LUT Python ohjelmointiopas 2023 - osa 1
Sisällysluettelo

Alkusanat

Tervetuloa LUT-yliopiston Python-ohjelmointioppaan pariin. Tämä on oppaan yhdeksäs uusittu ja päivitetty versio. Alkuperäisen ohjelmointioppaan toteutti Jussi Kasurinen. Tämä uusin versio pohjaa siihen, mutta kaikki materiaali on päivitetty Pythonin versiolle 3 ja sisältöä on kehitetty sekä laajennettu muutenkin.

Ohjelmointioppaan ensimmäisen versio tehtiin käyttäen hyväksi kolmea verkkolähdettä. Näistä ensimmäinen on CH Swaroopin kirjoittama teos ’Byte of Python’ (www.byteofpython.info), toinen on Python Software Foundationin ylläpitämä Python-dokumenttiarkisto (docs.python.org) ja kolmas on Allen B. Downey’n kirjoittama ’How to Think Like a Computer Scientist: Learning with Python’ (http://www.greenteapress.com/thinkpython/), josta on otettu lähinnä täydentävää materiaalia. Python 3 -päivityksen yhteydessä käytettiin myös Mark Pilgrim’n Dive Into Python 3 -oppaan materiaalia (http://diveintopython3.org/). Alkuperäinen opas oli vapaa käännös ja yleisesti ottaen käännöksen alkuperäinen kieliasu pyrittiin säilyttämään, mutta tekstejä on muunneltu luettavuuden ja jatkuvuuden parantamiseksi. Myös esimerkit on lokalisoitu englanninkielisestä Linux-shell-ympäristöstä suomenkieliseen IDLE-kehitysympäristöön. Myöhempien päivitysten yhteydessä oppaan sisältö on muuttunut ja kehittynyt, joten yhtäläisyydet alkuperäisiin lähteisiin vähenevät koko ajan.

Tämä opas on tehty LUTin Ohjelmoinnin perusteet -kurssin ohjelmointioppaaksi. Oppaan rakenne perustuu kurssin viikkojakoon ja sisältöä muokataan kurssin kehittämisen yhteydessä. Oppaan liitteissä on tiivistettyä tietoa Pythonin kirjastoista, Pythonin yleinen sanasto, tulkin virheilmoitusten tulkintaopas, ohjelmoinnin tyyliopas sekä Python ohjelmien tyypillisiä ohjelmointirakenteita. Oppaan tueksi on olemassa myös asennusvideo, jossa käydään läpi asennuksessa oleelliset asiat.

Kaikki oppaan tehtävät on tehty Pythonin IDLE-kehitysympäristössä Windows-ympäristössä Pythonin versiolla 3.11.1. Samat työkalut ovat saatavilla myös Mac- tai Linux-ympäristöihin.

Tämä opas on päivitetty ja korjattu versio aiemmin ilmestyneistä Python-ohjelmointioppaista ja korvaa aiemmat versiot ohjelmoitaessa Pythonin versiolla 3.

Johdanto eli miksi ohjelmointi on kivaa ja tärkeää

Tietokoneiden kehitys lähti kasvuun toisen maailmansodan jälkeen ja sen sijaan, että yksi kone olisi osannut tehdä vain yhden asian, koneita alettiin ohjelmoida tekemään erilaisia asioita. Ensimmäiset ohjelmat olivat mahdollisimman lähellä tietokoneen itsensä ymmärtämää kieltä ja siksi ne kirjoitettiin konekielellä. Sen kanssa työskentely on tehokasta, mutta erittäin hidasta, eikä sitä nykypäivänä käytetä kuin hyvin harvoin.

Seuraava askel oli kirjoittaa ohjelmat englantia muistuttavalla kielellä ja kääntää tämä koodi konekielelle, mikä teki ohjelmien tuottamisesta paljon nopeampaa. Ensimmäisissä ohjelmointikielissä ei ollut paljoa toimintoja, mutta eipä ollut tietokoneissakaan tehoja – tai näyttöjä. Ohjelmakoodi oli usein varsin järjestelemätöntä, sillä niin kutsuttua spagettikoodia oli helppo kirjoittaa.

Seuraava vaihe oli siirtyä rakenteiseen ohjelmointiin. Ensin tulivat proseduraalinen ohjelmointi ja sitten olio-ohjelmointi, joista jälkimmäinen on tällä hetkellä vallitseva ohjelmointiparadigma. Tämän lisäksi on olemassa esimerkiksi funktionaalista ohjelmointia, joka voi olla vallitsevassa asemassa 20 vuoden kuluttua tai sitten jokin muu tapa todetaan paremmaksi. Hopealuotia ei ole vielä keksitty ja paljon työtä täytyy edelleen tehdä käsin.

Kaikki edellä mainitut ohjelmointiparadigmat kehitettiin 1950- ja 60-luvuilla ja niiden nouseminen vallitsevaan asemaan kesti vain vuosikymmeniä. Huomaa myös, että tällä kurssilla käyttämämme Python-ohjelmointikieli tukee kaikkia näitä ohjelmointitapoja.

Oli ohjelmointitapa mikä tahansa, ohjelmoinnin tarkoitus on kuitenkin sama: saada tietokone tekemään asioita, joista on hyötyä käyttäjälle. Kyseessä voi olla tekstin oikoluku, jutteleminen kaverin kanssa Internetissä tai pelien pelaaminen. Jokainen näistä on vaatinut ohjelmointia.

Nykyinen yhteiskunta ei tulisi toimeen ilman tietotekniikkaa, toimivia ohjelmia ja niitä kirjoittavia alan ammattilaisia. Ohjelmoinnin perusajatusten ymmärtäminen kuuluu insinöörin osaamiseen, sillä nykypäivänä kaikki laitteet jääkaapista auton kautta kännyköihin sisältävät miljoonia ja taas miljoonia rivejä koodia. On hyvä tietää edes jotain jääkaapin sielunelämästä, puhumattakaan monimutkaisista Excel-taulukoista tai interaktiivisen PowerPoint-esityksen taustalla olevasta ohjelmoinnista.

Ohjelmia voidaan kirjoittaa oikein ja ei-niin-oikein eli väärin. Kannattaa jo alusta lähtien opetella ohjelmoimaan oikein, sillä se helpottaa elämää (nimim. 10 vuotta väärin ohjelmoinut). Tämän oppaan tarkoitus onkin perehdyttää lukija ohjelmoinnin ihmeelliseen maailmaan ja tarjota hyväksi todettuja tapoja ratkaista ongelmia ohjelmoimalla.

Kannattaa pitää mielessä, että ohjelmoimaan oppii vain yhdellä tavalla – ohjelmoimalla.

Oppimisen – ja ohjelmoinnin – riemua!

Luku 1: Python ohjelmointiympäristö ja ohjelma

Ohjelmoinnin opettelu edellyttää ohjelmointiympäristöä, joten aloitetaan tutustumalla Python-ohjelmointiympäristöön. Sen jälkeen käydään läpi muutamia keskeisimpiä asioita ohjelmien tekemiseen liittyen kuten tiedon kysyminen käyttäjältä, tiedon muokkaaminen ml. tietotyyppien muunnokset ja tiedon tulostaminen. Tutustumme myös muuttujiin ja muihin ohjelmoinnissa tarvittaviin peruskonsepteihin. Tämän luvun tavoitteena on tulla tutuksi ohjelmointiympäristön ja ohjelmoinnin kanssa, jotta pystyt tekemään ja ajamaan ohjelmia sekä opetelemaan uusia ohjelmointikonsepteja laajentamalla aiempia ohjelmia.

Python-ohjelmien tekeminen IDLEllä

Python-tulkin lataamisen yhteydessä (http://python.org) samassa paketissa tulee IDLE-editori ohjelmien kirjoittamiseen. Python-ohjelmat ovat tekstitiedostoja, joissa käytetään .txt -päätteen sijasta .py -päätettä. Aloitetaan ensin yksinkertaisella ohjelmalla ja tutustutaan muutamiin ohjelman rakenteeseen liittyviin yleisiin asioihin, jotta jatkossa voimme keskittyä itse ohjelmointiin.

Editori ja tulostaminen

Windows-käyttäjille helpointa on aloittaa Pythonin käyttö IDLE-editorilla, joka on saatavilla myös Linux- ja MacOS-järjestelmille. IDLE on lyhenne sanoista Integrated DeveLopment Environment, ja sen haku- ja asennusohjeet löytyvät erillisestä asennusohjeesta. Ohjelma löytyy käynnistä-valikosta polun Käynnistä → Kaikki ohjelmat → Python 3.x → IDLE (Python GUI) (Start -> All Programs -> Python 3.x -> IDLE (Python GUI)) kautta (x:n tilalla Pythonin uusimman version numero). IDLE-ympäristössä on kaksi erilaista ikkunaa, interaktiivinen komentorivitulkki- ja editori-ikkunat.

Huomaa, että jatkossa esimerkkien merkintä **`>>>`** tarkoittaa tulkille syötettyä Python-käskyä. Seuraavassa kirjoitamme interaktiiviseen ikkunaan käskyn **`print("Hello World!")`** ja painamme Enter-näppäintä.

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC
v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> print("Hello World!")
Hello World!
>>>
```

Huomaa, että Python palauttaa tulostetun rivin välittömästi näytölle. Kirjoitit siis itse asiassa yksinkertaisen Python-käskyn `print`. Python käyttää `print` –käskyä sille annettujen arvojen tulostamiseen ruudulle. Tässä esimerkissä annoimme sille tekstin ”Hello World!”, jonka se tulosti välittömästi ruudulle.

Lähdekooditiedostot

Ohjelmoinnin opiskelussa on olemassa perinne, jonka mukaan ensimmäinen opeteltava asia on ”Hello World” -ohjelman kirjoittaminen ja ajaminen. Ohjelma on yksinkertainen koodinpätkä, joka ajettaessa tulostaa ruudulle tekstin ”Hello World”. Edellä tulostimme tämän tekstin interaktiivisesta ikkunasta, mutta kirjoitetaan tuo käsky nyt lähdekooditiedostoon, tallennetaan tiedosto ja suoritetaan ohjelma sen jälkeen IDLE-editorilla. Tässä vaiheessa jokaiselle ohjelmalle kannattaa tehdä oma tiedosto, jotta ohjelmat pysyvät erillään ja muodostavat selkeitä kokonaisuuksia.

Avaa valitsemasi editori, esim. IDLEssä File-valikosta valitaan New Window, ja kirjoita alla

olevan esimerkin mukainen koodi. Tämän jälkeen tallenna tiedosto nimellä `E01_1.py`. Tässä oppaassa on kymmeniä esimerkkiohjelmia, joten niiden tiedostonimet perustuvat oppaan lukuun ja esimerkin numeroon. Jos teet itse kaikki oppaan esimerkit ja tallennat ne samaan kansioon, ovat tehtävät järjestyksessä kuin tässä oppaassa. Oppaan ensimmäisen luvun ensimmäisen esimerkin tiedostonimessä E01_1 osa E01 tarkoittaa **E**simerkkiä luvussa **01** ja **_1** tarkoittaa ensimmäistä esimerkkiä. Huomaa, että tiedoston nimessä on erottimena toimivan pisteen jälkeen tarkentimena py. Tämän tarkentimen avulla ohjelmat tietävät, että tiedosto sisältää Python-koodia.

Esimerkki 1.1. Ohjelma lähdekooditiedostossa

```
print("Hello World!")
```

Tämän jälkeen aja ohjelma. Jos käytät IDLEä, onnistuu tämä editointi-ikkunan valikosta Run-> Run Module. Tämä voidaan toteuttaa myös pikavalintanäppäimellä F5.

Tuloste

```
>>>
Hello World!
>>>
```

Miten se toimii

Jos koodisi tuotti Tuloste-kohdassa näkyvän tekstin niin onneksi olkoon – teit juuri ensimmäisen kokonaisen Python-ohjelmasi!

Jos taas koodisi aiheutti virheen, tarkasta, että kirjoitit koodisi täsmälleen samoin kuin esimerkissä ja aja koodisi uudestaan. Erityisesti huomioi se, että Python näkee isot ja pienet kirjaimet eri merkkeinä. Tämä tarkoittaa sitä, että esimerkiksi ”Print” on eri asia kuin ”print”. Varmista myös, että et epähuomiossa laittanut välilyöntejä tai muutakaan sisennystä rivin alkuun. Palamme tähän asiaan myöhemmin tässä luvussa.

Itse ohjelma sisältää nyt vain Python-käskyn `print`, joka siis tulostaa tekstin ”Hello World!” näytölle. Voidaan myös sanoa, että `print` on funktio ja ”Hello World!” on merkkijono, mutta näihin termeihin palataan myöhemmin tarkemmin. Tässä vaiheessa Python-ohjelma sisältää käskyjä ja `print`-käsky on yksi Pythonin yleisimmin käytettyjä käskyjä.

Kommenttirivit

Pythonissa voi kirjoittaa lähdekoodin sekaan vapaamuotoista tekstiä, jolla käyttäjä voi kommentoida tekemäänsä koodia eli kirjoittaa näkyville, kuinka koodi toimii. Näitä rivejä sanotaan kommenttiriveiksi ja ne tunnistaa #-merkistä rivin ensimmäisenä merkkinä.

Kun tulkki havaitsee rivillä kommentin aloittavan merkin ”#”, lopettaa tulkki kyseisen rivin lukemisen ja siirtyy välittömästi seuraavalle riville. Kannattaa huomata, että #-merkillä voidaan myös laittaa kommentti annetun käskyn perään. Tällöin puhutaan koodinsisäisestä kommentoinnista. Lisäksi kommenttimerkkien avulla voimme poistaa suoritettavasta koodista komentoja, joita käytämme esimerkiksi testausvaiheessa välitulosten tarkasteluun. Jos meillä on esimerkiksi seuraava koodi:

Esimerkki 1.2. Kommentteja lähdekooditiedostossa

```
# Tämä on kommenttirivi
# Tämäkin on kommenttirivi
# Kommenttirivejä voi olla vapaa määrä peräkkäin

print("Tulostetaan tämä teksti")
# print("Tämä käsky on kommenttimerkillä poistettu käytöstä.")
print("Tämäkin rivi tulostetaan") # Tämä kommentti jätetään huomioimatta
```

Tulostaisi se seuraavan tekstin näytölle

```
>>>
Tulostetaan tämä teksti
Tämäkin rivi tulostetaan
>>>
```

Käytännössä tulkki jättää kommenttirivit huomioimatta. Erityisesti tämä näkyy koodin toisessa tulostuskäskyssä, jota ei tällä kertaa huomioida, koska rivin alkuun on lisätty kommenttimerkki. Vaikka kommentoitu alue sisältäisi toimivaa ja ajettavaa koodia, ei sitä siitäkään huolimatta suoriteta – se on siis ”kommentoitu pois”.

Kommentteja kannattaa käyttää järkevästi. Kirjoita kommentteihin esim. mitä ohjelmasi mikäkin vaihe tekee, sillä tästä on hyötyä, kun ohjelmaa tuntematon yrittää tulkita sen toimintaa. Kannattaa myös muistaa, että ihmisen muisti on rajallinen. Kun kuuden kuukauden päästä yrität lukea koodiasi, niin huomaat, että se ulkopuolinen olet myös sinä itse!

Loogiset ja fyysiset rivit

Fyysisellä rivillä tarkoitetaan sitä riviä, jonka näet koodia kirjoittaessasi – se alkaa ensimmäisestä sarakkeesta ja loppuu rivinvaihtomerkkiin. Looginen rivi taas on se kokonaisuus, minkä Python näkee yhtenä käskynä. Lähtökohtaisesti Python-tulkki käsittelee yhtä fyysistä riviä yhtenä loogisena rivinä.

Esimerkiksi käsky `print("Hello World")` on yksi looginen rivi. Jos se kirjoitetaan yhdelle riville, on se samalla myös yksi fyysinen rivi. Yleisesti ottaen Pythonin syntaksi tukee yhden rivin ilmaisutapoja erityisen hyvin, koska se pakottaa koodin helposti luettavaksi sekä selkeästi jaotelluksi.

On mahdollista kirjoittaa useampi looginen rakenne yhdelle riville, mutta se ei ole yleisesti ottaen suositeltavaa. Sen sijaan koodi kannattaa kirjoittaa niin, että yksi fyysinen rivi vastaa yhtä loogista käskyä. Käytä useampaa fyysistä riviä yhtä loogista riviä kohti ainoastaan poikkeustapauksessa, esim. jos rivistä on tulossa erittäin pitkä. Keskeinen tavoite on pitää koodi mahdollisimman helppolukuisena ja yksinkertaisena tulkita sekä tarvittaessa korjata.

Alla esimerkki tilanteesta, jossa looginen rivi jakautuu useammalle fyysiselle riville. Huomaa, että merkkijonon jakaminen useammalla riville edellyttää rivin lopettamista \-merkkiin (takakeno).

```
print("Tämä on merkkijono. \
Merkkijono jatkuu täällä.")
```

Tämä ohjelma tulostaa seuraavan merkkijonon:

```
Tämä on merkkijono. Merkkijono jatkuu täällä.
```

Loogisen rivin jakaminen useammalla fyysiselle riville ei edellytä aina kenoviivaa rivin loppuun. Näin käy esimerkiksi silloin, kun looginen rivi sisältää sulkumerkin, joka loogisesti merkitsee rakenteen määrittelyn olevan kesken. Esimerkiksi listojen tai muiden sarjamuotoisten muuttujien kanssa tämä on tavallista ja palaamme tähän asiaan luvussa 8.

Tyypillisiä virheilmoituksia

Ohjelmia kirjoittaessa saattaa tulla kirjoitusvirheitä ja tulkki huomauttaa niistä. Esimerkiksi edellä oli puhetta print ja Print –käskyjen erosta eli siitä, että kirjoittamalla ohjelmaan käskyn Print, ei tulkki suorita käskyä vaan antaa seuraavan virheilmoituksen:

```
NameError: name 'Print' is not defined
```

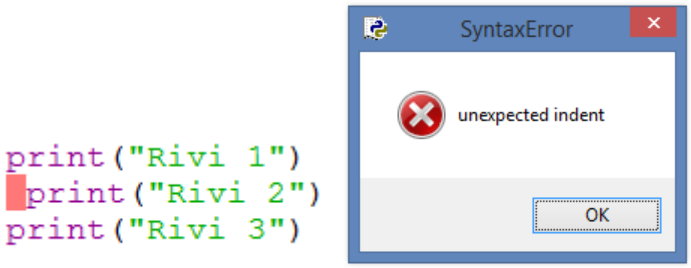
Tämä ongelma on helppo korjata vaihtamalla iso P pieneksi p:ksi.

Toinen varsin tyypillinen virhe liittyy koodin sisennykseen. Tässä vaiheessa ohjelmoinnin opiskelua kirjoitamme kaikki rivit alkamaan ensimmäisestä sarakkeesta ja jos näin ei toimita, antaa tulkki virheilmoituksen. Esimerkiksi seuraava kolmen rivin ohjelma johtaa virheilmoitukseen:

```
print("Rivi 1")
 print("Rivi 2")
print("Rivi 3")
```

Tällä kertaa tulkki ilmoittaa virheestä kahdella tavalla, näyttämällä SyntaxError –dialogin eli ”kirjoitusvirhe” ja sen lisäksi koodissa virheen aiheuttanut kohta on merkitty punaisella taustavärillä Kuva 1.1 mukaisesti. Virheilmoitusten muoto ja esitystapa voivat vaihdella eri versioissa ja ympäristöissä, mutta näitä virheilmoituksia kannattaa opetella lukemaan, sillä niiden avulla on tyypillisesti helppo löytää koodista virheen aiheuttanut kohta ja vähän asiaa pohtimalla syykin on usein helppo keksiä. Tulkki ei suorita ohjelmaa ennen kuin sen löytämä virhe on korjattu.

Kuva 1.1. Virhe lähdekoodissa



Tässä tapauksessa virheen syy on toisen rivin alussa olevassa ylimääräisessä välilyönnissä. *Syntax error* tarkoittaa sitä, että Python-koodissa on kielioppivirhe, eikä tulkki aina pysty sanomaan mitä siitä pitäisi korjata. Toinen tyypillinen virheilmoitus on *IndentationError*, joka kertoo sisennyksen olevan pielessä. Kannattaa siis muistaa, että **et voi aloittaa fyysisiä rivejä satunnaisesta kohtaa koodiriviä.** Sisennyksiä ja rakenteita, joissa luodaan uusia osioita, käsitellään tarkemmin luvussa 3.

Ohjelmointiympäristön ja ohjelmoinnin yhteenveto

Tutustuimme edellä Python ohjelmointiympäristöön, jonka perusasioita ovat IDLE-editori, lähdekooditiedostot, Python ohjelmat ja niiden suorittaminen. Nämä asiat kannattaa opetella hyvin, sillä ne ovat mukana kaikissa kurssilla tehtävissä ohjelmissa. Ohjelmia tehdessä tulee myös tehtyä virheitä, joten siksi katsoimme muutamia tyypillisiä virheitä ja virheilmoituksia, jotta et turhaan säikähdä niitä vaan opit lukemaan virheilmoituksia sekä etsimään ja korjaamaan ohjelmissa olevia virheitä. Näillä taidoilla pystyt tekemään Esimerkki 1.3 mukaisia ruudulle tekstiä tulostavia ohjelmia. Tämä esimerkki on vähän pidempi ja siten se tarjoaa monta mahdollisuutta tehdä virheitä sekä etsiä ja korjata ne. Nyt kun ohjelmointiympäristö on sinulle tuttu, voimme siirtyä tutustumaan tarkemmin ohjelmien tekemiseen.

Esimerkki 1.3. Sämpyläohje

```
print("Sämpyläohje")
print("===========")
print("Maitorahkaa:    1prk")
print("Vettä:          2,5dl")
print("Hiivaa:         25g")
print("Suolaa:         1,5tl")
print("Ruishiutaleita: 1,5dl")
print("Vehnäjauhoja:   8dl")
print("Voita:          50g")
print()
print("Liota hiiva nesteeseen. Lisää muut aineet ja alusta")
print("vehnäjauhoilla kimmoisaksi. Anna kohota 15 min. Leivo")
print("taikinasta sopivia palleroita ja anna kohota pellillä.")
print("Paista 225°C n. 12-15 min.")
```

Ohjelmat käsittelevät tietoa

”Hello World” ohjelman tekeminen ei ole vaikeaa, mutta myös ohjelman käytännön hyöty jää vähäiseksi. Oletettavasti haluat päästä kysymään käyttäjältä tietoja, muuntelemaan niitä ja tulostamaan ruudulle erilaisia tietoja. Onneksi Python mahdollistaa tämän kaiken.

Pythonilla pystyy käsittelemään tietoa eri tavoin ja käsittelytavat riippuvat käsiteltävästä tiedosta. Lähtökohtaisesti **merkkijonot** ja **numerot** ovat erilaisia tietotyyppejä, esimerkiksi nimi kuten ”Kalle” on merkkijono, jossa on monta (aakkos/ASCII) merkkiä peräkkäin. Ja numero voi taas olla esimerkiksi **kokonaisluku** 5 tai **desimaaliluku** 5,12345. Tämän lisäksi ohjelmoinnissa tieto voi olla **vakio**, jolloin sen arvo ei voi muuttua, tai **muuttuja**, jolloin sen arvo voi muuttua ohjelman kuluessa. Käydään nämä käsitteet läpi seuraavaksi vähän tarkemmin, jotta ymmärrämme paremmin, miten ne liittyvä ohjelmointiin.

Vakio ja merkkijono

Tähän mennessä tekemämme ohjelmat ovat tulostaneet merkkijonoja ja ”Hello World” on tyypillinen *merkkijonovakio.* Kyseessä on siis jono merkkejä eli kirjaimia, jotka eivät muutu, vaikka ohjelman ajaisi kuinka monta kertaa. Tilanne on sama, jos vaihdamme lainausmerkkien sisälle numeron `5, 10` tai `655.36`. Tällöin meillä on edelleen käytössä merkkijonovakio, koska ohjelma käsittelee tietoja kirjainmerkkeinä. Käytännössä **merkkijonot** erotetaan ohjelmakoodissa lainausmerkeillä (”) tai heittomerkeillä (’) ja ne ovat siis vakioita.

Ohjelmaan voidaan kirjoittaa myös numeroita kuten `5, 10` tai `655.36` ilman lainausmerkkejä. Jos ohjelmassa on esimerkiksi käsky `print(1.2345)`, niin se tulostaa näytölle luvun `1.2345` yhtä monta kertaa kuin ohjelma suoritetaan. Pelkkä numero on vakio, eikä sitä voi muuttaa.

Numerot

Pythonissa on kolme erilaista numerotyyppiä: kokonaisluvut, desimaaliluvut (tarkkaan ottaen *liukuluvut* eli floating point number) sekä kompleksiluvut.

- Esimerkiksi 42, 54321 tai -5 ovat kokonaislukuja, koska ne eivät sisällä desimaaliosaa.
- Desimaalilukuja ovat esimerkiksi 3.23 ja 52.3E-4, jossa merkki E tarkoittaa kymmenpotenssia. Tässä tapauksessa, 52.3E-4 on siis $52.3 * 10^{-4}$. **Huomaa, että ohjelmoitaessa desimaalierottimena käytetään pistettä eikä suomalaisittain pilkkua.**
- Kompleksilukuja ovat vaikkapa (-5+4j) ja (2.3 – 4.6j).

Muuttujat

Pelkkien vakioarvojen käyttäminen rupeaa nopeasti rajoittamaan ohjelmien joustavuutta, joten tarvitsemme jonkinlaisen keinon tallentaa tietoa sekä tehdä niihin muutoksia. Tämä on syy siihen, että ohjelmointikielissä kuten Python on muuttujia. Muuttujat ovat juuri sitä mitä niiden nimi lupaa eli ne ovat eräänlaisia säilytysastioita, joihin voit tallentaa mitä haluat ja muutella tätä tietoa tarpeen mukaan vapaasti. Muuttujat tallentuvat tietokoneesi muistiin käytön ajaksi, ja tarvitset jonkinlaisen tunnisteen niiden käyttämiseen. Tämän vuoksi muuttujille annetaan nimi aina käyttöönottovaiheessa. Muuttujiin liittyy siis kaksi asiaa – muuttujan nimi ja sen arvo – ja muuttujan arvoa voidaan muuttaa aina tarpeen mukaan.

Muuttuja otetaan käyttöön ja esitellään samalla, kun sille annetaan ensimmäinen arvo. Erillistä esittelyä tai tyypin määrittelyä ei tarvita, eli Pythonille ei tarvitse kertoa tuleeko muuttuja arvo olemaan esimerkiksi kokonaisluku tai merkkijono. Lisäksi muuttujaan, joka sisältää vaikkapa kokonaisluvun, voidaan tallentaa tilalle merkkijono. Joissain tapauksissa Pythonin syntaksi tosin vaatii, että käytettävä muuttuja on jo aiemmin määritelty joksikin soveltuvaksi arvoksi. Täten emme voi esimerkiksi tulostaa määrittelemättömän muuttujan arvoa.

Alla on määritelty kaksi merkkijonomuuttujaa, `Etunimi` ja `Sukunimi`, sekä kaksi numeromuuttujaa, `Ika` eli kokonaislukumuuttuja sekä `Paino` eli desimaaliluku.

```
Etunimi = ”Brian”
Sukunimi = ”Kottarainen”
Ika = 23
Paino = 123.4
```

Muuttujien nimeäminen

Muuttujan nimi ovat esimerkki *tunnisteesta*. Tunniste tarkoittaa nimeä, jolla yksilöidään jokin tietty asia. Samanniminen muuttuja ei siis voi kohdistua kahteen sisältöön. Muuttujien nimeäminen on melko vapaata, joskin seuraavat säännöt pätevät muuttujien sekä kaikkeen muuhunkin nimeämiseen Pythonissa:

- Nimen ensimmäinen merkki on oltava kirjain (iso tai pieni) taikka alaviiva ‘_’.
- Loput merkit voivat olla joko kirjaimia (isoja tai pieniä), alaviivoja tai numeroita (0-9).
- Nimet ovat aakkoskoosta riippuvaisia (eng. case sensitive), eli isot ja pienet kirjaimet ovat tulkille eri merkkejä. Tulkin kannalta ”talo” ja ”Talo” ovat eri sanoja samalla tavoin kuin ”talo” ja ”valo”.

Näiden sääntöjen perustella kelvollisia nimiä ovat muun muassa `Etunimi`, `i`, `_mun_nimi`, `nimi_23` ja `a1b2_c3`. Epäkelpoja nimiä taas ovat esimerkiksi `2asiaa`, `taa on muuttuja` ja `-mun-nimi`.

Python-tulkki tarkistaa muuttujien nimet eikä hyväksy esim. muuttujan nimen alkamista numerolla. Ohjelmointikielissä on erilaisia sääntöjä mm. kielioppiin eli syntaksiin liittyen ja tulkki tarkistaa niiden noudattamisen. Useimmat tulkin tarkistukset liittyvät käskyjen kielioppiin ja pystyäkseen ohjaamaan ohjelmien toteutusta tarkemmin, monissa yrityksissä on käytössä tyyliohje. Palaamme tyyliohjeeseen tarkemmin tämän luvun lopussa ja esittelemme silloin tämän oppaan tyyliohjetta mm. muuttujien nimeämiseen. Tässä vaiheessa kannattaa kirjoittaa muuttujien nimet samalla tavalla kuin esimerkeissä, ts. aloittaen isolla kirjaimella ja jättämällä ääkköset pois nimistä tyyliin `Ika`.

Muuttujien tietotyypit

Muuttujille voidaan antaa arvoksi mitä tahansa Pythonin tuntemia tietotyyppejä, kuten merkkijonoja tai numeroita. Kannattaa kuitenkin huomata, että jotkin operaatiot eivät ole mahdollisia keskenään. Esimerkiksi kokonaisluvusta ei voi vähentää merkkijonoa eli `32 - "auto"` ei siis toimi.

On myös syytä huomata, että eri tietotyypit voivat toimia eri tavoin. Esimerkiksi numeroiden ja merkkijonojen kohdalla + ja * operaatiot toimivat eri tavoin esimerkin 1.4 mukaan.

Esimerkki 1.4. Tietotyypin vaikutus tulokseen

```
>>> a = 3
>>> b = 5
>>> a + b
8
>>> a = "3"
>>> b = "5"
>>> a + b
'35'
>>>
>>> a = 3
>>> b = 5.2
>>> a - b
-2.2
>>>
```

Numero-muuttujilla voidaan tehdä normaaleja laskutoimituksia, mutta merkkijonojen osalta vain yhteenlasku on sallittua. Numero ja merkkijonomuuttujien välillä on myös määritelty muutamia operaatioita, mutta niihin palataan myöhemmin. Lähtökohtaisesti muuttujilla kannattaa käyttää kuvaavia nimiä ja laskentaoperaatioita tehdessä kannattaa käyttää maalaisjärkeä. Kuten edellä näkyy, numeroiden yhteenlasku tuottaa odotetun tuloksen, mutta merkkijonojen yhteenlasku johtaa niiden yhdistämiseen. Pythonissa kokonaisluvuilla ja desimaalinluvuilla voidaan laskea suoraan eikä Python erottele niitä tällaisessa käytössä mitenkään.

Laskentaohjelma

Kirjoita alla olevan Esimerkki 1.3 mukainen ohjelma IDLEllä ja tallenna tiedosto nimellä `E01_5.py`. Muista antaa Python-koodille pääte **py**, jotta käyttöjärjestelmä ja editorit tunnistavat tiedostosi Python-koodiksi. Suorita ohjelma ja katso, että ymmärrät sen toiminnan.

Esimerkki 1.5. Muuttujien käyttäminen ja vakioarvot

```
Luku = 4
print(Luku)
print(Luku + 1)
print(Luku)
Luku = Luku + 1
Luku = Luku * 3 + 5
print(Luku)

Teksti = "Tässä meillä on tekstiä."
print(Teksti)
```

Tuloste

```
>>>
4
5
4
20
Tässä meillä on tekstiä.
>>>
```

Kuinka se toimii

Ohjelmasi toimii seuraavasti: Ensin määrittelemme muuttujalle `Luku` vakioarvon `4` käyttäen sijoitusoperaattoria `(=)`. Tätä riviä sanotaan käskyksi, koska rivillä on määräys, että jotain pitäisi tehdä. Tässä tapauksessa siis sijoittaa arvo `4` muuttujaan `Luku`. Sijoitusoperaatio toimii loogisesti aina siten, että se arvo, johon sijoitetaan, on operaattorin vasemmalla puolella ja se, mitä sijoitetaan, on oikealla. Seuraavalla rivillä olevalla toisella käskyllä me tulostamme muuttujan `Luku` arvon ruudulle funktiolla `print`. Ja kun hyppäämme taas rivin eteenpäin tulostamme ruudulle arvon `Luku + 1`, eli tässä tapauksessa arvon `5`. Muuttujamme `Luku` on edelleen arvoltaan `4`, minkä kertoo myös seuraava `print`-lause.

Tämän jälkeen kasvatamme luvun arvoa yhdellä ja tallennamme sen takaisin muuttujaan `Luku`. Eli vasta tässä vaiheessa muuttujan arvo muuttuu! Seuraava rivi tuottaa vielä hieman monimutkaisemman laskennan ja tallentaa tuloksen jälleen muuttujaan `Luku`, eli itseensä. Tulostuskäsky `print` antaakin – kuten olettaa saattoi – tällä kertaa arvon 20.

Samalla menetelmällä toiseksi viimeinen rivi tallentaa merkkijonon muuttujaan `Teksti` ja viimeinen rivi tulostaa sen ruudulle.

Laskutoimitukset

Pythonin interaktiivista ikkunaa voi käyttää laskimena. Anna tulkille operandit ja operaattorin ja tulkki tulostaa sinulle vastauksen. Voit myös kokeilla IDLEn interaktiivisen ikkunan avulla yksinkertaisia yhdistelmiä ja rakenteita. Esimerkiksi tulkkia voidaan käyttää seuraavilla tavoilla:

```
>>> 2+2
4
>>> # Kommenttirivi
... 2+2
4
>>> 2+2  # Kommenttirivi ei sotke koodia
4
>>> (50-5*6)/4
5
>>> # Jakolasku tuottaa usein desimaaliluvun
... 7/3
2.3333333333333335
>>> 7%3
1
```

Yhtäsuuruusmerkki (‘=’) toimii sijoitusoperaattorina. Sijoitusoperaattoriin päättyvä lauseke ei tuota välitulosta vaan sillä voit syöttää vakiotietoja, joilla taas voit suorittaa laskutehtäviä. Alla olevassa esimerkissä muuttujiin `Leveys` ja `Korkeus` sijoitetaan arvot, joilla sitten tehdään varsinainen laskutoimitus:

```
>>> Leveys = 20
>>> Korkeus = 5*9
>>> Leveys * Korkeus
900
```

Kokonaislukuja ja desimaalilukuja voidaan käyttää vapaasti ristiin. Python suorittaa automaattisesti muunnokset sopivimpaan yhteiseen muotoon:

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

Operaattorit ja operandit

Seuraavaksi tutustumme yleisimpiin operaattoreihin, jotka liittyvät muuttujilla operointiin. Aikaisemmin olet jo tutustunut niistä muutamaan, (+) (*) ja (=) -operaattoreihin ja tärkeimmät operaattorit on kuvattu Taulukko 1.1.:ssa.

Useimmat kirjoittamasi käskyt sisältävät jonkinlaisen operaattorin. Yksinkertaisimmillaan tämä voi tarkoittaa vaikka riviä `2 + 3`. Tässä tapauksessa + edustaa lauseen operaattoria ja `2` sekä `3` lauseen operandeja.

Voit kokeilla operaattorien toimintaa käytännössä syöttämällä niitä Pythonin interaktiiviseen ikkunaan ja katsomalla, minkälaisia tuloksia saat aikaiseksi. Esimerkiksi yhteen- ja kertolaskuoperaattorit toimivat näin:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

Laskentajärjestyksestä

Lähtökohtaisesti suoritusjärjestys noudattaa matemaattisia laskusääntöjä. Kuten normaalisti matematiikassa, voit muuttaa järjestystä käyttämällä sulkuja. Suluilla työskennelläänkin täysin samalla tavoin kuin matemaattisesti laskettaessa, sisimmistä ulospäin ja samalla tasolla laskentajärjestyksen mukaisesti. Esimerkiksi jos haluat, että yhteenlasku toteutetaan ennen kertolaskua, merkitään se yksinkertaisesti `(2 + 3) * 4`.

```
>>> Leveys = 20
>>> Korkeus = 5*9
>>> Leveys * Korkeus
900

>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

Taulukko 1.1 Laskentaoperaattorit

Operaattori	Nimi	Selite	Esimerkki
=	Sijoitus	Sijoittaa annetun arvon kohdemuuttujalle	`Luku = 5` sijoittaa muuttujalle luku arvon 5. Operaattori toimii ainoastaan mikäli kohteena on muuttuja.
+	Summa	Laskee yhteen kaksi operandia	`3 + 5` antaa arvon `8`. `'a' + 'b'` antaa arvon `'ab'`.
-	Erotus	Palauttaa joko negatiivisen arvon tai vähentää kaksi operandia toisistaan	`-5.2` palauttaa negatiivisen numeron. `50 - 24` antaa arvon `26`.
*	Tulo	Palauttaa kahden operandin tulon tai toistaa merkkijonon operandin kertaa	`2 * 3` antaa arvon `6`. `'la' * 3` antaa arvon `'lalala'`.
**	Potenssi	Palauttaa x:n potenssin y:stä.	`3 ** 4` antaa arvon `81` (eli. `3 * 3 * 3 * 3`)
/	Jako	Jakaa x:n y:llä	`4/3` antaa arvon `1.3333333333333333`
//	Tasajako	Palauttaa tiedon, kuinka monesti y menee x:ään	`4 // 3` antaa arvon `1`
%	Jakojäännös	Palauttaa x:n jakojäännöksen y:stä.	`8%3` antaa `2`. `-25.5%2.25` antaa `1.5`.

Laskut ohjelmakoodissa

Lähdekooditiedostoon tallennetut Python-käskyt toimivat vastaavalla tavalla kuin interaktiivisessa ikkunassa. Voit siis kokeilla kaavoja ja käskyjä interaktiivisessa ikkunassa ja sitten tallettaa ne tiedostoon ja suorittaa ohjelman osana, kuten Esimerkki 1.6 osoittaa.

Esimerkki 1.6. Lausekkeiden käyttö

```
Pituus = 5
Leveys = 2

Pinta = Pituus * Leveys
print("Nelikulmion pinta-ala on", Pinta)
print("ja kehä on", 2 * (Pituus + Leveys))
```

Tuloste

```
>>>
Nelikulmion pinta-ala on 10
ja kehä on 14
>>>
```

Kuinka se toimii

Kappaleen pituus ja leveys on tallennettu samannimisiin muuttujiin. Me käytämme näitä muuttujia laskeaksemme kappaleen pinta-alan ja kehän pituuden operandien avulla. Ensin suoritamme laskutoimituksen pinta-alalle ja sijoitamme operaation tuloksen muuttujaan `Pinta`. Seuraavalla rivillä tulostamme vastauksen tuttuun tapaan. Toisessa tulostuksessa laskemme vastauksen `2 * (Pituus + Leveys)` suoraan `print`-funktion sisällä.

Huomaa myös, kuinka Python laittaa automaattisesti tulostuksissa välilyönnin tulostettavan merkkijonon `'Pinta-ala on'` ja muuttujan `Pinta` väliin. Tämä on yksi Pythonin erityispiirteistä, jotka tähtäävät ohjelmoijan työmäärän vähentämiseen.

Tiedon kysyminen käyttäjältä

Usein tulee tilanne, että etukäteen luodut merkkijonot tai koodiin kiintoarvoina määritellyt muuttujat eivät pysty hoitamaan kaikkia tarvittavia tehtäviä. Haluamme syöttää ohjelmaan arvoja ajon aikana ja antaa käyttäjälle mahdollisuuden vaikuttaa ohjelman tuottamiin tuloksiin. Seuraavaksi tutustumme Pythonin tarjoamiin tapoihin pyytää käyttäjältä syötteitä ja tallentaa niitä muuttujiin.

Pythonissa käyttäjältä voidaan kysyä tietoa `input`-käskyllä (funktiolla). Käskyn perään laitetaan sulkuihin käyttäjälle näytettävä teksti ja käsky palauttaa käyttäjän antaman syötteen. Huomaa, että käsky palauttaa kaikki merkit rivinvaihtomerkkiin asti, mutta ilman sitä. Ja tieto tosiaan tulee ohjelmaan yhtenä merkkijonona. Nimen kysyminen on tyypillinen esimerkki tästä (Esimerkki 1.7).

Esimerkki 1.7. input-funktion käyttö

```
Merkkeja = input("Anna merkkijono: ")
print("Annoit merkkijonon", Merkkeja)
```

Tuloste

```
>>>
Anna merkkijono: Kumiankka
Annoit merkkijonon Kumiankka
>>>
```

Kuinka se toimii

Tässä esimerkissä pyydämme käyttäjältä merkkijonon `input`-käskyllä, sijoitamme sen muuttujaan ja tulostamme sen lopuksi käyttäjälle `print`-käskyllä. Kaikki tieto käyttäjältä saadaan Pythonissa `input`-käskyllä ja tiedot tulostetaan käyttäjälle `print`-käskyllä, joten niitä kannattaa opetella käyttämään.

Tyyppimuunnokset merkkijonosta luvuksi

Kuten aiemmin oli puhetta, Pythonissa on kolme perustietotyyppiä – merkkijono, kokonaisluku ja desimaaliluku – ja näiden välillä voidaan tehdä tyyppimuunnoksia. Muunnokset luvuiksi tehdään funktioilla `int`() ja `float`(), joista `int` palauttaa kokonaisluvun, integer'n, `float` palauttaa desimaaliluvun, floating point number, ja molemmille annetaan suluissa muutettava arvo. Esimerkki 1.8 demonstroi näitä funktioita. Huomaa, että Pythonissa desimaalierotin on piste eikä pilkku.

Esimerkki 1.8. Lukujen kysyminen käyttäjältä, int ja float

```
Syote = input("Anna ympyrän säde kokonaislukuna: ")
Sade = int(Syote)
Syote = input("Anna piin likiarvo desimaalilukuna: ") # odotetaan arvoa 3.14
Pii = float(Syote)

Ala = Sade * Sade * Pii
print("Pinta-ala on", Ala)
print("Kehä on", 2 * Sade * Pii)
```

Tuloste

```
>>>
Anna ympyrän säde kokonaislukuna: 3
Anna piin likiarvo desimaalilukuna: 3.14
Pinta-ala on 28.26
Kehä on 18.84
>>>
```

Kuinka se toimii
Ensimmäisellä rivillä sijoitamme muuttujalle `Syote` arvoksi `input`-funktion tuloksen. `input`-funktio saa syötteenä merkkijonon, nyt ”Anna ympyrän säde kokonaislukuna: ”, ja tulostaa tämän merkkijonon ruudulle toimintaohjeeksi käyttäjälle. Käyttäjä syöttää mieleisensä luvun vastauksena kehotteelle, jonka jälkeen annettu luku tallentuu muuttujaan `Syote`. Pythonin `input`-funktio ei tiedä, että käyttäjä syöttää nimenomaan numeron, vaan se ottaa vastaa merkkijonon, tässä tapauksessa merkin ”3”. Laskussa tarvitsemme numeroa, joten muutamme `int`-funktiolla käyttäjän syöttämän merkkijonon numeroksi. Piin arvon osalta idea on sama, mutta nyt pyydämme käyttäjältä desimaaliluvun kokonaisluvun sijasta. Tämä jälkeen voimme laskea ja tulostaa pinta-alan ja kehän pituuden edellisen esimerkin tavoin.

Muunnettaessa merkkijono kokonaisluvuksi tai desimaaliluvuksi ei ole varmaa, että muunnos onnistuu. Siksi tyyppimuutoksissa kannattaa olla tarkkana ja palaamme myöhemmin tarkemmin tyyppimuunnoksiin liittyvään virheenkäsittelyyn. Siihen asti kysyttäessä käyttäjältä tietoja kannattaa antaa selkeät ohjeet siitä, minkälaista tietoa hänen tulisi syöttää: kokonaisluku, desimaaliluku vai merkkijono.

Yhteenveto

Osaamistavoitteet

Tässä ensimmäisessä luvussa tutustuimme Python-ohjelmointiympäristöön ja perusohjelmaan, jonka pohjalta pystyt tekemään Python-ohjelmia ja opettelemaan uusia käskyä ja rakenteita. Luvun keskeiset asiat on nimetty alla olevassa listassa, jonka avulla voit tarkistaa, että tunnistat nämä asiat ja sinulla on käsitys siitä, mitä ne tarkoittavat.

- Python ohjelmointiympäristön käyttö, ohjelmien teko ja suorittaminen
    - IDLE-editori, ohjelman ajo, virheiden tunnistaminen ja korjaaminen
    - Pythonin käyttö interaktiivisessa ikkunassa, esim. laskin =, +, -, *, /, **, //, %
- Ohjelmointi
    - Muuttujat ja tietotyypit, merkkijono, kokonaisluku ja desimaaliluku
    - Loogiset ja fyysiset rivit
    - Kommentit (#, E1.2, E1.9)
    - Vakiot, numerot ja merkkijonot, esim. 3.14 ja ”Kalle” (E1.5, E1.9)
    - Tiedolle tehtävät operaatiot: operaattori, operandi, laskentajärjestys, tyyppi-muunnokset (Taulukko 1.1, E1.8, E1.9)
    - Tietojen tulostaminen (print, E1.1, E1.3, E1.5, E1.9)
    - Tietojen kysymisen (input, E1.7, E1.8, E1.9)
- Keskeiset käskyt
    - print E1.2, input E1.7, int ja float E1.8 sekä E1.9

Mikäli joku käsite tuntuu oudolta, kannattaa lukea siihen liittyvät asiat uudestaan. Tässä vaiheessa on hyvä muistaa em. käsitteet ja käskyt, sillä seuraavaksi teemme niihin perustuvia ohjelmia.

Pienen Python-perusohjelman tyyliohjeet

Kuten aiemmin oli puhetta, Python-tulkki tarkistaa ohjelman virheiden varalta ja pysähtyy, jos se löytää ohjelmasta virheen. Kokemattomien ohjelmoijien tyypillisin virhe on syntaksivirhe eli ohjelma ei noudata Pythonin kielioppisääntöjä. Syntaksivirheistä pääsee eroon katsomalla dokumenteista käskyn syntaksin ja noudattamalla sitä. Käyttämällä käskyä tarpeeksi monta kertaa sen oppii muistamaan ulkoa, jolloin ohjelmointi helpottuu ja nopeutuu.

Kielioppisääntöjen lisäksi ohjelmointiin on olemassa tyyliohjeita, joissa annetaan tarkempia ohjeita ohjelmien tekemiseen. Esimerkiksi tyyliohjeessa voidaan käskeä kirjoittamaan kaikki kommentit tietyllä kielellä, esim. suomeksi tai englanniksi, kieltää tiettyjen rakenteiden käyttö, edellyttää koodin tallentamista tiettyyn paikkaan ja tiettyjen testien tekemistä ennen kuin ohjelma julkaistaan. Tyypillisesti tyyliohjeilla pyritään varmistamaan, että kaikkien yrityksen ohjelmoijien kirjoittamat ohjelmat näyttävät samanlaiselta ja on tehty samojen periaatteiden mukaisesti, jotta ohjelman ymmärtäisi ja sitä pystyisi ylläpitämään muutkin kuin sen alkuperäinen tekijä. Nämä ovat tärkeitä näkökulmia varsinkin silloin, kun ohjelmoija sairastuu, siirtyy toiseen projektiin tai eläkkeelle, tai jättää yrityksen. Tyypillisesti tyyliohjeilla pyritään varmistamaan ohjelmien ymmärrettävyys ja ylläpidettävyys myös poikkeustilanteissa.

Tämän oppaan ohjelmat noudattavat LUTin Python-ohjelmoinnin tyyliohjetta. Tämän oppaan jokaisen luvun lopussa esitellään lukuun liittyvät tyyliohjeet, joita tulee noudattaa tämän oppaan pohjalta tehdyissä pienissä Python-perusohjelmissa. Tämän tyyliohjeen keskeinen tavoite on tehdä toimivia ja ylläpidettäviä ohjelmia, joita on helppo ymmärtää ja muokata. Tyyliohjeet muuttuvat jonkin verran oppaan edessä, sillä oppaan tavoite on käydä läpi ohjelmoinnin perusasiat ja uusien rakenteiden sekä käskyjen käyttöönoton myötä ohjelmointityyliä kannattaa muuttaa ottamaan uudet mahdollisuudet huomioon.

Yleisiä ohjeita

1. Tee tehtävänannon mukaan toimivia ohjelmia. Lue tehtävänanto uudestaan ohjelman tekemisen jälkeen ja varmista, että olet noudattanut kaikkia annettuja ohjeita.
2. Käytä vain oppaassa käsiteltyjä käskyjä ja rakenteita, esim. älä käytä f-string:iä.
3. Kysy tiedot käyttäjältä `input`-käskyllä ja tarvittaessa muuta syöte oikeaan tietotyyppiin kysymisen jälkeen seuraavalla rivillä esim. `int` tai `float`-käskyllä. Nämä käskyt voi yhdistää, jos tietää varmasti, mitä tekee (ks. E1_9).
4. Käytä sulkeita laskentakaavoissa muuttamaan ja/tai selkiyttämään ryhmittelyä, esim. lasku ”2 * (3 + 4)” johtaa tulokseen 14.

Muuttujat

5. Nimeä muuttujat yksikäsitteisesti, käsiteltävää tietoa kuvaavasti ja muodosta nimet systemaattisesti erityisesti isojen kirjain osalta, esim. `Lukumaara`, `Nimi`, `Paino`
6. Muodosta nimet tarvittaessa useista sanoista ja liittää sanat toisiinsa uudet sanat suuraakkosilla, esim. `SummaSuurin`, `ListaTulokset`, `TiliPaiva`
7. Nimissä voi käyttää yleisesti käytettyjä selkeitä lyhenteitä, esim. `Lkm`, `Pvm`, `Nro`
8. Suosi vakiintuneiden tunnusten ja nimien käyttöä, esim. `PII = 3.14`
9. Määrittele muuttujat kootusti ohjelman alussa ja anna niille arvot, tarvittaessa voit alustaa muuttujan sopivalla arvolla tyyliin `Lkm=0`
10. Älä käytä mitäänsanomattomia ja/tai harhaanjohtavia nimiä, esim. kirjaimet `a`, `b`, `c` ja `a1`, `a2`, `a3` jne.
11. Älä käytä muuttujien ja tiedostojen nimissä skandinaavisia merkkejä (å, ä, ö, Å, Ä, Ö). Tyypillisesti ne voi korvata a tai o -kirjaimella.
12. Käytä merkkijonojen tunnisteena lainausmerkkiä (”), ei heittomerkkiä.

Kielletyt käytännöt

13. Ohjelmakoodi, nimet ja kommentit on kirjoitettava suomeksi, ei esim. englanniksi.
14. Tässä oppaassa ei käsitellä olio-ohjelmointia, joten olio-ohjelmia ei tule tehdä tämän oppaan pohjalta.
15. Koodin kopiointi muualta on kielletty.

Tämän oppaan tyyliohje on pääosin ohje eli suositus siitä, miten toimimalla pystyy tekemään ymmärrettäviä, selkeitä ja ylläpidettäviä ohjelmia. Siksi tyyliohjeiden mukaan tehtyjä ohjelmia on helpompi tarkastaa ja löytää niissä olevia virheitä, mikä koskee niin itse ohjelman kirjoittajaa kuin muita, jotka katsovat ohjelmaa.

Huomaa, että yllä nimetyt **Kielletyt käytännöt** ovat sellaisia, ettei niitä hyväksytä tämän tyyliohjeen pohjalta eikä näiden ohjeiden vastaisia ohjelmia tule tehdä. Jos valikoit itse oppaasta noudattamasi kohdat, otat myös itse vastuun tekemistäsi ohjelmistasi. Usein tyyliohjeen noudattaminen näkyy esim. kurssin tarkemmin arvioitavien ohjelmien kuten harjoitustyön ja tentin arvosanoissa, jolloin ohjeiden mukaan tehdyt ohjelmat saavat parempia arvosanoja kuin niiden vastaiset ohjelmat. Koodin kopioinnin kielto perustuu yritysmaailmassa mm. tekijänoikeusasioihin, kun taas tässä oppaassa koodin kopiointi on kielletty oppimisnäkökulmasta. Kopioimalla jonkun muun tekemää koodia ei opi mitään ja siksi se on kielletty.

Tyyliohjeet tehdään tyypillisesti analysoimalla tehtyjä ohjelmia ja tunnistamalla niiden virheet sekä miettimällä keinoja niiden välttämiseen. Siksi näitä ohjeita ja niiden taustoja ei tarvitse ymmärtää tässä vaiheessa, vaan ne selviävät oppaan edetessä. Etenkin, jos seuraa itse tekemiään virheitä ja miettii, miten ne voisi välttää.

Luvun asiat kokoava esimerkki

Esimerkki 1.9 toimii luvun asiat kokoavana esimerkkinä ohjelmasta, jonka pystyt nyt tekemään. Tiivistetysti siinä näkyy tiedon kysyminen, tyyppimuunnoksia ja tiedon tulostaminen.

Esimerkki 1.9. Leivontaohjeen rakennus

```
Ohje = input("Mikä ohje tämä on? ")
Syote = input("Kuinka monta desiä vettä? ")
Vesi = float(Syote)
Syote = input("Kuinka monta grammaa hiivaa? ")
Hiiva = int(Syote)
# Huomaa, että seuraavassa input() on laitettu suoraan float():n sisään.
Jauho = float(input("Kuinka monta desiä jauhoja? "))

print(Ohje)
print("==========")
print("Vettä:       ", Vesi, "dl")
print("Hiivaa:      ", Hiiva, "g")
print("Vehnäjauhoja:", Jauho, "dl")
```