. Jäsenfunktiot on tehty tehokkaiksi eli niillä saa tehtyä monia asioita nopeasti ja siten ne ovat opetteluun menevän ajan arvoisia, kunhan ymmärtää mihin kutakin jäsenfunktiota voi käyttää.

Taulukko 7.1. Pythonin listan yleisiä jäsenfunktioita.

Jäsenfunktio	**Kuvaus**
append(*x*)	Lisää alkio x listan loppuun.
clear()	Tyhjentää listan ja vapauttaa muistin, mutta jättää muuttujan käyttöön.
extend(*L*)	Lisää listaan kaikki annetun listan L alkiot. Eroaa append:ista siten, että testi.append(L) lisäisi listan testi viimeiseksi alkioksi listan L, kun taas testi.extend(L) lisää listan L alkiot listan testi loppuun.
insert(*i, x*)	Lisää alkion x listalle kohtaan i. Listan alkuun lisääminen tapahtuisi käskyllä a.insert(0, x), ja loppuun lisääminen – samoin kuin append tekee – tapahtuisi käskyllä a.insert(len(a), x).
remove(*x*)	Poistaa listalta ensimmäisen alkion, jonka arvo on x, eli siis jossa x == lista[i] == True. Palauttaa virheen, mikäli tämän arvoista alkiota ei ole olemassa.
pop(i)	Poistaa listalta tietueen kohdasta i ja palauttaa sen arvon. Mikäli i on määrittelemätön, poistaa se viimeisen listalla olevan alkion.
index(*x*)	Palauttaa numeroarvon, joka kertoo millä kohdalla listaa on alkio, jolla on arvo x. Palauttaa virheen mikäli lista ei sisällä alkiota, jonka arvo on x.
count(*x*)	Palauttaa numeroarvon, joka kertoo kuinka monta kertaa x esiintyy listalla.
sort()	Järjestää listan alkiot arvojärjestykseen.
reverse()	Järjestää listan alkiot käänteiseen järjestykseen, eli ensimmäinen viimeiseksi jne.

Esimerkissä 7.2 näkyy tyypillisiä listan operaatioita ja usein käytettyjä jäsenfunktioita. Esimerkissä 7.1 määrittelimme tyhjän listan ja lisäsimme siihen alkioita `append`-jäsenfunktiolla. Listan sisällön voi antaa myös muuttujan määrittelyn yhteydessä esimerkin 7.2 mukaisesti. Huomaa, että lista-erottimena käytetään pilkkua ja merkkijonot pitää laittaa lainausmerkkeihin. Esimerkin mukaisesti listoista voi ottaa leikkauksia vastaavalla tavalla kuin merkkijonosta eli `[::]` -operaattorilla. Alkioiden lisäys tapahtuu `append`-jäsenfunktiolla ja tietoalkioita voi poistaa `remove`-jäsenfunktiolla, joka poistaa listasta löytyvän ensimmäisen alkion, jolla on pyydetty arvo. Tämä ei ole aina haluttu tulos, joten listasta voi poistaa alkioita myös indeksin avulla käyttämällä `del`-käskyä.

Esimerkki 7.2. Listan operaatioita ja jäsenfunktioiden käyttö

```
print("Määrittely ja lisäys:")
Tiedot = [1, 4, 3, 8, 11, 2, 12, 23]
print(Tiedot)
Tiedot.append(20)
print(Tiedot, "- Lisätty 20 loppuun")
print()

print("Leikkausten ottaminen listasta:")
print(Tiedot[0])
print(Tiedot[1:6])
print(Tiedot[::-1])
print()

print("Lista on olio ja sillä on jäsenfunktioita kuten lajittelu:")
print(Tiedot)
Tiedot.sort()
print(Tiedot)
print()

print("Tietoalkion poistaminen listasta - yksittäinen tietoalkio:")
Tiedot.append(12)
print(Tiedot, "- Lisätty loppuun 12")
Tiedot.remove(12)
print(Tiedot, "- Ensimmäinen tietoalkio arvolla 12 poistettu")
del Tiedot[2]
print(Tiedot, "- Indeksillä 2 oleva alkio poistettu")
print()

print("Lista on tyhjennettävä käytön lopuksi:")
print(Tiedot)
Tiedot.clear()
print(Tiedot)
```

Tuloste

```
Määrittely ja lisäys:
[1, 4, 3, 8, 11, 2, 12, 23]
[1, 4, 3, 8, 11, 2, 12, 23, 20] - Lisätty 20 loppuun

Leikkausten ottaminen listasta:
1
[4, 3, 8, 11, 2]
[20, 23, 12, 2, 11, 8, 3, 4, 1]

Lista on olio ja sillä on jäsenfunktioita kuten lajittelu:
[1, 4, 3, 8, 11, 2, 12, 23, 20]
[1, 2, 3, 4, 8, 11, 12, 20, 23]
```

```
Tietoalkion poistaminen listasta - yksittäinen tietoalkio:
[1, 2, 3, 4, 8, 11, 12, 20, 23, 12] - Lisätty loppuun 12
[1, 2, 3, 4, 8, 11, 20, 23, 12] - Ensimmäinen tietoalkio arvolla 12 poistettu
[1, 2, 4, 8, 11, 20, 23, 12] - Indeksillä 2 oleva alkio poistettu

Lista on tyhjennettävä käytön lopuksi:
[1, 2, 4, 8, 11, 20, 23, 12]
[]
```

Listan voi järjestää `sort`-jäsenfunktiolla, joka asettelee listan alkiot arvojärjestyksen mukaisesti pienimmästä suurimpaan. Tässä tapauksessa listan järjestys näyttää päällisin puolin olevan aakkosjärjestyksen mukainen, mutta tässä asiassa on joitakin poikkeuksia, joihin palaamme myöhemmin. Kannattaa myös huomata, että `sort`:lla järjestettyä listaa ei tarvitse erikseen tallentaa uuteen muuttujaan vaan jäsenfunktiot vaikuttavat suoraan käsiteltävään listaan, toisin kuin esimerkiksi tyyppimuunnosten yhteydessä käytettävät funktiot `int`, `float` ja `str`.

Listan alkioiden indeksointi vastaa merkkijonojen indeksointia eli ensimmäisen alkion indeksi on 0. Listan kanssa operoidessa leikkausten suorittaminen onnistuu samalla tavoin kuin merkkijonoilla: kun merkkijonoissa leikkaamme merkkejä, listoissa leikkaamme tietoalkioita. Lisäksi alkionsisäiset leikkaukset suoritetaan ensin valitsemalla alkio, ja tämän jälkeen sille tehtävä leikkaus.

Alkioiden poistaminen listalta tapahtuu `del`-käskyllä. Yksinkertaisesti annamme `del`-käskylle arvoksi sen listan alkion, jonka haluamme poistaa, jolloin listan arvoksi jää uusi lista ilman ko. arvoa. Lisäksi lista muuttuu siten, että poistetun arvon paikalle ei jää tyhjää alkiota, vaan se yksinkertaisesti täytetään siirtämällä kaikkia seuraavia alkioita yksi paikka taaksepäin. `del`-käsky osaa poistaa yksittäisten alkioiden lisäksi listan leikkauksia. Näiden käyttöä kannattaa harjoitella, jos listoja joutuu operoimaan usein.

Merkkijonon jako listaksi `split`-jäsenfunktiolla

Kuten edellisessä luvussa oli puhetta, tiedoston rivillä voi olla useita tietoalkioita ja ohjelmassa näitä käytetään usein toisistaan riippumatta, joten rivin jako tietoalkioihin on yksi tekstitiedostojen perusoperaatio. Katsoimme aiemmin erilaisia tapoja merkkijonojen pilkkomiseen, joista toinen perustui merkkijonon leikkauksiin ja toinen merkkijonon läpikäyntiin merkki kerrallaan erotinmerkkejä etsien. Nämä molemmat tavat toimivat, mutta asiaan on olemassa myös muita ratkaisuja ja yleisin niistä on edellisessä luvussa mainittu merkkijonon `split`-jäsenfunktio. Tällöin ongelmaksi muodostui aiemmin se, että `split`-palauttaa listan ja emme tunteneet sen toimintaa, mutta tämä asia on nyt hallinnassa.

Esimerkissä 7.3 on merkkijonoja ja ne jaetaan `split`-jäsenfunktiolla tietoalkioihin. `split` jakaa merkkijonon parametrina annetusta erotinmerkistä, joka on oletusarvoisesti pilkku eli ”,”, kun taas suomalaisessa datassa se on tyypillisesti puolipiste eli ”;”. `split`:n tuloksena meillä on lista ja voimme hyödyntää sen tietoalkioita sopivalla tavalla. Usein rivillä olevat tietoalkiot sijoitetaan muuttujien arvoiksi esimerkin 7.3 mukaisesti. Tietoalkiot kannattaa muuttaa tässä vaiheessa omiin luonnollisiin tietotyyppeihin eli kokonaisluvuiksi, desimaaliluvuiksi tai jättää merkkijonoiksi, jos kyseessä on nimi tai muu merkkijono. Näin myöhemmin ohjelmassa nämä tiedot ovat käyttövalmiita, eikä niiden tietotyyppiä tarvitse miettiä.

Esimerkki 7.3 Merkkijonon jako listaksi `split`:llä erotin-merkeistä

```
# split-palauttaa listan, jossa on monta alkiota
Merkkijono = "1,2,3,4,5,5,6,8,9,10"
print("Merkkijono on: '"+Merkkijono+"'")
print("Merkkijono split:n jälkeen:", Merkkijono.split(","))
print()

# Tiedostosta luettu rivi on rivinvaihtomerkkiin päättyvä merkkijono
Rivi = "Ville;Vallaton;0404567890\n"
Tietoalkiot = Rivi.split(";")
Etunimi = Tietoalkiot[0]
Sukunimi = Tietoalkiot[1]
Puhelinnumero = Tietoalkiot[2][:-1] # Jätetään rivinvaihtomerkki pois
print("Rivi on '"+Rivi+"'")
print("Tietoalkiot-lista on:", Tietoalkiot)
print("Muuttujat ovat: "+"'"+Etunimi+"', '"+Sukunimi+"', '"+Puhelinnumero+"'")
```

Tuloste

```
Merkkijono on: '1,2,3,4,5,5,6,8,9,10'
Merkkijono split:n jälkeen: ['1', '2', '3', '4', '5', '5', '6', '8', '9', '10']

Rivi on 'Ville;Vallaton;0404567890
'
Tietoalkiot-lista on: ['Ville', 'Vallaton', '0404567890\n']
Muuttujat ovat: 'Ville', 'Vallaton', '0404567890'
```

2-ulotteinen lista eli matriisi

Toistorakenteiden yhteydessä totesimme, että ohjelmoinnissa käsitellään usein taulukko-muodossa olevaa tietoa. Taulukko voidaan toteuttaa sisäkkäisillä listoilla eli lisäämällä listaan listoja. Tämä on kuitenkin riskialtis rakenne, koska listojen koko voi muuttua. Siksi tässä oppaassa taulukot käsitellään matriisirakenteella, johon tutustumme myöhemmin.

Käytännössä listassa olevat merkkijonot muodostavat rakenteen, joka muistuttaa listassa listoja -rakennetta. Näitä ei voida välttää, sillä tällaista käytettiin jo esimerkissä 7.3, kun puhelinnumerosta otettiin viimein merkki eli rivinvaihtomerkki pois. Esimerkin tulosteissa näkyy `Tietoalkiot`-listassa olevat 3 merkkijonoa ja alla olevalla kahdella leikkausoperaattorilla saadaan puhelinnumeron sisältävästä kolmannesta merkkijonosta (ts. indeksi 2) otettua muut merkit mukaan paitsi lopusta yksi merkki (`[:-1]`).

```
Puhelinnumero = Tietoalkiot[2][:-1] # Jätetään rivinvaihtomerkki pois
```

Teoriassa listoja voi olla loputon määrä sisäkkäin, mutta käytännössä kannattaa keskittyä yksiulotteiseen listaan eli listan yhteydessä käytetään vain yhtä indeksiä. Merkkijonot ovat tähän liittyvä luonnollinen laajennos, mutta sitä pidemmälle mennessä ymmärrettävyys kärsii ja todennäköisyys virheisiin lisääntyy. Siksi niitä käyttäessä pitää tietää mitä tekee eivätkä ne ole lähtökohtaisesti sopivia rakenteita tämän oppaan laajuudessa.

Muita sekvenssitietotyyppejä

Lista on yksi Pythonin sekvenssitietotyypeistä ja toinen hyvin samantyyppinen rakenne on tuple. Emme paneudu tupleen tarkemmin, mutta perusasiat on hyvä tietää, sillä muuttujasta on helppo tehdä vahingossa listan sijaan tuple, ja siksi niiden erot on hyvä tietää.

Tuple on rakenteeltaan samanlainen kuin lista yhdellä merkittävällä erolla: tuplen alkiot ovat kiinteitä. Käytännössä tuplella ei ole jäsenmuuttujia eikä muitakaan tapoja manipuloida alkioita vaan tuplen sisältö on muuntumaton alustamisen jälkeen. Tuple sopii hyvin, kun halutaan varmistaa, että esimerkiksi funktiolle annettu sarjamainen parametri ei tule muuttumaan missään vaiheessa.

Esimerkissä 7.4 näkyy listan ja tuplen käyttö. Lista luodaan, siihen lisätään alkio `append`-jäsenfunktiolla, jonka jälkeen listan sisältö voidaan tulostaa ja lista voidaan tyhjentää. Tuple luodaan vastaavalla tavalla ja tässä vaiheessa ainoa ero on, että lista määritellään hakasuilla `[]` kun taas tuple määritellään kaarisuluilla `()`. Kuten edellä todettiin, tuplella ei ole jäsenfunktioita ja siksi `append`-jäsenfunktion kutsu johtaa kääntäjän virheilmoitukseen esimerkin tulosteen mukaisesti: ”`AttributeError: 'tuple' object has no attribute 'append'`”. Jos siis käytät listaa ja saat tämän tyylisen virheilmoituksen – jossa lukee tuple – niin tarkista, että olet määritellyt listan käyttäen hakasulkuja.

Esimerkki 7.4 Tietotyypit lista ja tuple eli [] vs. ()

```
# Listan käyttö
Data = []      # Lista määritellään hakasuluilla
Data.append(2) # Listalla on jäsenfunktiot ja niitä voi kutsua
print(Data)
Data.clear()

# Tuple muistuttaa listaa, mutta on eri rakenne, määrittely Data[] vs. Data()
Data = ()      # Tuple määritellään kaarisuluilla
Data.append(2) # Johtaa virheilmoitukseen, tuplella ei ole
jäsenfunktioita
               # AttributeError: 'tuple' object has no attribute 'append'
```

Tuloste

```
[2]
Traceback (most recent call last):
  File "C:\Users\unikula\OneDrive - LUT
University\Opetus\Opetusmateriaalit\OhjelmoinninPerusteet\Opas\Esimerkit2
02308\tst.py", line 10, in <module>
    Data.append(2) # Johtaa virheilmoitukseen, tuplella ei ole jäsenfunktioita
AttributeError: 'tuple' object has no attribute 'append'
```

Luokka/olio

Luokan ja olion peruskäyttö

Lista on hyödyllinen tietorakenne, mutta aina sen rakenne ei ole optimaalinen vaan joskus on parempi rakentaa uusi oma tietotyyppi ja määritellä sille sekä sen tietoalkioille omat nimet. Esimerkiksi käsiteltäessä henkilötietoja helpoin tapa olisi tallentaa ne muuttujaan, joka on suunniteltu ihmisten henkilötietojen käsittelyyn. Tällöin voisimme määritellä, että jokaisella ihminen-tietotyypin jäsenellä on tarvittavat muuttujat esimerkiksi etunimen, sukunimen, iän ja ammatin tallentamiseen. Python-ohjelmointikielessä tämä onnistuu luokkarakenteella, jonka tunnisteena on avainsana `class`.

Esimerkissä 7.5 määritellään ensin luokka OPISKELIJA, jonka sisältämiksi tiedoksi määrittelemme Etunimi ja Sukunimi -jäsenmuuttujat. Alustamme jäsenmuuttujat aina arvolla None, jotta tiedämme ettei niissä ole vielä oikeaa tietoa. Luokan määrittelyn jälkeen siitä tehdään muuttuja Opiskelija, mikä tarkoittaa, että nyt Opiskelija-muuttujalla on kaksi jäsenmuuttujaa, Etunimi ja Sukunimi, joille voidaan antaa halutut arvot. Esimerkissä määritellään Opiskelija-muuttujan jälkeen Opiskelija2-muuttuja samalla tavalla, mutta sen jäsenmuuttujiin annetaan eri arvot. Opiskelija ja Opiskelija2 ovat siis kaksi eri muuttujaa, joissa molemmissa on omat tiedot. Tässä esimerkissä ei näy vielä selvästi luokka/olio -rakenteiden käytön edut vaan tässä keskitytään siihen, miten niitä käytetään eli määritellään luokka, määritellään muuttuja, annetaan jäsenmuuttujille arvot ja käytetään jäsenmuuttujien arvoja.

Esimerkki 7.5. Luokan ja olion peruskäyttö

```
# Luokan eli uuden tietotyypin määrittely, jäsenmuuttujien alustus None:lla
class OPISKELIJA:
    Etunimi = None
    Sukunimi = None

# Olion eli muuttujan määrittely ja jäsenmuuttujien arvojen määrittely sekä käyttö
Opiskelija = OPISKELIJA() # Huomaa sulut luokan perässä !!
Opiskelija.Etunimi = "Kalle"
Opiskelija.Sukunimi = "Eränkävijä"
print(Opiskelija.Etunimi, Opiskelija.Sukunimi)

# Toisen muuttujan määrittely vastaavalla tavalla
Opiskelija2 = OPISKELIJA() # Huomaa sulut luokan perässä !!
Opiskelija2.Etunimi = "Ville"
Opiskelija2.Sukunimi = "Metsänkävijä"
print(Opiskelija2.Etunimi, Opiskelija2.Sukunimi)
```

Tuloste

```
Kalle Eränkävijä
Ville Metsänkävijä
```

Esimerkissä 7.5 kannattaa huomata, että määrittelemme ensin uuden rakenteisen tietotyypin eli luokan. Luokka on eräänlainen muotti, jolla tehdään varsinaisia instansseja luokasta eli olioita. Käytännössä olio vastaa tässä oppaassa muuttujaa sillä erolla, että olio sisältää aina vähintään kaksi jäsenmuuttujaa eli useita tietoalkioita. Olio määritellään käskyllä `Opiskelija = OPISKELIJA()`, eli olemme nyt luoneet olion `Opiskelija`, jolla on jäsenmuuttujina `Etunimi` ja `Sukunimi`. Huomaa, että oliota määritellessä luokan nimen perässä pitää olla tyhjät kaarisulut. Seuraavilla riveillä näille jäsenmuuttujille annetaan arvot ja ne tulostetaan `print`-käskyllä. Voimme käyttää jäsenmuuttujia normaalien muuttujien tavoin ja ainoa muistettava asia on, että pistenotaatio kertoo, minkä olion jäsenmuuttujaa käytämme. Esimerkissä 7.5 meillä on kaksi oliota, `Opiskelija` ja `Opiskelija2`, ja pistenotaation avulla kerromme kumman olion tietoja haluamme käyttää tai muuttaa. Periaate on sama kuin tiedostokahvoja käytettäessä.

Luokka, jäsenfunktiot ja olio-ohjelmointi

Luokkien yhteydessä puhutaan *jäsenmuuttujista*, joten herää kysymys liittyvätkö aiemmin oppaassa käsittelemämme *jäsenfunktiot* myös luokkiin? Vastaus tähän on kyllä. Luokassa voi olla jäsenmuuttujien lisäksi myös jäsenfunktioita, joiden avulla rakenne joko muokkaa omia tietojaan tai käsittelee saamiansa parametreja. Jäsenfunktiot ovat olio-ohjelmoinnin keskeinen ominaispiirre, kun taas tietorakenteet eli tietojen paketointi yhden muuttujan taakse on tyypillinen piirre useimmissa ohjelmointikielissä. Esimerkiksi C-kielessä uusia tietorakenteita voi tehdä struct-rakenteella, mutta C-kielessä ei ole jäsenfunktioita. Koska tämä ei ole olio-ohjelmointiopas, keskitymme tässä oppaassa olioihin ja jäsenmuuttujiin tietorakenteina. Jäsenfunktioita käytämme tarpeen mukaan, mutta emme käsittele niiden tekemistä vaan se jää olio-ohjelmointikursseille.

Python on täysverinen olio-ohjelmointikieli, vaikka tällä kurssilla sitä käytetään proseduraaliseen ohjelmointiin. Aiheesta löytyy tarkempaa tietoa esimerkiksi Pythonin dokumenteista.

Luokka globaalina tunnuksena

Luokka helpottaa yhteenkuuluvien tietojen käsittelyä ja usein tiedot tallennetaan tiedostoon yhden olion tiedot aina yhdelle riville eroteltuina sopivalla erottimella (ks. edellinen luku). Näin ollen samaa tiedostoa luettaessa on luonnollista käyttää samaa luokkaa lähtökohtana

olioille, kun riveillä olleita tietoja käsitellään taas ohjelman sisällä. Ja kun samaa tietorakennetta käytetään useissa eri paikoissa ohjelmaa – tietojen kysyminen ja tiedoston luku sekä kirjoitus – kannattaa joka paikassa ohjelmaa käyttää samaa luokan määrittelyä. Tämän mahdollistamisiksi luokan määrittely sijoitetaan tyypillisesti ohjelmatiedoston alkuun päätasolle globaaliksi tunnukseksi, jotta se näkyy kaikkialle ohjelmassa. Huomaa, että luokka poikkeaa tähän asti käsittelemistämme tunnuksista, sillä se määrittelee tietotyypin ja sitä käytetään muuttujien/olioiden luomiseen. Siksi luokka määritellään globaalina tunnuksena ohjelman alussa, kun taas käsiteltävää tietoa sisältävät muuttujat määritellään paikallisina aliohjelmien ja pääohjelman sisällä.

Tässä oppaassa luokkien nimet kirjoitetaan isoilla kirjaimilla, esim. ”`class OPISKELIJA:`”. Tämä ei ole välttämätöntä, mutta helpottaa luokan ja olioiden/muuttujien erottamista toisistaan ja siten vähentää virheitä. Luokasta tehtävien olioiden nimet kannattaa nimetä normaalien muuttujien nimeämisohjeiden mukaisesti käyttäen luokan nimeä lähtökohtana mahdollisuuksien mukaan tyyliin `Opiskelija`.

Oliolista

Esimerkissä 7.5 käsittelimme kahta OPISKELIJA-luokan oliota ja koska opiskelijoita voi olla paljonkin, ei olion luominen jokaista opiskelijaa kohti ole järkevää pitkään. Aloitimme tämän luvun lista-rakenteen kanssa ja totesimme, että listaan voi lisätä mitä tahansa alkioita. Näin ollen lisäämällä listaan olioita, meillä on käytössä tarvittava määrä olioita ja niiden määrää voidaan muuttaa ohjelman aikana eli lisätä ja poistaa tarpeen mukaan.

Esimerkissä 7.6 näkyy oliolista luominen ja käyttö aiempia ideoita laajentamalla. Ohjelman alussa on määritelty luokka HENKILO, josta tehdään 3 oliota ja niiden jäsenmuuttujille annetaan sopivat arvot. Tämän jälkeen teemme listan, lisäämme oliot listaan ja käymme listan läpi for-rakenteella tulostaen alkioina olevien olioiden jäsenmuuttujien arvot. Listan perusominaisuuksia on alkioiden poistaminen ja oliolistan tapauksessa se onnistuu myös. Tällä kertaa joudumme käymään listaa läpi etsien alkiota, jonka jäsenmuuttujassa on etsitty arvo. Kannattaa huomata, että tällä kertaa oliolistan läpikäynti perustuu listan indekseihin ja kun etsitty alkio löytyy, voimme käytössä olevan indeksin arvolla poistaa listasta halutun alkion. Ohjelmassa on useita tulosteita, jotta käyttäjä pystyy seuraamaan ohjelman kulkua ja kuten aiemminkin, ohjelman lopussa lista tyhjennetään eli muisti vapautetaan.

Esimerkki 7.6 Oliolistan peruskäyttö

```
# Luokan eli tietotyypin määrittely
class HENKILO:
    Nimi = None
    Ika = None

# Muuttujien määrittely ja arvojen asetus, luokasta tehdään olioita
Henkilo1 = HENKILO()
Henkilo1.Nimi = "Jussi"
Henkilo1.Ika = 35
Henkilo2 = HENKILO()
Henkilo2.Nimi = "Annika"
Henkilo2.Ika = 25
Henkilo3 = HENKILO()
Henkilo3.Nimi = "Kari"
Henkilo3.Ika = 55
```

```
# Oliot lisätään listaan eli tehdään oliolista
Henkilosto = []
Henkilosto.append(Henkilo1)
Henkilosto.append(Henkilo2)
Henkilosto.append(Henkilo3)

print("Oliolistassa olevat tiedot, yksi olio eli henkilö rivillä:")
for Henkilo in Henkilosto:
    print(Henkilo.Nimi, Henkilo.Ika)
print()

PoistettavaNimi = "Kari"
Loytyi = False
print("Poistetaan listalta nimen '"+PoistettavaNimi+"' sisältävä olio.")
for i in range(len(Henkilosto)):
    if (Henkilosto[i].Nimi == PoistettavaNimi):
        del Henkilosto[i]
        Loytyi = True
        break

if (Loytyi == True):
    print("Poistettu listalta:", PoistettavaNimi)
else:
    print("Poistettavaa nimeä ei löytynyt.")
print()

print("Listassa on nyt seuraavat tiedot:")
for Henkilo in Henkilosto:
    print(Henkilo.Nimi, Henkilo.Ika)

# Oliolista tyhjennys
Henkilosto.clear()
```

Tuloste

```
Oliolistassa olevat tiedot, yksi olio eli henkilö rivillä:
Jussi 35
Annika 25
Kari 55

Poistetaan listalta nimen 'Kari' sisältävä olio.
Poistettu listalta: Kari

Listassa on nyt seuraavat tiedot:
Jussi 35
Annika 25
```

Rakenteisten tietotyyppien yleisiä ominaisuuksia

Lista ja tiedostot

Olemme käyneet listan perusoperaatiot läpi aiemmin, mutta esimerkissä 7.7 on uutena asiana listan käyttö tiedostojen kanssa eli ohjelmassa on pääohjelman lisäksi aliohjelma listan kirjoittamiseen tiedostoon sekä toinen tietojen lukemiseen tiedostosta listaan. Huomaa, että tässä tapauksessa tiedoston rivillä on vain 1 tietoalkio ja useita tietoalkioita sisältävän rivin tiedostonkäsittely käsitellään seuraavassa esimerkissä.

Esimerkin pääohjelman alussa määritellään nimiä sisältävä lista, jonka pohjalta voimme kutsua tiedoston kirjoittavaa aliohjelmaa, välittää sille parametrina tiedostonimen sekä tallennettavan listan sekä kirjoittaa listan tiedostoon aliohjelmassa. Tiedoston kirjoittamisen idea on käydä lista läpi ja kirjoittaa jokainen tietoalkio merkkijonona tiedostoon omalle rivilleen. Tiedostoa luettaessa aliohjelma saa parametrina taas tiedostonimen sekä listan, johon tiedot lisätään. Tämä lista tyhjennetään aliohjelman alussa, jotta mahdollisesti

aiemmin luetut tiedot eivät jäisi kummittelemaan listaan. Luonnollisesti tämä ei ole aina oikea toimintatapa, mutta se estää ylimääräisen datan kertymisen listaan ja siksi se on hyvä lähtökohta tiedoston lukemiseen. Mikäli jonkun ohjelman halutaan toimivan toisin, tulee asia näkyä sen kuvauksessa. Tiedoston lukeminen tehdään rivi kerrallaan ja tiedostosta luetut rivit lisätään listaan ilman rivin päättäviä rivinvaihtomerkkejä. Rivinvaihtomerkit lisättiin tietoalkioihin kirjoitusvaiheessa, sillä rivinvaihtomerkki toimi tietojen erotinmerkkinä tiedostossa. Siksi lukuvaiheessa ne pitää ottaa pois, jotta luettu tietoalkio vastaa kirjoitettua tietoalkiota. Koska listaa muutetaan aliohjelmassa, tulee se palauttaa paluuarvona kutsuvaan ohjelmaan ja sijoittaa siellä parametrina lähetetyn listan uudeksi arvoksi. Ja ohjelman lopuksi lista tyhjennetään.

Esimerkki 7.7 Listan käyttö tiedostojen ja aliohjelmien kanssa

```
def kirjoitaLista(Nimi, Tiedot):
    Tiedosto = open(Nimi, "w")
    for Tietoalkio in Tiedot:
       Tiedosto.write(Tietoalkio + "\n")
    Tiedosto.close()
    return None

def lueListaan(Nimi, Rivit):
    Rivit.clear()
    Tiedosto = open(Nimi, "r")
    Rivi = Tiedosto.readline()
    while (len(Rivi) > 0):
       Rivit.append(Rivi[:-1])
       Rivi = Tiedosto.readline()
    Tiedosto.close()
    return Rivit

def paaohjelma():
    Tiedostonimi = "L07E7.txt"
    Lista = ["Erkki", "Mia", "Jari", "Sanna-Katariina"]
    kirjoitaLista(Tiedostonimi, Lista)
    Lista = lueListaan(Tiedostonimi, Lista)
    for Alkio in Lista:
        print(Alkio, end=" ")
    print()
    Lista.clear()
    return None

paaohjelma()
```

Tuloste

```
Erkki Mia Jari Sanna-Katariina
```

Tiedostot: L07E7.txt

```
Erkki
Mia
Jari
Sanna-Katariina
```

Oliolista ja tiedostot

Tietoja sisältävissä tiedostoissa on tyypillisesti useita tietoalkioita samalla rivillä, joten esimerkin 7.7 ratkaisu ei ole aina riittävä. Kuten luokka/olio -keskustelun yhteydessä tuli ilmi, kirjoitetaan yhden olion sisältämät tiedot tyypillisesti yhdelle riville erottamalla tietoalkiot erotinmerkeillä. Näin ollen lukemalla tiedostosta yhden rivin, saamme yhteen olioon kuuluvat tietoalkiot käsiteltäviksi. Pilkkomalla rivi erotinmerkkien kohdista listaksi,

meillä on tietoalkiot merkkijonoina listassa ja muuttamalla merkkijonot takaisin alkuperäisiksi tietotyypeiksi, voimme sijoittaa ne olion jäsenmuuttujien arvoiksi eli palauttaa tekstitiedostoon tallennetut tiedot alkuperäiseen tietorakenteeseen. Tällä tavoin oliolista vastaa useista riveistä muodostuvaa tekstitiedostoa, jossa jokaisella rivillä on vastaavat tietoalkiot erotettuina erotinmerkeillä.

Esimerkin 7.8 alussa on määritelty tekstitiedoston kenttäerotin kiintoarvona ja sen jälkeen luokka, jonka jäsenmuuttujat nimeävät ohjelmassa käsiteltävät tiedot. Tämä ohjelma perustuu esimerkin 7.6 käskyihin, jotka on nyt ryhmitelty useisiin aliohjelmiin. Kaikki aliohjelmat ovat lyhyitä ja keskittyvät nimen mukaisiin toimintoihin, jotta niistä jokainen on itsessään helppo ymmärtää. Ohjelmassa on oleellista se, että ohjelman sisällä käytetään oliolistaa ja jäsenmuuttujia käsiteltävän datan tallentamiseen ja tiedostossa näitä vastaava rakenne on useat rivit sekä rivillä erotinmerkeillä erotetut tietoalkiot.

Esimerkki 7.8 Oliolistan käyttö tiedostojen ja aliohjelmien kanssa. Esimerkki 7.6 aliohjelmista muodostuvana ohjelmana

```
EROTIN = ";"

class HENKILO:
    Nimi = None
    Ika = None

def kysyTiedot(Tiedot):
    Henkilo = HENKILO()
    Henkilo.Nimi = input("Anna nimi: ")
    Henkilo.Ika = int(input("Anna ikä: "))
    Tiedot.append(Henkilo)
    print()
    return Tiedot

def tulostaTiedot(Tiedot):
    print("Henkilöstöstä on nyt seuraavat tiedot:")
    for Henkilo in Tiedot:
        print(Henkilo.Nimi, Henkilo.Ika)
    print()
    return None

def tallennaTiedot(Nimi, Tiedot):
    Tiedosto = open(Nimi, "w", encoding="UTF-8")
    for Henkilo in Tiedot:
        Tiedosto.write("{0}{1}{2}\n".format(Henkilo.Nimi, EROTIN, Henkilo.Ika))
    Tiedosto.close()
    return None

def lueTiedot(Nimi, Tiedot):
    # Tiedot.clear() # Kommenttimerkin takia tiedoston sisältö moninkertaistuu
    Tiedosto = open(Nimi, "r", encoding="UTF-8")
    Rivi = Tiedosto.readline()
    while (len(Rivi) > 0):
        Sarakkeet = Rivi.split(EROTIN)
        Henkilo = HENKILO()
        Henkilo.Nimi = Sarakkeet[0]
        Henkilo.Ika = int(Sarakkeet[1])
        Tiedot.append(Henkilo)
        Rivi = Tiedosto.readline()
    Tiedosto.close()
    return Tiedot
```

```
def paaohjelma():
    Tiedostonimi = "L07E8.txt"
    Henkilosto = []
    for i in range(3): # Kysytään useita tietoja listaan
        Henkilosto = kysyTiedot(Henkilosto)
    tulostaTiedot(Henkilosto)
    tallennaTiedot(Tiedostonimi, Henkilosto)
    Henkilosto = lueTiedot(Tiedostonimi, Henkilosto)
    tulostaTiedot(Henkilosto)
    Henkilosto.clear()
    return None

paaohjelma()
```

Tuloste

```
Anna nimi: Johanna
Anna ikä: 41

Anna nimi: Markku
Anna ikä: 45

Anna nimi: Noomi
Anna ikä: 23

Henkilöstöstä on nyt seuraavat tiedot:
Johanna 41
Markku 45
Noomi 23

Henkilöstöstä on nyt seuraavat tiedot:
Johanna 41
Markku 45
Noomi 23
Johanna 41
Markku 45
Noomi 23
```

Tiedosto L07E8.txt

```
Johanna;41
Markku;45
Noomi;23
```

Esimerkin 7.8 tiedonvälityksen kannattaa kiinnittää huomiota. Kaikkiin aliohjelmiin välitetään parametrinä oliolista ja jos sen sisältö muuttuu, palautetaan ko. lista kutsuvaan ohjelmana sekä asetetaan se parametrina lähteneen listan uudeksi arvoksi. Kuten aiemminkin, tiedostonkäsittelyä tekevät aliohjelmat saavat ensimmäisenä parametrina tiedostonimen, jonka perusteella ne avaavat ja sulkevat tiedoston aliohjelman sisällä.

Listan analysointi, tulos-olio ja virheiden ennaltaehkäisy

Esimerkissä 7.8 näimme ohjelman sisäisen oliolistan ja sitä vastaavan tekstitiedoston toteutuksen, mikä mahdollistaa tekstitiedostoja käsitteleviä ohjelmien toteuttamisen oliolistaan perustuen. Toinen yleinen ohjelmatyyppi analysoi luettua dataa ja selvittää siitä erilaisia tietoja seuraavan esimerkin mukaisesti.

Esimerkin 7.9 ohjelma lukee tiedostosta dataa, sijoittaa tiedot oliolistaan, analysoi tiedot ja tulostaa selvitetyt tiedot. Ohjelmassa on oliolistassa käytettävän HENKILO-luokan lisäksi TULOS-luokka, jossa on useita jäsenmuuttujia selvittäviä tietoja varten. Näin ollen analyysi-aliohjelmalle lähetetään parametreina analysoitava oliolista sekä tulos-olio ja aliohjelman tehtäväksi jää tulos-olion tietojen selvittäminen sekä palauttaminen kutsuvaan

ohjelmaan paluuarvona. Pääohjelma ottaa tulos-olion talteen ja lähettää sen tällä kertaa tulostavaan aliohjelmaan, jolle toinen luonnollinen vaihtoehto olisi tiedostoon kirjoittava aliohjelma. Oleellista tässä esimerkissä on tulos-olion käyttö, jolla useita tietoja saadaan palautettua analyysi-aliohjelmasta kutsuvaan ohjelmaan ja ne on helppo välittää sieltä eteenpäin muihin aliohjelmiin yhdessä oliossa.

Esimerkin pääohjelma on minimaalinen, koska se keskittyy analyysikonsepteihin eikä esim. valikko-rakenteita ole mukana pidentämässä ohjelmaa. Huomaa, että pääohjelmassa analysoi-aliohjelman kutsu on laitettu ehdolliseksi eli ennen sitä varmistetaan, että listassa on alkioita analysoitavaksi. Palaamme tähän konseptiin myöhemmin virheenkäsittelyn yhteydessä, sillä virheiden ennaltaehkäisy on varteenotettava vaihtoehto virheenkäsittelylle.

Esimerkki 7.9 Oliolistan analyysi tulos-oliolla

```
EROTIN = ";"

class HENKILO:
    Nimi = None
    Ika = None

class TULOS:
    Nuorin = None
    Nimi = None
    KeskiIka = None

def lueTiedosto(Nimi, Tiedot):
    Tiedot.clear()
    Tiedosto = open(Nimi, "r", encoding="UTF-8")
    Rivi = Tiedosto.readline()
    while (len(Rivi) > 0):
        Sarakkeet = Rivi.split(EROTIN)
        Henkilo = HENKILO()
        Henkilo.Nimi = Sarakkeet[0]
        Henkilo.Ika = int(Sarakkeet[1])
        Tiedot.append(Henkilo)
        Rivi = Tiedosto.readline()
    Tiedosto.close()
    return Tiedot

def analysoi(Tiedot, Tulokset):
    Lukumaara = 0
    Summa = 0
    NuorinIka = None
    NuorinNimi = None

    for Henkilo in Tiedot:
        if ((NuorinIka == None) or (NuorinIka > Henkilo.Ika)):
            NuorinIka = Henkilo.Ika
            NuorinNimi = Henkilo.Nimi
        Lukumaara = Lukumaara + 1
        Summa = Summa + Henkilo.Ika

    Tulokset.Nuorin = NuorinIka
    Tulokset.Nimi = NuorinNimi
    Tulokset.KeskiIka = Summa / Lukumaara
    return Tulokset

def tulostaTulokset(Tulokset):
    print("Nuorin henkilö on {0:2d} vuotta ja nimeltään {1:s}.".
            format(Tulokset.Nuorin, Tulokset.Nimi))
    print("Keski-ikä on {0:.1f} vuotta.".format(Tulokset.KeskiIka))
    return None
```

```
def paaohjelma():
    Tiedostonimi = "L07E9.txt"
    Henkilosto = []
    Tulos = TULOS()
    Henkilosto = lueTiedosto(Tiedostonimi, Henkilosto)
    if (len(Henkilosto) > 0):
        Tulos = analysoi(Henkilosto, Tulos)
    else:
        print("Ei tietoja analysoitavaksi.")
    tulostaTulokset(Tulos)
    Henkilosto.clear()
    return None

paaohjelma()
```

Tuloste

```
Nuorin henkilö on 25 vuotta ja nimeltään Annika.
Keski-ikä on 38.3 vuotta.
```

Tiedosto L07E9.txt

```
Jussi;35
Annika;25
Kari;55
```

Tiedonvälitys eli parametrit ja paluuarvo

Aliohjelmien yhteydessä tutustuimme tiedonvälitykseen pää- ja aliohjelmien välillä, jossa lähtökohtana on välittää tietoa aliohjelmiin parametreilla ja palauttaa niistä tietoa kutsuvaan ohjelmaan paluuarvona. Kuten luvussa 5 totesimme, ovat merkkijono, kokonaisluku ja desimaaliluku arvoparametreja eikä aliohjelmassa tehdyt muutokset näy niissä. Mainitsimme jo tällöin, että tilanne muuttuu jatkossa ja nyt rakenteisten tietotyyppien kohdalla tilanne muuttuu, sillä ne ovat muuttujaparametreja ja parametreihin tehdyt muutokset näkyvät myös kutsuvassa ohjelmassa.

Arvo- ja muuttujaparametrien teknisten yksityiskohtien käsittely ei kuulu tämän oppaan laajuuteen, joten tässä oppaassa muuttujaparametreja käsitellään samalla tavoin kuin arvoparametreja. Käytännössä tämä tarkoittaa sitä, että tieto välitetään aliohjelmiin parametreilla ja jos näiden parametrien tietoja muutetaan aliohjelmassa, tulee ne palauttaa takaisin kutsuvaan ohjelmaan ja sijoittaa parametrina lähetetyn muuttujan uudeksi arvoksi sijoituslauseella. Kaikkien parametrien käsittely samalla tavalla tekee ohjelmista ymmärrettävämpiä sekä vähentää virheiden määrää.

Aiemmin oppaassa oli myös puhetta, että aliohjelmista palautetaan vain yksi paluuarvo. Rakenteisten tietotyyppien myötä aliohjelmista voi palauttaa useita arvoja yhden muuttujan sisällä, joten tämän jälkeen useille paluuarvoille ei ole tarvetta eikä niitäkään käytetä virheiden vähentämiseksi.

Kasvavien ohjelmien ominaispiirteitä

Kasvava valikkopohjainen ohjelma

Tässä oppaassa esitellään Pythonin keskeisiä käskyjä ja tietorakenteita, jotka voi tyypillisesti esittää tiiviisti muutamalla rivillä. Tämä on ollut Python-ohjelmointikielen keskeisiä suunnittelutavoitteita ja Pythonia kehitetään edelleen samaan suuntaan lisäämällä siihen pieniä hyvin toimivia ominaisuuksia, joita pystyy käyttämään tyypillisesti muutamalla rivillä koodia.

Pythonin tiiviistä toteutuksesta huolimatta ohjelmat tahtovat kasvaa eli pidentyä ja niihin tulee useita toiminnallisuuksia, aliohjelmia jne. Ohjelmien kasvaessa niiden rakenteeseen

pitää kiinnittää aiempaa enemmän huomioita, jotta ohjelman rakenne ja toiminta olisi ymmärrettävää ja virheet olisi helppo huomata sekä korjata ennen kuin ne aiheuttavat isompia ongelmia. Siksi tyypillisesti hyödylliset ohjelmat kasvavat ja niistä tulee jossain vaiheessa isoja eli pitkiä.

Tässä oppaassa kasvavien ohjelmien rakenteen pyritään pitämään hallinnassa valikkopohjaisen ohjelman avulla. Valikko tarjoaa selkeän ja yksinkertaisen käyttöliittymän, josta käyttäjä voi valita haluamansa toiminnon – ja jos sellaista ei ole, voi valikkoon lisätä helpolla uuden valinnan ja ohjelman rakenne tarjoaa selkeän tavan laajentaa ohjelmia sisäisesti aliohjelmina. Pääohjelman valintarakenne keskittyy käyttöliittymän pyörittämiseen ja siitä käyttäjän haluamien toimintojen kutsumiseen. Tässä luvussa näkyy jo hyvin, miten valikkopohjainen ohjelma kasvaa käytännössä lyhyiden aliohjelmien kautta isommaksi, mutta silti hallittavaksi kokonaisuudeksi.

Ohjelmat jatkavat kasvuaan myös oppaan myöhemmissä luvuissa. Keskeisimmät seuraavat asiat ovat yhä kasvavien ohjelmien koon hallinta, uudelleenkäyttö sekä virheenkäsittely. Esimerkiksi uudelleenkäytöllä pyritään vähentämään tarvittavan koodin määrä ja toisaalta virheenkäsittelyllä pyritään estämään ohjelman kaatuminen ja muut vakavat ongelmat. Erityisesti virheenkäsittely kasvattaa ohjelmia, sillä perusidea on tunnistaa ongelmat eli erikoistapaukset ja kirjoittaa niitä varten koodia, jolla pyritään tyypillisesti ehkäisemään ongelmia ennalta tai minimoimaan poikkeustilanteiden ikävät seuraukset.

Tiedoston alkukommentti

Ohjelmien kasvaessa ja monipuolistuessa niiden käyttömahdollisuudet laajenevat, käyttöikä pitenee ja käyttäjäkunta voi kasvaa. Siksi usein herää kysymys, kuka ohjelman on tehnyt, milloin ja mitä tarkoitusta varten. Toinen tyypillinen kysymys liittyy ohjelmiin, joita on käytetty jo tovi ja niihin on tehty muutoksia, jolloin luonnollinen kysymys on muutoshistoria eli kuka on tehnyt ohjelmaan muutoksia, milloin ja mitä. Koska ohjelmat kirjoitetaan tekstitiedostoihin, voi ohjelman alkuun laittaa kommenttina vastauksia näihin sekä muihin vastaaviin kysymyksiin.

Kuvassa 7.1 näkyy ohjelmoinnin peruskurssille sovitettu alkukommentti kurssilla tehtäviin ohjelmiin. Kommenttikentän tiedot tulee aina sovittaa siihen ympäristöön, missä ohjelmia tehdään ja tässä tapauksessa alla näkyvät tiedot on katsottu sopiviksi. Jokainen yritys ja projekti voi määritellä itse omiin tilanteisiin sopivat kommentit, jotka löytävät tilanteeseen sopivan muodon aina tarpeen mukaan. Kommenttien tehtävä on siis ratkaista joku ongelma ja esim. ohjelman tekijän nimi mahdollistaa asian kysymisen tekijältä ja muutoshistoria taas auttaa ymmärtämään, miksei esim. vanha tiedosto aukea enää ohjelman uudella versiolla jne.

Kuva 7.1. Tiedoston alkukommentti

```
#########################################################################
# CT60A0203 Ohjelmoinnin perusteet
# Tekijä:
# Opiskelijanumero:
# Päivämäärä:
# Kurssin oppimateriaalien lisäksi työhön ovat vaikuttaneet seuraavat
# lähteet ja henkilöt, ja se näkyy tehtävässä seuraavalla tavalla:
#
# Mahdollisen vilppiselvityksen varalta vakuutan, että olen tehnyt itse
# tämän tehtävän ja vain yllä mainitut henkilöt sekä lähteet ovat
# vaikuttaneet siihen yllä mainituilla tavoilla.
#########################################################################
# Tehtävä LxTx.py

# eof
```

Yhdessä tiedostossa olevan ohjelman tiedostorakenne

Tässä luvussa tutustuimme uuden tietotyypin luomiseen eli luokkaan ja totesimme, että sen pitäisi olla käytettävissä kaikkialla ohjelmassa. Lisäksi totesimme, että ohjelmissa pitäisi olla systemaattinen alkukommentti, joten ohjelmien tiedostorakenne muuttuu näiden havaintojen myötä ja näyttää nyt seuraavalta:

1. Alkukommentit
2. Kiintoarvojen määrittelyt
3. Luokkien määrittelyt
4. Aliohjelmat
5. Pääohjelma
6. Pääohjelmakutsu

Kaikissa ohjelmissa ei ole kaikkia näitä elementtejä, mutta ohjelmassa olevat elementit tulee laittaa yllä olevaan järjestykseen.

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.

- Lista
    - Peruskäyttö (E7.1)
    - Perusoperaatiot ja jäsenfunktiot (E7.2)
    - Matriisi eli 2-ulotteinen tietorakenne
    - Merkkijonon jako listaksi split:llä erotin-merkeistä (E7.3)
    - Muita sekvenssitietotyyppejä: tuple (E7.4)
- Luokka/olio
    - Peruskäyttö (E7.5)
    - Luokan jäsenfunktioiden käyttö vs. toteutus ja olio-ohjelmointi
    - Luokka globaalina tunnuksena
- Oliolista
    - Peruskäyttö, alkion poistaminen (E7.6)
- Rakenteisten tietotyyppien yleisiä ominaisuuksia
    - Listan kirjoittaminen tiedostoon ja tiedoston luku listaan (E7.7)
    - Oliolistan luku ja kirjoitus, tiedonvälitys aliohjelmien kanssa (E7.8)
    - Analysoiva aliohjelma, virheiden ennaltaehkäisy ja tulos-olio (E7.9)
    - Tiedonvälitys aliohjelmiin ja takaisin, arvo- ja muuttujaparametrit
- Kasvavien ohjelmien ominaispiirteitä
    - Tiedoston alkukommentti
    - Yhdessä tiedostossa olevan ohjelman tiedostorakenne
    - Kasvava valikkopohjainen ohjelma (E7.8, 7.9, 7.10)

Pienen Python-perusohjelman tyyliohjeet

Tähän lukuun liittyvät tyyliohjeet on koottu alle. Alla olevia yleisohjeita noudattamalla vältät yleisimmät rakenteisiin tietorakenteisiin liittyvät ongelmat ja perustellusta syystä niistä voi ja tulee poiketa.

Listaan liittyviä ohjeita

1. Dynaamisia rakenteita kuten listaa käytetään, jos samanlaisia muuttujia on 5 tai enemmän
2. Listan kaikkien alkioiden tulee olla samaa tietotyyppiä
3. Tiedostoa luettaessa listaan, se tyhjennetään ennen alkioiden lisäämistä
4. Tiedostossa yhdellä rivillä olevista tiedoista muodostetaan yksi olio, joka tallennetaan oliolistaan
5. Listan läpikäyntiin käytetään `for Alkio in Lista` –rakennetta, paitsi jos indeksin käyttö tuo lisäarvoa
6. Listan alkiot tulostetaan erillisinä tietoalkioina. Koko listan tulostus yhdellä kertaa tarkoittaa keskeneräistä ohjelmaa
7. Lista tyhjennetään sen pää/aliohjelman lopussa, jossa se luodaan
8. Tässä oppaassa ei käytetä tuplea ja sitä käsitellään sen verran, että vahingossa luotu tuple on helpompi huomata ja korjata listaksi
9. Rakenteisten tietorakenteiden vertailussa käytetään == operaattoria. Operaattorit is/is not liittyvät olioiden identiteetin tarkastukseen eikä niitä käytetä tässä oppaassa

Luokkiin ja olioihin liittyviä ohjeita

10. Luokat määritellään tiedoston alussa globaaleina rakenteina
11. Luokan nimi kirjoitetaan suuraakkosilla, esim. `AUTO`, `DATA`
12. Luokassa tulee olla vähintään kaksi jäsenmuuttujaa, jotka ovat perustietotyyppejä eli merkkijono, kokonaisluku tai desimaaliluku, ne vastaavat käytettävän tiedon tietotyyppiä (esim. lukumäärä on kokonaisluku eikä merkkijono) ja ne alustetaan `None`:lla
13. Luokassa ei saa olla jäsenmuuttujana rakenteisia tietorakenteita, esim. listaa
14. Luokkaa käytetään vain olioiden luomiseen
15. Olio nimetään luokan perusteella muuttujien nimeämisohjeilla, esim. `Auto`, `Data`, `Auto1`, `Auto2`
16. Olioita luotaessa käytetään kaarisulkuja (ja), esim. `Data = DATA()`
17. Olioihin ei saa lisätä jäsenmuuttujia ohjelman suorituksen aikana
18. Oliolista tarkoittaa listaa, jonka kaikki alkiot ovat olioita
19. Tässä oppaassa ei tehdä jäsenfunktioita, sillä tämä ei ole olio-ohjelmointiopas. Samasta syystä init-metodia ei käytetä tässä oppaassa

Aliohjelmiin, tiedonvälitykseen ja virheiden ennaltaehkäisyyn liittyviä ohjeita

20. Valikkopohjaisen ohjelman pääohjelman valintarakenteessa tarkistetaan ennen aliohjelmakutsuja, että parametrina käytettävissä dynaamisissa rakenteissa on tietoalkioita. Jos näin ei ole, ei aliohjelmaa kutsuta vaan ongelma kerrotaan käyttäjälle
21. Aliohjelmiin viedään tietoa parametreilla ja tietoa palautetaan paluuarvolla, joka sijoitetaan kutsuvassa ohjelmassa muuttujaan. Muuttujaparametreja ei käytetä
22. Aliohjelmasta palautetaan vain yksi paluuarvo. Mikäli palautettavia arvoja on useampia, tulee käyttää sopivaa rakenteista tietorakennetta

Tietojen analysointiin liittyviä ohjeita

23. Analyysi-aliohjelmaan lähetetään analysoitava data ja tulostietorakenne. Analysoitava data on tyypillisesti listassa ja tulostietorakenne on tyypillisesti olio tai lista
24. Aiemman analyysin tulokset poistetaan ennen uusien tulosten laskentaa
25. Etsittäessä datasta tiettyjä arvoja kuten minimi tai maksimi, tulee tilapäismuuttujat alustaa datasetin ensimmäisen alkion arvoilla

26. Mikäli datassa on useita ehdon täyttäviä arvoja, esim. useita yhtä suuria minimi-/maksimiarvoja, valitaan näistä
    a. Erikseen mainitun ehdon täyttävä arvo, jos tällainen ehto on olemassa
    b. Vanhin, jos datassa on aikaleima
    c. Alkuperäisen datan ensimmäinen arvo (rivinumeron mukaan)
27. Mikäli tehtävässä käytetään datasetin ominaisuuksia, tulee ne selvittää datasta, esim. alkioiden lukumäärä tai ensimmäisen/viimeisen alkion aikaleima
28. Kaikki analyysit tulee suorittaa alkuperäisissä yksiköissä ja mahdollinen tulosten pyöristys tehdään muotoiltaessa lopullisia tulosteita

Luvun asiat kokoava esimerkki

Luvun kokoava esimerkki 7.10. kokoaa yhteen tähän asti tässä oppaassa läpikäydyt asiat. Ohjelman käyttöliittymä näkyy `valikko()`-aliohjelmassa, ja tyypillisesti valikon valinnat käyvät ilmi kirjoitettavan ohjelman toimeksiannosta. Valikon rakenne heijastuu pääohjelman valintarakenteeseen ja tyypillisesti jokaiseen valikon kohtaan liittyvä toiminnallisuus toteutetaan omana aliohjelmanaan. Tässä ohjelmassa on tiedostonkäsittelyyn liittyvät `tallenna()` ja `lue()` aliohjelmat, käyttäjältä tiedot kysyvä `kysy()`-aliohjelma ja tiedot tulostava `tulosta()`-aliohjelma. Tiedot näiden kaikkien aliohjelmien välillä välitetään lista-parametrilla ja paluuarvolla. Koska useissa aliohjelmissa käytetään samaa luokkaa ja tiedoston kenttäerotinta, on ne määritelty globaaleina kiintoarvoina päätasolla ohjelman alussa. Globaalien kiintoarvojen kohdalla kannattaa muistaa, etteivät ne muutu ohjelman suorituksen aikana vaan muuttuvia tietoja sisältävät muuttujat ovat paikallisia muuttujia aliohjelmissa ja tieto kulkee aliohjelmien välillä parametreilla sekä paluuarvoilla.

Esimerkki 7.10. Kokoava esimerkki oliolistalla

```
EROTIN = ';'

class HENKILO:
    Nimi = None
    Pituus = None

def valikko():
    print("Mitä haluat tehdä:")
    print("1) Kysy tiedot")
    print("2) Tallenna tiedosto")
    print("3) Lue tiedosto")
    print("4) Tulosta tiedot")
    print("0) Lopeta")
    Valinta = int(input("Valintasi: "))
    return Valinta

def kysy(Lista):
    Henkilo = HENKILO()
    Henkilo.Nimi = input("Anna nimi: ")
    Henkilo.Pituus = int(input("Anna pituus: "))
    Lista.append(Henkilo)
    return Lista

def tallenna(Nimi, Lista):
    Tiedosto = open(Nimi, "w", encoding="utf-8")
    for Henkilo in Lista:
        Tiedosto.write("{0:s}{1:s}{2:d}\n".format(
                        Henkilo.Nimi, EROTIN, Henkilo.Pituus))
    Tiedosto.close()
    return None
```

```
def lue(Nimi, Lista):
    Lista.clear()
    Tiedosto = open(Nimi, "r", encoding="utf-8")
    Rivi = Tiedosto.readline()
    while (len(Rivi) > 0):
        Rivi = Rivi[:-1] # rivinvaihtomerkki pois
        Sarake = Rivi.split(EROTIN)
        Henkilo = HENKILO()
        Henkilo.Nimi = Sarake[0]
        Henkilo.Pituus = int(Sarake[1])
        Lista.append(Henkilo)
        Rivi = Tiedosto.readline()
    Tiedosto.close()
    return Lista

def tulosta(Lista):
    for Henkilo in Lista:
        print("Henkilö {0:s} on {1:d} cm pitkä.".format(
                  Henkilo.Nimi, Henkilo.Pituus))
    return None

def paaohjelma():
    Valinta = 1
    TiedostoNimi = "L07E10.txt"
    Tiedot = []
    while (Valinta != 0):
        Valinta = valikko()
        if (Valinta == 1):
            Tiedot = kysy(Tiedot)
        elif (Valinta == 2):
            if (len(Tiedot) > 0):
                tallenna(TiedostoNimi, Tiedot)
            else:
                print("Ei tietoja tallennettavaksi.")
        elif (Valinta == 3):
            Tiedot = lue(TiedostoNimi, Tiedot)
        elif (Valinta == 4):
            tulosta(Tiedot)
        elif (Valinta == 0):
            print("Lopetetaan.")
        else:
            print("Tuntematon valinta, yritä uudestaan.")
        print()
    Tiedot.clear()
    print("Kiitos ohjelman käytöstä.")
    return None

paaohjelma()
```

Tuloste

```
Mitä haluat tehdä:
1) Kysy tiedot
2) Tallenna tiedosto
3) Lue tiedosto
4) Tulosta tiedot
0) Lopeta
Valintasi: 1
Anna nimi: Leena
Anna pituus: 168

Mitä haluat tehdä:
1) Kysy tiedot
```

```
2) Tallenna tiedosto
3) Lue tiedosto
4) Tulosta tiedot
0) Lopeta
Valintasi: 1
Anna nimi: Matti
Anna pituus: 189

Mitä haluat tehdä:
1) Kysy tiedot
2) Tallenna tiedosto
3) Lue tiedosto
4) Tulosta tiedot
0) Lopeta
Valintasi: 2

Mitä haluat tehdä:
1) Kysy tiedot
2) Tallenna tiedosto
3) Lue tiedosto
4) Tulosta tiedot
0) Lopeta
Valintasi: 3

Mitä haluat tehdä:
1) Kysy tiedot
2) Tallenna tiedosto
3) Lue tiedosto
4) Tulosta tiedot
0) Lopeta
Valintasi: 4
Henkilö Leena on 168 cm pitkä.
Henkilö Matti on 189 cm pitkä.


Mitä haluat tehdä:
1) Kysy tiedot
2) Tallenna tiedosto
3) Lue tiedosto
4) Tulosta tiedot
0) Lopeta
Valintasi: 0
Lopetetaan.

Kiitos ohjelman käytöstä.
```

Tiedosto L07E10.txt

```
Leena;168
Matti;189
```