CT10A0013 Ohjelmointi Pythonilla

L06: Tiedoston käsittely

Uolevi Nikula

Päivän asiat

- Teoria
  - Muuttujat, kovakoodaus ja muuttujien roolit
- Käytäntö
  - Tiedostot: taustaa, kirjoittaminen, lukeminen
  - Merkkijonot ja jäsenfunktiot
  - Merkkijonojen yhteenveto ja muotoiltu tulostus format:lla
- Koodiesimerkkejä
- Lopuksi

Teoria

Muuttujat
Kovakoodaus
Muuttujien roolit

Muuttujat ja ohjelman tila

- Ohjelmoinnissa on kolme peruskontrollirakennetta
  - Peräkkäinen suoritus
  - Haarautuminen
  - Toisto
- Nämä mahdollistavat ohjelmien tekemisen ja ohjelmien kulun ymmärtämisen sekä ohjaamisen
- Ohjelman varsinaisen kulun ymmärtäminen ja selvittäminen edellyttää tietoa **ohjelman tilasta**
  - Ohjelman tila tallennetaan normaalisti muuttujiin

Muuttuja vs. “kovakoodaus”

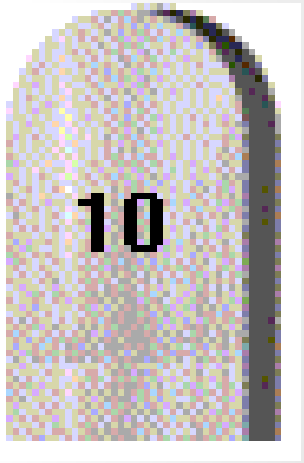
- Tietokone-ohjelmat tehdään joustaviksi, jotta ne pystyvät ratkaisemaan samanlaisia ongelmia eri lähtöarvoilla
- Joustavuus saadaan mm. kysymällä haluttu tieto käyttäjältä, tallettamalla se muuttujaan ja käyttämällä ohjelmassa muuttujaa aina kun ko. tietoa tarvitaan
- Etenkin kehitysvaiheessa tietoa ei aina kysytä käyttäjältä vaan muuttujalle annetaan arvo suoraan koodissa
  - Tämä on “**kovakoodausta**”: tieto ei tule käyttäjältä tms. vaan käytössä on kiintoarvo muuttujan sijasta
- Ohjelmaa testataan usein yhdellä tiedostolla, jolloin tulee houkutus kovakoodata esim.
  - Tiedoston nimi
  - Tiedostossa olevat tiedot, erityisesti ensimmäinen ja viimeinen alkio
  - Tiedostossa olevien tietojen lukumäärä – esim. vuodessa on 365 tai kuukaudessa 31 päivää tms.
- Kovakoodaus nopeuttaa kehitysvaiheessa, muttei ole järkevää lopullisessa ohjelmassa
  - Kokenut ohjelmoija tunnistaa tiedot, joita ei kannata kovakoodata
  - Useiden testitiedoston käyttö estää kovakoodauksen
  - Viikkotehtävissä ja harjoitustyössä on usein mainittu, mikäli asioita voidaan yksinkertaistaa kovakoodaamalla

Muuttujien roolit

- Muuttujilla on keskeinen asema ohjelmoinnissa, mm. ohjelman tiedon varastointi ja suorituksen ohjaaminen
  - Muuttujilla on erilaisia rooleja
- Kolme eniten käytettyä muuttujien roolia ovat kiintoarvo, askeltaja ja tuoreimman säilyttäjä
  - Nämä roolit kattavat noin 70 % kaikkien muuttujien rooleista
- Seuraavassa käydään kootusti läpi tämän kurssin kannalta keskeisimmät muuttujien roolit

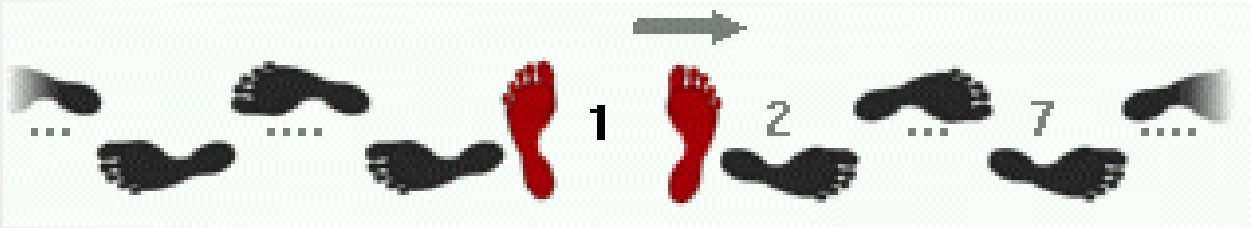
Kiintoarvo

- Muuttuja, jonka arvoa ei muuteta sen asettamisen jälkeen, on *kiintoarvo*
- Keskeisimmät syyt kiintoarvojen käyttöön ovat
  - Ohjelman selventäminen eli ymmärrettävyys
  - Muutosten tekemisen helpottaminen
- Käyttö ei yleensä ole pakollista
- Tyypillisiä esimerkkejä ovat luonnonvakiot (pii) ja muuntokertoimet (tuuma <-> sentti)



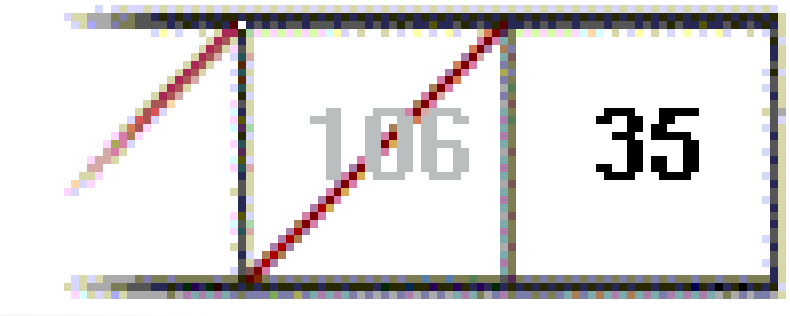
Askeltaja

- *Askeltaja* muuttuja käy läpi arvoja jollain systemaattisella tavalla
- Yleisimmin askeltaja esiintyy toisto-rakenteissa
  - for:ssa aina
  - while:ssa joskus



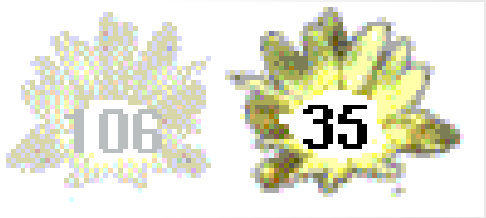
Tuoreimman säilyttäjä

- *Tuoreimman säilyttäjä* –muuttuja sisältää viimeisimmäksi käsitellyn arvon. Tämä voi olla esim.
  - yksi alkio joukosta
  - viimeisimmäksi saatu arvo (esim. käyttäjältä silmukassa)
  - tyypillisesti while-rakenteessa askeltajan sijasta



Sopivimman säilyttäjä

- *Sopivimman säilyttäjän* arvo on "paras" tai jollain muulla tavoin halutuin siihen asti läpikäydyistä arvoista
- Arvojen paremmuuden mittaamisessa ei ole mitään rajoituksia: sopivin voi tarkoittaa esimerkiksi
  - Pienin luku
  - Suurin luku
  - Luku, joka on lähinnä jotain tiettyä arvoa



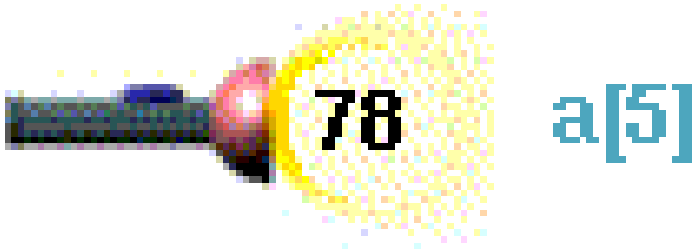
Kokooja

- *Kokoojan* arvo kerääntyy kaikista siihen mennessä läpikäydyistä arvoista
- Esimerkiksi keskiarvon laskennassa käytettiin kokooja-muuttujaa *summa*
  - Tässä tapauksessa kokoojaan *summa* lasketaan yhteen kaikkien läpikäytyjen arvojen summa (josta sitten lasketaan jotain muuta)
- Kokoojaa ei välttämättä ole tarvinnut missään ennen ohjelmointia
  - Muuttujan, kokoojan, *alustus nollaksi*
  - Muuttujan *kasvattaminen* aina uudella (tuoreimmalla) arvolla
- Esimerkki pseudokoodina

```
Summa = 0
while (…):
    Summa = Summa + UusiArvo
```

Tilapäissäilö

- Muuttuja on tilapäissäilö, jos sen arvoa tarvitaan aina vain hyvin lyhyen ajan. Tyypillisiä käyttötarkoituksia ovat
  - Ohjelman tehostaminen: usein tarvittavan laskutoimituksen tulos talletetaan muuttujaan turhan laskennan välttämiseksi
  - Ohjelman selventäminen: lasketaan tulos omaan muuttujaansa vaikkei tämä ole välttämättä tarpeen



Muut roolit ja lisämateriaalia

- Muuttujilla on myös muita rooleja
  - Lähes kaikki yksinkertaisissa ohjelmissa käytettävät muuttujat kuuluvat 11 perusroolin joukkoon
  - Edellä käsiteltyjen 6 roolin lisäksi muut ovat **seuraaja**, **yksisuuntainen lippu**, järjestelijä, säiliö ja kulkija
- Roolien esittely löytyy Internetistä

  http://saja.kapsi.fi/var_roles/stud_vers/stud_Python3_fin.html
- Ohjelmien toimintaa ja muuttujien rooleja on havainnollistettu PlanAni ohjelmalla
  - Ohjelman voi ladata Internetistä, asentaa ja testailla
  - Ohjelma löytyy alla olevasta osoitteesta

  http://saja.kapsi.fi/var_roles/planani/index.html

Käytäntö

Tietokoneet ja tiedostonkäsittely
Keskusmuisti ja oheismuisti
Tiedostoista ja niiden käsittelystä
Data-alkioita tiedostossa riveillä
Merkkijonot ja jäsenfunktiot, format

Tietokoneet ja tiedostonkäsittely

- Tähän asti ohjelmat ovat kysyneet tiedot ohjelman suorituksen aikana
  - Mitään tietoa ei ole luettu tiedostosta tai kirjoitettu tiedostoon
- Käytännössä kaikki ”oikeat” ohjelmat käyttävät tiedostoja, esim.
  - Tekstinkäsittely: Työstettävä dokumentti talletetaan kovalevylle, sitä laajennetaan ja muokataan jne. Esim. diplomityötä työstetään n. 6kk
  - Taulukkolaskenta: Numerot laitetaan soluihin, lisätään laskentaan, muokataan tietoja ja lisätään laskentaa jne.
  - Valokuvat: Kamera tallentaa valokuvat muistiin ja niitä voidaan muokata, katsella ja jakaa sen jälkeen aina niin haluttaessa
- Tämän luennon aiheena on tiedoston kirjoittaminen ja lukeminen
  - Keskitymme yksinkertaisimpaan tiedostomuotoon eli tekstitiedostoon

Keskusmuisti ja oheismuisti

- Ohjelmien käsittelemä **tieto** voi olla keskusmuistissa tai oheismuistilla
  - Nopeassa keskusmuistissa olevia tietoja käsitellään muuttujien avulla
  - Oheismuistilla olevat tiedot ovat siirretty hitaammalle muistialueelle yleensä kahdesta syystä
    - Ne odottavat käyttöä myöhemmin (pysyvä tallennus; normaali syy)
    - Ne eivät mahdu keskusmuistiin ohjelman suorituksen aikana (poikkeus)
  - Oheismuistilla oleva **tietovarasto** on yleensä teksti- tai binaaritiedosto tai tietokanta
    - Tällä kurssilla keskitytään tekstitiedostoihin
- **Suoritettava ohjelma** on suorituksen aikana tietokoneen keskusmuistissa (RAM) ja muutoin oheismuistilla eli massamuistilla (.py tiedostona)

Tietoa sisältävät tiedostot

- Tekstitiedostoja voidaan käyttää eri tarkoituksiin. Ohjelmoinnin kannalta ne voivat sisältää
  - **Ohjelmia**, esim. Python-ohjelmia, nämä ovat *lähdekooditiedostoja*
  - **Dataa**, esim. nimiä ja osoitteita, nämä ovat *tieto- tai datatiedostoja*
  - Kaikki tekstitiedostot muodostuvat kirjanmerkeistä eli merkkijonoista
- Tällä kurssilla keskitytään tiedon tallettamiseen tekstitiedostoihin
  - Tekstitiedosto kannattaa avata koodieditorilla, esim. IDLE
  - On mahdollista käsitellä myös binaaritiedostoja, palataan myöhemmin
- Tiedostoja voi käsitellä eri tavoin
  - Luku (read eli r)
  - Kirjoitus (write eli w)
  - Lisäys eli olemassa olevan tiedoston perään kirjoittaminen (append eli a)
- Tiedosto tulee avata ennen käyttöä ja sulkea käytön jälkeen
  - Tiedoston avaamisen epäonnistuminen on yleistä, joten siihen tulee varautua
  - Virheenkäsittelyyn palataan luennolla 9

Tiedoston lukeminen ja kirjoittaminen

- Tiedoston lukeminen noudattaa samaa kaavaa kuin vihon lukeminen
  - Avaa (tiedosto/vihko) – open(…)
  - Lue merkkejä (kunnes olet valmis/lopussa) – readline(…)
  - Sulje (tiedosto/vihko) – close()
- Tiedoston kirjoittaminen noudattaa samaa kaavaa kuin vihon kirjoittaminen
  - Avaa (tiedosto/vihko) – open(…)
  - Kirjoita merkkejä (kunnes olet valmis) – write(…)
  - Sulje (tiedosto/vihko) – close()

Huomioita tiedostonkäsittelystä

- Usein tiedostojen avaamisen yhteydessä määritellään käytetty koodaus

```
tiedosto = open("data.txt", "r", encoding="utf-8")
```

  - Erikoismerkkejä sisältävä tekstitiedosto kannattaa koodata UTF-8:lla (ööå)
- Tiedoston luku kannattaa tehdä `readline()`-funktiolla. Funktiot `readlines()` ja `read()` käyvät myös, mutta toimivat hieman eri tavalla
- Ammattimaiseen tiedostonkäsittelyyn liittyy monia asioita, joihin pitää kiinnittää huomiota erityistilanteissa kuten esimerkiksi
  - Tiedon haku tiedostosta – ”kirjanmerkkien” käyttö käsittelyn nopeuttamiseksi
  - Puskurointi – tiedostonkäsittelyn nopeutus vs. tiedon häviäminen
  - Lukitukset – voiko samaa tiedostoa käyttää useat käyttäjät, luku vs. kirjoitus
  - Näitä ja vastaavia edistyneitä asioita ei käsitellä tällä kurssilla

Data-alkioita tiedostossa riveillä

- Tyypillisesti **dataa** sisältävä **tekstitiedosto** muodostuu **riveistä** ja rivit muodostuvat **tietoalkioista**, jotka on erotettu toisistaan **erottimella**
- Oletusarvoisesti erotin on pilkku, mutta usein erotin voidaan valita. Suomessa erotin on tyypillisesti puolipiste (;) (vrt. desimaalipilkku)
- Erityisesti taulukkolaskentaohjelmat hyväksyvät datatiedostoja, joissa alkiot on erotettu pilkulla, comma separated values eli csv, ja se on tyypillinen tekstitiedostomuoto datalle. Alla esimerkki suomalaisesta sähkönkulutusdatasta, jossa alkiot on eroteltu puolipisteillä

```
Tunti;Energian yhteensä (kWh);Energian kulutus Yö 22 - 07 (kWh);Energian kulutus Päivä 07 - 22 (kWh);Status;"Kustannukset € (yhteensä)";"Perusmaksu € (myynti)";"Päivä 07 - 22 kustannus € (myynti)";"Yö 22 - 07 kustannus € (myynti)";"Perusmaksu € (siirto)";"Päivä 07 - 22 kustannus € (siirto)";"Yö 22 - 07 kustannus € (siirto)";"Energiavero €(Sähkön veroluokka 1)";"Lämpötila (°C)"
1.1.2014 0:00;2,10;2,10;;Mitattu;0,12;0,00;;;0,02;;0,05;0,05;2,8
1.1.2014 1:00;2,37;2,37;;Mitattu;0,13;0,00;;;0,02;;0,05;0,06;3,0
1.1.2014 2:00;2,20;2,20;;Mitattu;0,12;0,00;;;0,02;;0,05;0,05;3,0
```

Merkkijonot ja jäsenfunktiot

- Merkkijonoja sisältäville muuttujille on määritelty jäsenfunktioita eli metodeja
  - Esim. isupper(), islower(), isalpha(), upper(), lower(), split(), replace(), strip(), format()
  - Jäsenfunktiot muokkaavat muuttujaa halutulla tavalla ja palauttavat muutetun arvon (esim. upper() ja lower()) tai tietoa merkkijonosta (esim. isupper())
  - Dokumenteista löytyy tarkempia tietoja jäsenfunktiot, ks. esim. Ohjelmointiopas
- Myös tiedostonkäsittelyssä käytetään jäsenfunktioita, esim. readline, write ja close

Jäsenfunktiot

- Jäsenfunktiot liittyvät olio-ohjelmointiin ja tarkoittavat sitä, että operaatio kohdistuu ko. muuttujaan (olioon/luokkaan)
- Jäsenfunktioita käytetään pistenotaatiolla, esim. tiedostonkäsittelyn ja merkkijonojen yhteydessä
- Katso tarkemmin myöhemmin olevat koodiesimerkit ja ohjelmointivideo

```
Tiedosto.write("moi")
print("moi".upper()) # merkkijonon upper()-jäsenfunktio
print("moi".isupper()) # merkkijonon isupper()-jäsenfunktio
```

Merkkijonojen käsittely

- Tulostusta voi muotoilla monella tavalla kuten on nähty aiemmilla luennoilla
  - Tulostus tehdään print-käskyllä
  - print-käskyssä voidaan asettaa erotin- ja loppu-merkit sep ja end –parametreillä
  - Merkkijonoihin voi lisätä erilaisia ohjausmerkkejä kuten rivinvaihtomerkkejä '\n' ja sarkaimia (tabulaattori) '\t'. Huomaa, että nämä ovat usein ”näkymättömiä” merkkejä eli whitespace
- Mikäli tulosteiden suhteen on tarkkoja vaatimuksia, kannattaa ne muodostaa erikseen merkkijonoina
  - Tulostukseen käytetään edelleen print-käskyä
  - Merkkijonoja voi muodostaa mm. str ja round –käskyjen avulla erilaisia tietoja yhdistämällä
  - Merkkijonoja voi muodostaa format-jäsenfunktiolla, joka tarjoaa laajat muokkausmahdollisuudet
- format-jäsenfunktio mahdollistaa mm. numeroiden tallentamisen halutun levyisiin kenttiin halutulla määrällä desimaaleja
  - Alla on format-funktion perusesimerkit kokonaisluvun ja liukuluvun tulostukseen

```
Rivi = "Luku1 on kokonaislukuna {0:d}".format(Luku1)
print(Rivi)
Rivi = "Luku2 on liukulukuna {0:8.2f}".format(Luku2)
print(Rivi)
```

Merkkijonojen käsittely: format-käsky

- **Merkkijonon muodostaminen format-käskyllä**
  - merkkijono lainausmerkeissä ml. muotoilukoodit
  - aaltosulkujen ({}) sisällä tulostettavan muuttujan indeksi, kaksoispiste ja tulostuksen muoto ja tietotyyppi, esim. 0:d tai 1:5.2f
  - piste ja format, eli kutsutaan merkkijonon jäsenfunktiota format()
  - aaltosulkujen kohdalle sijoitettavat muuttujat suluissa parametreina, Luku1:n indeksi on 0 ja Luku2:n indeksi 1

```
Rivi = "Luku1 on {0:d} ja Luku2 on {1:5.2f}".format(Luku1, Luku2)
print(Rivi)
```

format-käskyn muotoilumerkeistä

- Tietotyyppi
  - **Kokonaisluku** tulostuu kirjaimilla 'd' (integer), tai 'i'
  - **Desimaaliluku** eli liukuluku tulostuu kirjaimella 'f' (float)
  - **Merkkijono** tulostuu kirjaimella 's' (string)
  - Yksi **merkki** tulostaa kirjaimella 'c' (char)
- Muotoilussa m.nf
  - m tarkoittaa koko kentän leveyttä merkkeinä
  - n tarkoittaa desimaalien määrää
  - f on muuttujan tietotyyppi (yllä, float eli liukuluku)
  - esim. numero 5 formaatilla 6.2f tuottaa ' `5.00`' (fonttina courier new, vakioleveys)
- Tulosteet voi tasata vasemmalla tai oikealle < ja > merkeillä, {0:>8s}
- Ks. tarkemmin Ohjelmointioppaasta tai Help-tiedostosta The Python Standard Library, 6.1. string — Common string operations

Merkkijonojen käsittely yhteenveto

- Merkkijonoja voidaan muodostaa **format-käskyllä**
  - Myös yhdistely palasista onnistuu kuten aiemminkin
- Merkkijonoja voidaan jakaa osiin **leikkauksilla**
  - `merkkijono[0:2:1]` –notaatiolla merkkejä voidaan erottaa niiden indeksien perusteella, esim. 2 ensimmäistä merkkiä
  - merkkijono voidaan jakaa osiin tietyn merkin kohdalta, ns. ”erotinmerkki”, joka on Suomessa tyypillisesti puolipiste esim.
    - `Etunimi;Sukunimi;Ika;Sukupuoli;Paino;Pituus`

Koodiesimerkkejä

Tiedoston kirjoittaminen ja lukeminen
Jäsenfunktiot
Valikkopohjainen ohjelma tiedostoilla

Tiedoston kirjoittaminen ja lukeminen

```
# Tekstitiedoston kirjoittaminen
Tiedosto = open("teksti.txt", "w") # avataan kirjoitusta varten
Tiedosto.write("Tämä menee tiedostoon.\n")
                # kirjoitetaan tiedostoon tekstiä + rivinvaihto
Tiedosto.close() # suljetaan tiedosto

# Tekstitiedostosta lukeminen
Tiedosto = open("teksti.txt", "r") # avataan lukemista varten
Rivi = Tiedosto.readline() # Luetaan tiedostosta yksi rivi
print(Rivi, end = "") # Tulostetaan rivi, ei lisätä rivinvaihtoa
Tiedosto.close() # suljetaan tiedosto
```

Jäsenfunktioesimerkkejä

```
Tiedosto.write("moi\n")  # Tiedoston write-jäsenfunktio
Tiedosto.close("")       # Tiedoston close-jäsenfunktio

print("moi".upper())     # Merkkijonon upper-jäsenfunktio
print("moi".isupper())   # Merkkijonon isupper-jäsenfunktio

Mjono = input("Anna merkkijono: ")
if (Mjono.isalpha()): # Merkkijonon isalpha-jäsenfunktio
    print("Mjono sisältää vain kirjaimia.")
```

Valikkopohjainen ohjelma tiedostoilla

```
def paaohjelma():
    Valinta = 1
    TiedostoLue = "L06Lue.txt"
    TiedostoKirjoita = "L06Kirjoita.txt"
    while (Valinta != 0):
        Valinta = valikko()
        if (Valinta == 1):
            Merkkijono = lueMerkkijono(TiedostoLue)
        elif (Valinta == 2):
            tallenna(TiedostoKirjoita, Merkkijono)
        elif (Valinta == 3):
            tallenna(TiedostoKirjoita, Merkkijono[::-1])
        elif (Valinta == 0):
            print("Lopetetaan")
        else:
            print("Tuntematon valinta, yritä uudestaan.")
        print()
    print("Kiitos ohjelman käytöstä.")
    return None

paaohjelma()
```

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

def lueMerkkijono(Tiedosto): # Lukee aina rivin 1
    Tdsto = open(Tiedosto, "r")
    Rivi = Tdsto.readline()
    Tdsto.close()
    Merkkijono = Rivi[:-1]
    return Merkkijono

def tallenna(Tiedosto, Merkit):
    Tdsto = open(Tiedosto, "a")
    Rivi = Merkit + '\n'
    Tdsto.write(Rivi)
    Tdsto.close()
    Tulosta = "Kirjoitettu tiedostoon '" + \
           Tiedosto + "' merkkijono '" + Merkit + \
           "'."
    print(Tulosta)
    return None

# paaohjelma on tässä aliohjelmien alla/jälkeen
```

Lopuksi

Osaamistavoitteet

Osaamistavoitteet

- Keskusmuisti ja oheismuisti
- Muuttuja ja kovakoodaus
- Muuttujien roolit
- Tekstimuotoiset datatiedostot: kirjoittaminen ja lukeminen
- Jäsenfunktiot: pistenotaatio, tiedostonkäsittely, merkkijonojen muokkaaminen – format-käsky tavoitetasolla

Täydennyksiä oppaan lukuun 6

Tyyliohjeita pienille Python-ohjelmille
ASPAn tarkistukset
Oppaan esimerkit ja käsitellyt asiat

”Pienen Python ohjelman tyyliohjeet”

- Pieni on suhteellinen käsite
  - Tällä kurssilla ”pieni ohjelma” on 1-10 riviä, ts. absoluuttisesti pieni
  - Teollisuudessa ”iso ohjelma” voi olla 1-10 miljoonaa riviä pitkä, ts. absoluuttisesti iso
- Tyyliohjeen ”pieni Python ohjelma” tarkoittaa aliohjelmia sisältävää n. 20-1000 rivin ohjelmia
  - Kurssin harjoitustyö on noin 200-400 riviä pitkä ”pieni Python ohjelma”, jolle nämä tyyliohjeet sopivat – ja tämän kurssin puitteissa kyseessä on ”iso ohjelma”
- Ohjelmoijat saavat työpaikalle tyyliohjeet, jotka voivat olla yritys- tai projektikohtaisia
  - Noudata aina projektikohtaisia tyyliohjeita – kieli, sovellusalue, asiakkaan tarve, …
- Tällä kurssilla harjoitellaan tyyliohjeiden noudattamista
  - Python –tulkki tarkistaa ohjelmien **syntaksin** ja ilmoittaa virheistä
  - CodeGrade tarkistaa ohjelman **tulosteet** ja ilmoittaa virheistä niissä (syntaksin lisäksi)
  - ASPA tarkistaa ohjelman **rakenteen** ja varoittaa, jos ohjelma ei noudata kurssin tyyliohjeita
  - Kurssilla on tyyliohje, jota tulee noudattaa tämän kurssin harjoitustyössä

Pienen Python-ohjelman tyyliohjeet 1

- Tiedostonkäsittely
  - Lähtökohtaisesti tiedostonkäsittely tulee keskittää aliohjelmiin
    - Tiedoston nimi parametrina aliohjelmaan. Yksi aliohjelma hoitaa tiedoston lukemisen ja toinen kirjoittamisen
    - Tiedostonkäsittelyn käskyt tiiviisti ”yhdessä pötkössä” open-close –käskyjen välissä
  - Datatiedostojen rakenne pitää miettiä tarkasti
    - Kirjoita tiedot tiedostoon systemaattisesti ja selkeästi
    - Lukemisen onnistuminen edellyttää tietoa rivi- ja sarakerakenteesta
    - Tyypillisesti yhdellä rivillä on yhteen kuuluvat tiedot ja eri riveillä vastaavat tiedot eri tapauksiin liittyen, tästä tarkemmin L07
  - Tiedosto kirjoitetaan write-funktiolla
    - Parametri on merkkijono, joka kannattaa usein muodostaa ennen write-käskyä
  - Tiedosto luetaan readline-funktiolla
    - Tiedoston loppu on helppo tunnistaa ja jokainen rivi voidaan käsitellä asianmukaisesti

ASPA tarkistukset L06

- Tiedostojen käsittelyyn liittyen ASPA tarkistaa ohjelmasta seuraavat asiat
  - Avattu tiedosto suljetaan
  - Tiedoston avaus, käsittely ja sulkeminen tapahtuu yhdessä aliohjelmassa, ts. lähtökohtaisesti aliohjelmaan lähetetään parametrina tiedostonimi eo. koodiesimerkkien mukaisesti
  - Käytetään open-close -rakennetta eikä open-with rakennetta. Tällä kurssilla suositellaan open-close –rakennetta, koska se on käytössä muissa C-pohjaisissa ohjelmointikielissä

Käsitellyt asiat oppaan luvussa 6

- Tiedoston kirjoittaminen: Esimerkki 6.1, 6.2, 6.4
- Tiedoston lukeminen: Esimerkki 6.1, 6.2, 6.4
- Tiedoston kirjoittaminen, lukeminen, tiedostokahva, avaaminen, sulkeminen, avaustilat, kirjanmerkit, merkistökoodaus
- Merkkijonojen metodeja ja muokkaaminen: Esimerkki 6.3