Python-kurssin oppikirja (nimi työn alla)

Mikko Tampio

10. lokakuuta 2017

Sisältö

1 Pythonin asennus ja tekstin tulostaminen **2**
- 1.1 Yleistietoa Pythonista . . . 2
- 1.2 Tästä kirjasta . . . 2
- 1.3 Python 2 vai Python 3? . . . 3
- 1.4 Ympäristön asentaminen . . . 3
- 1.5 Lyhyt IDLE-esittely . . . 4
- 1.6 Pythonin käyttäminen komentoriviltä . . . 5
- 1.7 Ensimmäinen ohjelma . . . 6
- 1.8 Useiden rivien tulostaminen . . . 7
- 1.9 Kommentointi . . . 7
- 1.10 Tehtäviä . . . 8

2 Muuttujat ja tietotyypit **10**
- 2.1 Muuttujien idea . . . 10
- 2.2 Tekstinsyöttö . . . 11
- 2.3 Tietotyypit . . . 11
- 2.4 Merkkijonot . . . 12
- 2.5 Kokonaisluvut . . . 12
- 2.6 Liukuluvut . . . 14
- 2.7 Tehtäviä . . . 14

3 Ehtolauseet **16**
- 3.1 `if`-lause yksinkertaisimmillaan . . . 16
- 3.2 `bool`-tyyppi . . . 16
- 3.3 Monimutkaisempia `if`-lauseita . . . 18
- 3.4 Tehtäviä . . . 19

4 Toisto **21**
- 4.1 Toisto `while`-silmukalla . . . 21
- 4.2 `for`-lause . . . 22
- 4.3 Tehtäviä . . . 23

Sanasto **25**

Luku 1

Pythonin asennus ja tekstin tulostaminen

1.1 Yleistietoa Pythonista

Python on yleiskäyttöinen ohjelmointikieli, johon tämä teos keskittyy. Aloittelijoille kieltä suositellaan sen yksinkertaisuuden ja käyttäjäystävällisyyden vuoksi; ammattikäytössä Pythoniin törmää usein tieteellisessä tutkimuksessa, mutta sillä on mahdollista myös esimerkiksi palvelinohjelmointi ja peliohjelmointi. Ominaisuuksiltaan Python on lähellä 2010-luvun muita käytetyimpiä ohjelmointikieliä (mm. Java, C, C++, JavaScript), minkä vuoksi sen parissa oppii varmasti hyödyllisiä taitoja, jotka helpottavat muihin kieliin siirtymistä.

Ensimmäisen version Pythonista julkaisi Guido van Rossum vuonna 1991, ja nykyisin sen kehityksestä vastaa Python Software Foundation (`https://www.python.org/`). Säätiön sivuilta löytyy kattava englanninkielinen Python-opas (`https://docs.python.org/3/tutorial/`) sekä Pythonin dokumentaatio (`https://docs.python.org/3/`) eli yksityiskohtainen kuvaus kaikista Pythonin sisäänrakennetuista ominaisuuksista (ymmärtäminen vaatii perustiedot Pythonista).

1.2 Tästä kirjasta

Tämä hyvin keskeneräinen oppikirja on tarkoitettu lukion kurssimateriaaliksi ohjelmointia vähän tai ei lainkaan harrastaneille. Pyrin Pythonin alkeiden opettamisen lisäksi yleissivistämään lukijaa tietojenkäsittelytieteen maailmasta; tätä varten käytän muutamia käsitteitä, joita ei erikseen tarvitse opetella, jos se ei mielekkäältä tunnu. Sanasto kirjan lopussa toivottavasti auttaa, jos jonkin sanan merkitys on epäselvä.

Oppimisen tukena kannattaa käyttää (tai on pakko käyttää, jos kirja valmistuu hitaammin kuin opiskelet) tässä luvussa listattuja muita Python-

oppaita.

Kirjan tehtävät ovat pääasiallisesti peräisin Helsingin yliopiston Python-kurssimateriaalista (`https://www.cs.helsinki.fi/group/linkki/materiaali/python-perusteet/materiaali.html`) sekä Ohjelmointiputkan Python-oppaasta (`https://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=python_01`). Tarkemmat lähdetiedot on merkitty tehtäviin.

1.3 Python 2 vai Python 3?

Python-ohjelmointia aloittava voi törmätä yllättävään ongelmaan: edes oppaan ensimmäinen, yhden rivin esimerkkikoodipätkä ei toimi. Kyse on siitä, että Pythonista on yhä käytössä useita eri versioita: vanhempi Python 2 ja uudempi Python 3.

Kirjoitushetkellä Python 3:n julkaisuhetkestä alkaa olla jo kymmenisen vuotta, mikä on valtavan pitkä aika tietotekniikan maailmassa. Erinäisistä syistä (kuten siitä, että useat kirjastot toimivat yhä vain Python 2:lla) Python-ohjelmoijat ovat kuitenkin vaihtaneet uudempaan versioon melko hitaalla tahdilla, eikä ole epätodennäköistä, että uudetkin Python-oppaat opettavat vanhaa Python 2:ta.

Totuus on kuitenkin se, että aloittelijan ei ole mitään syytä olla käyttämättä Python 3:a. Se on ainut Python-versio, johon enää tulee uusia ominaisuuksia ja bugikorjauksia – Python 2:n aktiivinen kehitys loppui jo viisi vuotta sitten. Suomenkielistä ohjelmoijaa ilahduttaa lisäksi se, että ääkköset toimivat Python 3:ssa ilman sen erityisempää säätämistä. Siispä tämä kirja opettaa asiat Python 3 -tavalla; jos näet virheilmoituksia, kokeile muuttaa koodistasi tällaiset rivit

```
1 print "tekstin tulostaminen"
```

tällaisiksi

```
1 print("tekstin tulostaminen")
```

Toinen usein esiin tuleva ero on, että `raw_input()`-funktion nimi on Python 3:ssa pelkkä `input()`.

Jos tietokoneessasi on jo asennettuna Python, varmista, että kyseessä on uudempi versio. Useissa Linux-pohjaisissa käyttöjärjestelmissä on molemmat; jos `python`-komento avaa Python 2:n, kokeile komentoa `python3` (sama pätee myös kaikkiin Python-työkaluihin).

1.4 Ympäristön asentaminen

Pythonin voi ladata osoitteesta `https://www.python.org/downloads/` (muista valita Python 3!). Windowsilla ja Macilla mukana tulee IDLE-niminen

ohjelma, jonka avulla Python-koodia voi kirjoittaa ja suorittaa kätevästi.

Linux-pohjaisilla käyttöjärjestelmillä jokin versio Pythonista on hyvin todennäköisesti jo asennettu. Jos näin ei ole (tai koneella on Python 2), Python tulee asentaa omaa pakettienhallintaohjelmaa käyttäen; lisäksi IDLE-ohjelma todennäköisesti puuttuu. Esimerkiksi Ubuntussa ne asennetaan syöttämällä konsoliin seuraavat komennot:

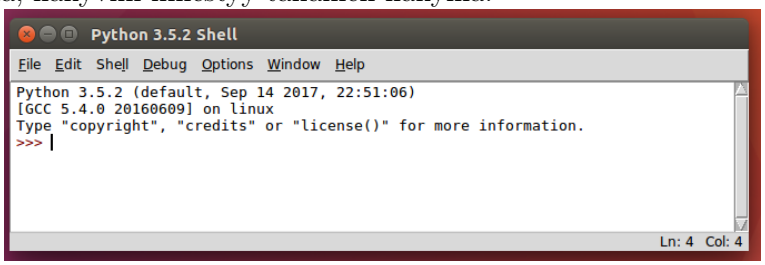
```
$ sudo apt install python3
$ sudo apt install idle3
```

Lyhyt huomio notaatiosta: merkkiä $ komennon alussa ei syötetä konsoliin, vaan sitä käytetään tavanomaisesti erottamaan ohjeissa komennot ja se, mitä komennot tulostavat konsoliin. Asentaakseen Pythonin käyttäjä siis kirjoittaa `sudo apt install python3`

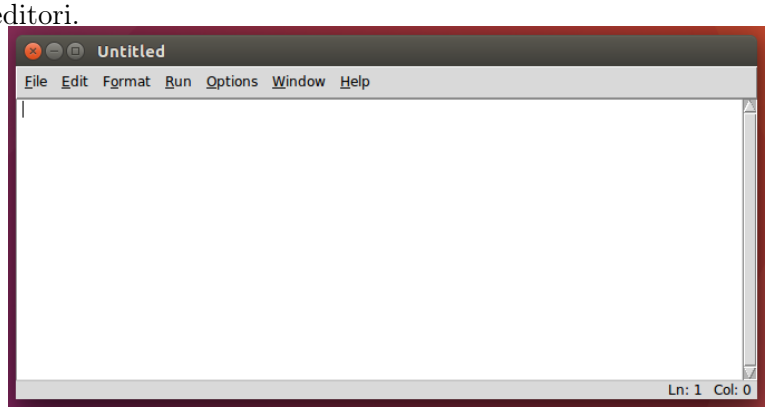
Pythonin asennukseen liittyviin ongelmiin löytyy todennäköisesti vastaus Googlesta.

1.5 Lyhyt IDLE-esittely

Kun IDLE (Integrated Development and Learning Environment)-ohjelman avaa, näkyviin ilmestyy tällainen näkymä.



Kyseessä on Python-komentotulkki – ohjelma, joka suorittaa välittömästi siihen syötetyn Python-koodin. Jos niin haluaa, kaikki Python-ohjelmansa voi syöttää rivi riviltä komentotulkkiin, mutta kätevämpää on pitää ne erillisissä tiedostoissa. Valitsemalla `File -> New File` avautuu IDLE:n tiedostoeditori.



Kun Python-ohjelmansa on kirjoittanut IDLE:n koodieditorilla, sen voi suorittaa painamalla `F5`.

1.6 Pythonin käyttäminen komentoriviltä

Aloittelija pärjää hyvin graafisilla työkaluilla, mutta halutessaan kaiken tässä kirjassa esitetyn voi tehdä Pythonin komentorivityökaluilla. Konsolin saa auki Windowsilla painamalla `Windows + R` ja kirjoittamalla `cmd`, Unix-pohjaisilla käyttöjärjestelmillä (mm. OSX ja eri Linuxit) yleensä painamalla `CTRL + ALT + T`.

Alla on muistin virkistämiseksi taulukoituna peruskomennot, joiden avulla esimerkkien suorittamisen pitäisi onnistua.

Windows-komento	**Unix-komento**	**Selitys**
`cd` *kansio*	`cd` *kansio*	Siirtyy annettuun kansioon
`dir`	`ls`	Tulostaa nykyisen kansion sisältämät tiedostot
`help` *komento*	`man` *komento*	Näyttää lisätietoja annetusta komennosta
`python`	`python` tai `python3`	Avaa interaktiivisen Python-komentotulkin
`python` *tiedosto*	`python` *tiedosto* tai `python3` *tiedosto*	Suorittaa annetun Python-tiedoston

Seuraa lyhyt esimerkkisessio, jonka aikana Unix-pohjaista käyttöjärjestelmää käyttävä henkilö siirtyy `python`-kansioon tiedostojärjestelmässään, listaa sen sisältämät tiedostot ja suorittaa `kiva_ohjelma.py`-nimisen tiedoston, joka tulostaa ruudulle viestin `Se toimii!`.

```
$ cd python
$ ls
kiva_ohjelma.py
toinen_ohjelma.py
oppikirja.pdf
$ python kiva_ohjelma.py
Se toimii!
```

Kuten edellisessä esimerkissä, $ komentojen alussa ei ole osa komentoa vaan erottaa komennot ja niiden tulostaman tekstin.

Aihetta voisi käsitellä enemmänkin, mutta tässä vaiheessa Python-opintoja se ei ole kovin mielekästä. Konsolin käytöstä innostunut tai ongelmia kohdannut lukija voi turvautua Googleen.

1.7 Ensimmäinen ohjelma

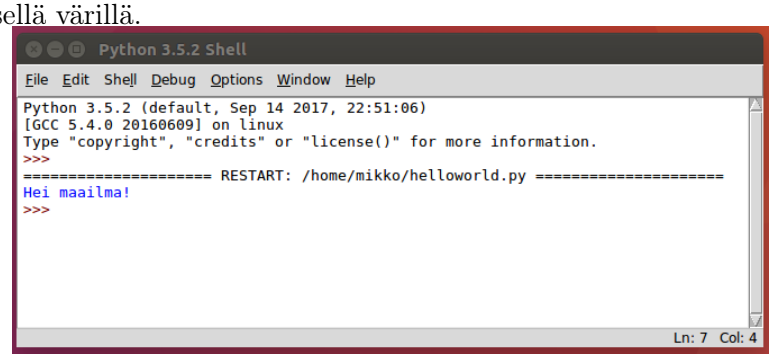
Ensimmäisistä ohjelmista klassisin on yksinkertainen sovellus, joka tulostaa ruudulle tekstin `Hei maailma!`

Alla on esitetty Hei maailma -ohjelman Python-toteutus. Kirjoita se IDLE:n avulla tiedostoon ja suorita painamalla `F5`.

Esimerkki 1.1: Hei maailma!

```
1 print("Hei maailma!")
```

Jos onnistuit, teksti `Hei maailma!` tulostui IDLE:n komentotulkkiin sinisellä värillä.



Vastedes sitä, mitä Python-koodi tulostaa ruudulle, merkitään tässä kirjassa näin.

```
Hei maailma!
```

Jos saat virheilmoituksen, palaa ensin tämän kirjan selostukseen Python 2:n ja Python 3:n eroista ja varmista, että sinulla on asennettuna Python 3. Vika voi olla myös koodissa itsessään – tarkista, että kopioit esimerkkiohjelman merkilleen oikein. Aloitteleva ohjelmoija huomaa nopeasti, että toisin kuin ihminen, tietokone ei yritä arvailla, mitä käyttäjä on voinut tarkoittaa: jos vaikkapa viimeisen sulun unohtaa, on seurauksena virhe, vaikka onkin ilmiselvää, mitä ohjelmoija on halunnut tehdä.

Kun ohjelman saa toimimaan, koittaa aika leikkiä. Muuta tulostettavaa tekstiä muuttamalla lainausmerkkien sisältöä tai kokeile laittaa useampi tulostus peräkkäin kirjoittamalla useita rivejä, joissa on jokaisessa `print()` ja sulkujen sisällä haluttu teksti merkkijonojen välissä. Tehtävissä on lisäehdotuksia.

Vaikka luvun ainut esimerkki onkin yksinkertainen, se täyttää Python-ohjelman määritelmän. Python-ohjelma koostuu (yksinkertaistetusti) lauseista. Ohjelmassamme on yksi ainoa lause: `print("Hei maailma!")`. Se on tarkemmin sanottuna *funktiokutsu*, jossa funktiolle `print` annetaan *argumenttina* teksti `Hei maailma!`, joka on tarkennettuna merkkijono. Myöhemmin

tutustutaan toisiin funktioihin, joilla voimme tehdä muutakin kuin vain tulostaa tekstiä.

Funktioihin ja merkkijonoihin palataan tulevissa luvuissa.

1.8 Useiden rivien tulostaminen

Edellisen osion lyhyt esimerkkiohjelma tulostaa Python-konsoliin yhden ainoan rivin tekstiä. Jos rivejä haluaa useampia, voi toimia monella tavalla; helpointa on yksinkertaisesti kirjoittaa monta `print`-lausetta, joista jokainen tulostaa oman rivinsä. Tässä havainnollistava esimerkki.

```
1 print("Ensimmäinen rivi...")
2 print("... ja toinen.")
```

Esimerkki tulostaa näytölle seuraavan tekstin:

```
Ensimmäinen rivi...
... ja toinen.
```

Muitakin tapoja on. Kenoviivaa (\) voidaan käyttää koodinvaihtomerkkinä, jolloin sen avulla voidaan kirjoittaa rivinvaihtomerkki (`\n`). Rivinvaihtomerkin kohdatessaan Python jatkaa tekstiä uudelta riviltä, joten seuraavalla esimerkillä on tismalleen sama ulostulo.

```
1 print("Ensimmäinen rivi...\n... ja toinen.")
```

Näin on tehtävä, koska Python ei salli merkkijonon jatkuvan seuraavalle riville. Tämä rajoitus kuitenkin poistuu, jos yhden lainausmerkin sijaan merkkijonon alussa ja lopussa käytetään kolmea.

```
1 print("""Ensimmäinen rivi...
2 ... ja toinen.""")
```

1.9 Kommentointi

Kommentit ovat hyödyllinen työkalu, jonka avulla koodiin voi jättää pieniä merkintöjä. Seuraava esimerkki havainnollistaa kommenttien eri käyttötarkoituksia; kommentti alkaa #-merkistä ja jatkuu rivin loppuun saakka. Python-tulkki jättää huomiotta kaiken kommentissa olevan, joten kommenttien sisältö ei vaikuta mitenkään ohjelman toimintaan, mutta joitakin käyttötarkoituksia niillä siitä huolimatta on.

Esimerkki 1.2: Kommennointi

```
1 # Koodannut Olli Ohjelmoija 4.10.2017
2 # Erkki Esimerkki korjasi bugin 6.10.2017
3
4 # Tulostaa tekstin "kissa"
5 print("kissa")
6
7 # print("koira")
```

Ensinnäkin kommenteilla voi jättää koodiin tietoja siitä, minkälaisia muutoksia eri henkilöt ovat siihen tehneet. Nykyisin tämä hoidetaan yleensä erillisellä versiohallintaohjelmalla, mutta aloittelijoiden ryhmätyössä voi olla kätevintä käyttää koodin alkuun sijoitettavia kommentteja.

Tämän lisäksi kommenteilla voidaan selittää, mitä hankalat tai muuten epäselvät pätkät koodia tekevät. Aloittelija voi toki jättää itselleen esimerkin kaltaisia kommentteja, jotka edistyneemmälle ohjelmoijalle ovat täysin turhia, mutta yleensä kannattaa miettiä kriittisesti kommenttien tarpeellisuutta – tarpeettomat kommentit voivat lisätä koodin lukemisen hankaluutta. Hyvä nyrkkisääntö on, että kommenttien pitäisi kertoa, *miksi* sen kirjoittanut ohjelmoija päätyi juuri valitsemaansa ratkaisuun; koodi voi puhua omasta puolestaan.

Esimerkin kolmas kommentti näyttää, kuinka tietyn koodirivin voi nopeasti ottaa pois käytöstä muuttamalla sen kommentiksi. Todettakoon kuitenkin, että on huonoa tyyliä jättää koodiin sadoittain pois kommentoituja rivejä – kuten turhat ja itsestäänselvät kommentit, nekin saavat koodia lukevan henkilön kiinnittämään huomionsa itseensä. Jos kommentoituja koodirivejä projektiinsa kuitenkin jättää, on hyvätapaista kirjoittaa selventävä kommentti, joka selittää, miksi rivit on otettu pois käytöstä.

1.10 Tehtäviä

1.10.1 Muuta esimerkkiohjelma 1.1 tulostamaan oma nimesi.

1.10.2 Muokkaa esimerkkiohjelmaa 1.1. Kokeile, mitä käy, jos...

(a) ... viimeisen sulun poistaa

(b) ... viimeisen lainausmerkin poistaa

(c) ... lainausmerkkien sisällä ei ole mitään

(d) ... `print`-funktion kirjoittaa väärin (vaikkapa `pirnt`)

1.10.3 Kokeile eri tapoja tulostaa useita rivejä tekstiä. Tee ainakin kaksi ohjelmaa, jotka tulostavat

```
Hei maailma!
Englanniksi se on Hello, World!
```

1.10.4 Mitä hyötyä kommenteista on, jos ne voisi poistaa, eikä ohjelma muuttuisi mitenkään?

Luku 2

Muuttujat ja tietotyypit

2.1 Muuttujien idea

Tähän asti olemme antaneet `print`-funktiolle suoraan merkkijonon. Sen lisäksi voimme käyttää aiemmin määriteltyä muuttujaa.

```
1 x = "Tulostettava teksti"
2 print(x)
```

Ohjelman tuloste on

```
Tulostettava teksti
```

Esimerkki havainnollistaa, että muuttuja viittaa sille aiemmin annettuun arvoon. Muuttuja määritellään *`muuttuja = arvo`*; kuten seuraava esimerkki näyttää, muuttuja voidaan myös määritellä uudelleen, jolloin se viittaa aina viimeiseksi määriteltyyn arvoonsa.

Esimerkki 2.1: Muuttujan määrittely ja uudelleenmäärittely

```
1 var = "muuttujan arvo"
2 print(var)
3 var = "toinen arvo"
4 print(var)
```

Tämä ohjelma tulostaa

```
muuttujan arvo
toinen arvo
```

Esimerkkien muuttujat olivat nimiltään `x` ja `var`. Muuttujilleen voi antaa nimeksi lähes mitä tahansa, mutta tärkeitä rajoituksia on kaksi.

- Muuttuja ei saa sisältää erikoismerkkejä, joita Python syntaksissaan käyttää. Esimerkiksi `muut(tuja` ja `muut"tuja` ovat virheellisiä muuttujia, joiden käyttämisestä on seurauksena syntaksivirhe.
- Muuttuja voi sisältää numeroita, mutta se ei saa alkaa sellaisella. `nimi4` on sallittu muuttuja, mutta `2kissaa` ei.

Tämän lisäksi on kiellettyä viitata sellaiseen muuttujaan, jota ei ole määritelty. Muuttujan voi joko määritellä itse (kuten esimerkeissä tehdään) tai saada valmiiksi Pythonilta.

2.2 Tekstinsyöttö

Muuttujien todellinen hyöty käy esille, kun täytyy käsitellä arvoja, jota ei tunneta ennen koodin suorittamista. Tällaisia ovat mm. tiedostojen sisältö, kellonaika, päivämäärä ja käyttäjän syöttämä teksti. Seuraava esimerkki havainnollistaa näistä viimeistä: käyttäjän nimeä kysytään `input`-funktiolla, minkä jälkeen se säilötään muuttujaan.

Esimerkki 2.2: Tekstinsyöttö

```
1 nimi = input("Syötä nimesi: ")
2
3 print("Hei, " + nimi + "!")
```

`input`-funktiolle annetaan merkkijono, joka näytetään käyttäjälle, joka voi sitten kirjoittaa Python-konsoliin haluamansa vastauksen. Esimerkin ulostulossa käyttäjä syöttää nimekseen `sari`, ja sitten ohjelma tulostaa tervehdyksen, jossa `nimi`-muuttuja on korvattu käyttäjän syöttämällä arvolla.

```
Syötä nimesi: sari
Hei, sari!
```

Toinen uusi asia esimerkissä on se, että merkkijonoja voi yhdistää `+`-merkillä. Esimerkiksi `"kis"+"sa"` on sama asia kuin `"kissa"`.

2.3 Tietotyypit

Tähän asti on selvitty pelkillä merkkijonoilla, mutta Pythonin muihkin tyyppeihin on tarpeen tutustua. Tyyppiä voi ajatella joukkona arvoja, joilla on samanlaisia ominaisuuksia. Esimerkiksi kaikilla merkkijonoilla, kuten `"hevonen"` ja `"veropetos"`, on se ominaisuus, että niitä voi edellä opitun mukaan yhdistellä `+`-merkillä.

Pythonin perustyyppeihin lukeutuvat merkkijono, totuusarvo ja numeeriset tyypit, joihin seuraavaksi tutustutaan. Totuusarvoja käsitellään ehtolauseiden yhteydessä. Tyyppejä on lisääkin, ja niitä voi jopa määritellä itse – lisää olio-ohjelmointia käsittelevässä luvussa.

Eri tyyppisiä arvoja voi myös muuttaa toiseen tyyppiin. Tämä on tarpeen esimerkiksi silloin, kun haluamme käsitellä käyttäjän syötettä (`input`-funktio palauttaa aina merkkijonon) lukuna. Lisää aiheesta kokonaislukuosion esimerkissä.

2.4 Merkkijonot

On aika syventää tietämystä merkkijonoista. Syntaksi niiden määrittelyyn tunnetaan jo: lainausmerkkien sisällä oleva osa koodia tulkitaan merkkijonoksi. Tässä lisää tarpeellisia ominaisuuksia, joita tarvitaan merkkijonojen kanssa työskentelyssä.

Esimerkki 2.3: Merkkijonojen ominaisuuksia

```
# Merkkijonojen yhdistäminen
"a" + "b" # 'ab'

# Merkkijonon pituuden selvittäminen
len("kissa") # merkkijonon pituus eli 5

# Merkkijonon toistaminen
"ha" * 4 # 'hahahaha'

# Isot ja pienet kirjaimet
"isolla".upper() # 'ISOLLA'
"PIENELLÄ".lower() # 'pienellä'

# Merkkijonoksi muuttaminen
# (6 on tässä kokonaisluku; lue seuraava osio)
str(6) # merkkijono '6'
```

Esimerkkilausekkeita voi testata syöttämällä ne Python-konsoliin, joka laskee niiden arvon ja tulostaa sen. Tiedostossa työskennellessään on muistettava tulostaa haluamansa operaation tulos `print`-funktiolla.

2.5 Kokonaisluvut

Kokonaislukuja voi kaikessa yksinkertaisuudessaan määritellä vain kirjoittamansa halutun luvun: `36`. Lainausmerkkejä ei tule käyttää, koska muuten Python tulkitsee arvon merkkijonoksi. Monet funktiot, kuten äsken esitelty merkkijonon pituuden laskemiseen käytetty `len`, palauttavat kokonaisluvun.

Useissa ohjelmointikielissä kokonaisluvuilla on määrätyt minimi- ja maksimiarvot, mutta Pythonissa kiinteitä rajoja ei ole. On teoreettisesti mahdollista luoda niin iso kokonaisluku, että tietokoneen muisti loppuu kesken, mutta käytännössä laskeminen käy sitä ennen niin hitaaksi, ettei Pythonin käyttö onnistu.

Esimerkki 2.4: Kokonaislukujen ominaisuuksia

```
# Peruslaskutoimituksia
4 + 6 # 7
3 - 8 # -5
2 * 3 # 6

# Eri jakolaskut
7 / 3 # 2.3333333333333335
7 // 3 # 2

# Potenssilasku
13 ** 21 # 247064529073450392704413
```

Kuten esimerkki osoittaa, kokonaisluvuilla on samanlaisia laskutoimituksia kuin matematiikassa yleensä. Erilaisia jakolaskuoperaattoreita on kaksi: `/` tuottaa liukuluvun (katso seuraava osio) ja `//` kokonaisluvun. Kokonaislukujakolaskussa pyöristetään alaspäin; esimerkiksi `5//2` on `2`, ei `3`.

Usein aloittelijoilta unohtuva asia on se, että arvoja täytyy muuttaa toiseen tyyppiin, jos niitä haluaa käsitellä eri tavalla. Käyttäjän syöte `input`-funktiolla on merkkijono; vain merkkijonoja voi yhdistää `+`-operaattorilla. Tämän vuoksi seuraavassa esimerkissä muutamme ensin käyttäjän syötteen kokonaisluvuksi ja sitten jälleen merkkijonoksi, kun haluamme yhdistää sen toiseen merkkijonoon.

Esimerkki 2.5: Käyttäjän iän kysyminen

```
luku = int(input("Syötä ikäsi: "))

print("Vuoden päästä olet " + str(luku+1) + " vuotta vanha.")
```

Tässä ohjelman ulostulo, kun käyttäjä syöttää iäkseen 19.

```
Syötä ikäsi: 19
Vuoden päästä olet 20 vuotta vanha.
```

Ensimmäisellä rivillä kysytään käyttäjän ikää, joka muutetaan kokonaisluvuksi `int`-funktiolla, koska sitä halutaan pian käsitellä aritmeettisesti. Laskutoimituksen `luku+1` tulos on kokonaisluku, joten se on muutettava merkkijonoksi `str`-funktiolla, jotta sen voi yhdistää muihin merkkijonoihin.

2.6 Liukuluvut

Liukuluvut ovat ohjelmoinnissa desimaalilukujen vastine. Niiden käyttö on muuten samanlaista kuin kokonaislukujen, mutta kokonaislukuosa ja desimaaliosa erotetaan pisteellä (ei pilkulla, kuten suomen kielessä yleensä). `float`-funktiolla voi muuttaa merkkijonoja ja kokonaislukuja liukuluvuiksi.

Esimerkki 2.6: Liukulukulaskuri

```
1 luku = float(input("Syötä liukuluku: "))
2
3 print("4.6 * " + str(luku) + " = " + str(4.6 * luku))
```

Kun käyttäjä syöttää `7.8`, ohjelman ulostulo on seuraava:

```
Syötä liukuluku: 7.8
4.6 * 7.8 = 35.879999999999995
```

Tarkkaavainen lukija saattaa huomata, että laskun tulos on epätarkka. Liukulukujen ominaisuuksiin palataan matemaattista laskentaa käsittelevässä luvussa.

2.7 Tehtäviä

2.7.1 Tee ohjelma, joka kysyy käyttäjältä nimeä ja tulostaa lyhyen tarinan, jossa päähenkilön nimi on korvattu annetulla nimellä.

2.7.2 Mitkä seuraavista muuttujista ovat sallittuja? Jos et tiedä, kokeile Python-tulkissa ja perustele, miksi asia on näin.

(a) `muut63`
(b) `Kokonaisluku`
(c) `5kulmio`
(d) `kissan_nimi`
(e) `luku(käyttäjänikä)`

2.7.3 Tee ohjelma, joka kysyy käyttäjältä merkkijonoa ja tulostaa sen pituuden.

2.7.4 Tee ohjelma, joka kysyy käyttäjän nimeä ja ikää ja tulostaa ne muodossa `Hei, [nimi], [ikä] vuotta!`

2.7.5 Katso jotakin esimerkeistä, jossa arvoja muutetaan toiseen tyyppiin. Minkälaisen virheen saat, kun poistat funktiot, jolla muunnos tehdään? (kuten `str` tai `int`)

2.7.6 Tee ohjelma, joka kysyy käyttäjältä kahta lukua `a` ja `b` ja tulostaa laskutoimitukset `a+b`, `a-b`, `a*c` ja `a/b`.

2.7.7 Kokeile, mitä seuraava ohjelma tekee:

```
1 tulostus = print
2 tulostus("Tulostetaan...")
```

Mitä tämä kertoo sinulle siitä, mitä `print` ja muut funktiot ylipäätään ovat?

Luku 3

Ehtolauseet

3.1 `if`-lause yksinkertaisimmillaan

Tähänastiset ohjelmamme ovat edenneet täysin suoraviivaisesti: Python-tulkki työskentelee rivi kerrallaan, kunnes saapuu tiedoston loppuun. `if`-lauseiden avulla ohjelma voi tehdä valintoja ja toimia eri tavalla riippuen vaikkapa käyttäjän syötteestä, päivämäärästä tai mistä tahansa ehdosta.

Seuraava esimerkki esittää ehtolauseen kaikista yksinkertaisimmillaan: yksi ainoa ehto, jonka täyttyessä tulostetaan tekstiä.

```
1 salasana = input("Syötä salasana: ")
2
3 if salasana == "correct horse battery staple":
4         print("Oikein!")
```

Ehtolause alkaa `if`-avainsanalla ja sen ehdolla. `==`-operaattori tarkastaa, ovatko sen oikea ja vasen puoli yhtäsuuret – lauseke `salasana == "..."` siis testaa, täsmääkö muuttuja `salasana` annettuun arvoon. Ehdon jälkeen seuraa kaksoispiste.

`if`-lausetta kirjoittaja ohjelmoija haluaa, että ehdon ollessa tosi ohjelma suorittaa joitakin koodirivejä. Tämä onnistuu siten, että ne kirjoittaa kaksoispisteen jälkeen sisennettynä eli jonkin määrän välilyöntejä tai tabulaattoreita jälkeen. Esimerkissä `if`-lauseen sisällä on yksi ainoa lause, mutta niitä voi olla useitakin, kunhan kaikki on sisennetty tismalleen samalla tavalla.

Tätäkin monimutkaisempia ehtolauseita voi rakentaa, mutta sitä ennen on tarpeen tutustua tarkemmin siihen, mitä ehdot oikeastaan ovat.

3.2 `bool`-tyyppi

Tarkemmin sanottuna `if`-lauseen ehtona on oltava lauseke, joka on tyypiltään totuusarvo eli `bool`. Totuusarvoja on kaksi: `True` eli tosi ja `False` eli

epätosi.

Aiemmin näimme `==`-operaattorin, joka palauttaa totuusarvon. `x == y` on `True`, jos `x` ja `y` ovat yhtä suuria, mutta `False`, jos ne eivät ole. Vertailla voi muillakin operaattoreilla; tässä ne taulukoituna.

Operaattori	**Merkitys**
`x == y`	Palauttaa, ovatko `x` ja `y` yhtä suuria
`x != y`	Palauttaa, ovatko `x` ja `y` eri suuria
`x > y`	Palauttaa, onko `x` suurempi kuin `y`
`x >= y`	Palauttaa, onko `x` suurempi tai yhtä suuri kuin `y`
`x < y`	Palauttaa, onko `x` pienempi kuin `y`
`x <= y`	Palauttaa, onko `x` pienempi tai yhtä suuri kuin `y`

Neljä viimeistä vertaavat merkkijonoja aakkosjärjestyksessä; esimerkiksi `"z"` on suurempi kuin `"a"`.

Vertailuoperaattorien avulla voi muodostaa yksinkertaisia ehtoja `if`-lauseisiin. Jos haluaa tarkistaa, onko käyttäjä täysi-ikäinen, voi kirjoittaa ehtolauseen `if ika >= 18: ...` Monimutkaisemmaksi kuitenkin menee. Jos on tarpeen tarkastaa useita ehtoja samaan aikaan, voi käyttää sisäkkäitä `if`-lauseita tällä tavoin:

```
if ika >= 18:
        if nimi == "Jussi":
                ...
```

Siistimpää on kuitenkin yhdistää ehdot. Jos `a` ja `b` ovat joitakin ehtolausekkeita, Python tarjoaa seuraavat operaattorit, joilla niistä voi muodostaa uusia ehtoja.

Operaattori	**Merkitys**
`a and b`	Tosi vain silloin, jos sekä `a` että `b` ovat tosia
`a or b`	Tosi, jos `a`, `b` tai molemmat ovat tosia
`not a`	Tosi, jos `a` on epätosi

Aiemmin esitetyn ehdon voi siis yhdistää `and`-operaattorilla.

```
if ika >= 18 and nimi == "Jussi":
        ...
```

On hyvin mahdollista kirjoittaa sama ehto monella eri tavalla. Jos haluaa muodostaa ehdon, joka on tosi vain silloin, kun muuttuja `x` ei ole arvoltaan `7`, voi kirjoittaa esimerkiksi `x != 7` tai `not x == 7`. Sellaista muotoa kannattaa suosia, jota on helpoin ymmärtää – Python ei siitä välitä, kuinka monimutkaisia tai yksinkertaisia ehtoja kirjoittaa, mutta koodia lukevat

muut ihmiset välittävät.

3.3 Monimutkaisempia `if`-lauseita

Aiemmin käsittelimme vain yksinkertaisimpia `if`-lauseita, joiden sisältämät lauseet joko suoritetaan tai ei suoriteta sen mukaan, onko ehto tosi vai ei. Entä jos haluammekin tehdä jotakin muuta siinä tapauksessa, että ehto on epätosi? Äsken näkemämme `not`-operaattorin avulla se on mahdollista.

```
1 if luku == 7:
2         print("Luku on 7!")
3
4 if not luku == 7:
5         print("Luku on jotain muuta!")
```

Itsensä toistaminen on kuitenkin pahasta. Ylläoleva ohjelma on varsinainen bugipesä – on helppoa kuvitella, että käy muokkaamassa toisen `if`-lauseen ehtoa, mutta unohtaakin muokata toista, minkä seurauksena jotakin odottamatonta tapahtuu. Onneksi asian voi tehdä siistimminkin.

Esimerkki 3.1: `else`-haara

```
1 if luku == 7:
2         print("Luku on 7!")
3 else:
4         print("Luku on jotain muuta!")
```

`if`-lauseeseensa voi esimerkin mukaisesti liittää `else`-haaran, joka suoritetaan silloin ja vain silloin, kun annettu ehto on epätosi.

Sisennyksellä on väliä: `else`-avainsanan on oltava samalla sisennystasolla (siis sitä ennen on oltava sama määrä välilyöntejä) kuin sen `if`-lauseen, johon se liittyy. `else`-lohkoon liittyvien lauseiden täytyy olla sisennetty enemmän kuin itse avainsanan. Kaksoispiste `else`-avainsanan jälkeen on myös muistettava; Python hälyttää syntaksivirheestä, jos mikään `if`-lauseen anatomiassa on pielessä.

Entä jos ei haluakaan suorittaa `else`-haaraa joka kerta? Voiko sillekin asettaa oman ehtonsa? Aina olisi mahdollista laittaa sisään toinen `if`-lause.

```
1 if nimi == "Mari":
2         print("Hei Mari!")
3 else:
4         if nimi == "Jasper":
5                 print("Terve Jasper!")
6         else:
7                 print("Tuntematon henkilö.")
```

Kuten arvata saattaa, siistimpi tapa on olemassa. `elif`-avainsanalla voi liittää `if`-lauseeseen lohkoja, jotka suoritetaan, jos jokin toinen ehto on tosi.

Esimerkki 3.2: Palaute arvosanasta

```
arvosana = int(input("Syötä arvosanasi: "))

if arvosana == 10:
        print("Loistavasti tehty.")
elif arvosana >= 8:
        print("Hyvin tehty.")
elif arvosana >= 5:
        print("Kohtuullisen hyvin tehty.")
elif arvosana == 4:
        print("Et päässyt läpi.")
else:
        print("Outo arvosana.")
```

Huomaa, että `if`-lauseesta suoritetaan vain ensimmäinen haara, jonka ehto on tosi. Arvosana 9 täyttää kyllä ehdon `arvosana >= 5`, mutta koska se täyttää ensin esiintyvän ehdon `arvosana >= 8`, ohjelma tulostaa `Hyvin tehty.`

Osaatko päätellä, mitä ohjelma tulostaa arvosanalla 6? Entä 4? Entä -2?

3.4 Tehtäviä

3.4.1 Tee ohjelma, joka kysyy käyttäjältä salasanaa ja tulostaa joko `Oikein!` tai `Väärin!` siitä riippuen, oliko salasana oikea.

3.4.2 Olkoon `a`, `b`, `c` ja `d` kokonaislukuja. Muodosta `if`-lauseet, jotka tarkastavat, ovatko seuraavat ehdot tosia.

(a) `a` on suurempi kuin `b`

(b) `c` on pienempi tai yhtä suuri kuin `d`

(c) `a` ei ole yhtä suuri kuin `b`

(d) `a` on suurempi kuin `b` ja `c` on suurempi kuin `d`

(e) `a` on `7` tai `b` on `3`

3.4.3 Tee ohjelma, joka kysyy kahta lukua ja kertoo, oliko ensimmäiseksi vai toiseksi syötetty suurempi.

3.4.4 **Karkausvuositarkistin.** Ohjelmointitehtävien klassikko; tee ohjelma, joka kysyy käyttäjältä vuosilukua ja kertoo, onko se karkausvuosi vai ei. Karkausvuosia ovat sellaiset vuodet, jotka ovat neljällä jaollisia, mutta jos vuosi on myös jaollinen sadalla, se on karkausvuosi vain,

jos se on jaollinen myös luvulla 400. Esimerkiksi 2014 ja 2400 ovat karkausvuosia, mutta 2007 ja 1900 eivät ole.

Jaollisuutta voit tarkastella `%`-operaattorilla, joka palauttaa jakojäännöksen; esimerkiksi `7 % 3` on `1`. Jos `x`:llä jaettaessa jakojäännös on `0`, luku on jaollinen `x`:llä.

3.4.5 Milloin seuraavat ehtolausekkeet ovat tosia? `x` ja `y` ovat kokonaislukuja.

(a) `x < 10 or y < 10`

(b) `x > y and x != 8`

(c) `not x != 0 and x == 0` (kuinka voisit ilmaista tämän yksinkertaisemmin?)

3.4.6 **Nelilaskuri.** Tee ohjelma, joka kysyy käyttäjältä kahta lukua ja laskutoimitusta ja tulostaa sitten laskun tuloksineen. Ohjelma voi toimia esimerkiksi näin:

```
Syötä ensimmäinen luku: 4
Syötä toinen luku: 5
Syötä laskutoimitus (+, -, *, /): +

4 + 5 = 9
```

Luku 4

Toisto

4.1 Toisto `while`-silmukalla

Nyt esiteltävä `while`-lause muistuttaa hyvinkin paljon `if`-lausetta, joten lukijan kannattaa kerrata edellisen luvun ehtolauseita käsittelevää osiota, jos tarvetta siihen on. Itse asiassa `while`-lause on syntaksiltaan lähes täydellinen `if`-lauseen kopio; ainut ero on se, että käytettävä avainsana on `while`, ei `if`. Samaan tapaan tarvitaan kaksoispiste ehdon jälkeen ja kasvava sisennys sisälohkolle.

Mitä eroja sitten on lauseiden semantiikassa? Jos `if`-lause suorittaa sisältämänsä lauseet siitä riippuen, onko sen ehto tosi, `while`-lause toistaa niitä niin kauan. Esimerkki auttanee.

Esimerkki 4.1: Lukujen iteroiminen `while`-silmukalla

```
1 x = 0
2 while x < 10:
3         print(x)
4         x = x + 1
```

Esimerkin ehto `x < 10` on tosi, jos `x` on alle 10. Silmukan sisällä rivillä 4 `x`:n arvoa kasvatetaan yhdellä, joten lopulta ehdosta tulee epätosi, kun `x` saa arvon 10, ja silmukasta poistutaan. `x`:n arvo tulostetaan silmukan kierroksella, joten ulostulo näyttää tältä:

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

Ehdoksi voi `while`-silmukalle antaa mitä tahansa. Yksi hyödyllinen erikoistapaus on ääretön silmukka, jonka avulla ohjelmansa toimintoa voi toistaa, kunnes käyttäjä sen sulkee. Sen voi muodostaa yksinkertaisesti antamalla `while`-lauseelle ehdoksi totuusarvon `True`.

```
while True:
        # varsinainen ohjelma
```

Ehto on aina tosi, joten ohjelma toistaa sisälohkoaan ikuisesti. On kuitenkin mahdollista myös poistua silmukasta `break`-avainsanan avulla.

Esimerkki 4.2: `break`-avainsana

```
while True:
        salasana = input("Syötä salasana: ")
        if salasana == "kissa":
                break

print("Pääsit sisään.")
```

Kun käyttäjä syöttää oikean salasanan, kaikessa yksinkertaisuudessaan `break` poistuu silmukasta, jolloin koodin suoritus jatkuu eteenpäin ja ruudulle tulostuu teksti `Pääsit sisään!`.

On hyvä huomata, että ohjelman voisi toteuttaa myös ilman `break`-ominaisuutta tarkistamalla `while`-silmukan ehdossa, milloin salasana on oikea. Periaatteessa `break` ei mahdollistakaan mitään uutta, mutta joissain tapauksissa sillä voi tulla siistimpää koodia. Omaa harkintaa kannattaa käyttää.

4.2 `for`-lause

Esimerkissä 4.1 nähtin yksi tapa käydä läpi kaikki tietyn lukuvälin luvut. Yksi siistin ja selkeän ohjelmoinnin säännöistä on se, että kuhunkin käyttötarkoitukseen käytetään siihen parhaiten sopivaa työkalua, ja sellainen lu-

kuvälin läpikäymiseen löytyy: `for`-lause. Havainnollistamisen vuoksi tässä aiempi esimerkki toteutettuna `for`-silmukalla.

Esimerkki 4.3: Lukujen iteroiminen `for`-silmukalla

```
1 for x in range(10):
2         print(x)
```

Koodirivien määrä puolittui – `for`:in käyttö selvästi kannattaa, mikäli välittää koodinsa siisteydestä. Sen toinen hyvä puoli on se, että kokeneelle Python-ohjelmoijalle `for` viestii välittömästi, mikä koodipätkän tarkoitus todennäköisesti on. `while`-lauseen käyttötarkoitukset ovat sen verran laajat, että koodia lukeva ei välttämättä heti ymmärrä, mikä on sen tarkoitus.

Kuinka `for`-silmukat siis toimivat? `for`-avainsanan jälkeen tulee muuttuja, johon silmukan nykyinen arvo sijoitetaan joka iteraatiolla. Sitten seuraa `in`-avainsana ja jokin lauseke, jota käydään läpi. Tässä luvussa käytetään vain `range`-funktiota, mutta myöhemmin opitaan muitakin tapoja käydä läpi erinäisiä asioita.

`range`-funktio palauttaa siis jonkin lukuvälin, jonka voi iteroida läpi `for`-lauseessa. Sitä voidaan käyttää useilla tavoilla.

Käyttötapa	**Merkitys**
`range(x)`	Luvut välillä $[0, x-1]$
`range(a, b)`	Luvut välillä $[a, b-1]$
`range(a, b, c)`	Joka `c`:s luku väliltä $[a, b-1]$

Kaikissa tapauksissa lukualueen jälkimmäinen raja ei sisälly siihen, mutta ensimmäinen sisältyy. `range(3)` on luvut 0, 1, ja 2; `range(4, 7)` on luvut 4, 5, 6. Aivan viimeisimmän tavan käyttää `range`-funktiota `c`-parametri voi olla vaikeaselkoinen, mutta esimerkki voi auttaa.

```
1 for i in range(0, 10, 3):
2         print(i)
```

Ohjelman ulostulo näyttää tältä:

```
0
3
6
9
```

4.3 Tehtäviä

4.3.1 Valitse jokin vanha tehtävä tai itse tekemäsi ohjelma, joka kysyy käyttäjältä syötettä ja tekee jotakin sen mukaisesti. Muokkaa ohjelma sellaiseksi, että se toistaa itseään ikuisesti `while`-silmukan avulla.

4.3.2 Kirjoita seuraava ohjelma uudestaan käyttäen `while`-silmukkaa.

```
for luku in range(0, 10, 2):
        print(luku)
```

4.3.3 Kirjoita seuraava ohjelma tiiviimpään ja siistimpään muotoon `for`-silmukan avulla.

```
luku = 0
while luku < 20:
        print("Luvun " + str(luku) + " neliö on " +
    ↪  str(luku * luku))
        luku = luku + 1
```

4.3.4 Tee ohjelma, joka kysyy käyttäjältä luvun ja toistaa niin monta kertaa viestin `Terve` *`n`*`. kerran!`, missä `n` kasvaa joka iteraatiolla.

4.3.5 Tee ohjelma, joka kysyy käyttäjältä luvun ja kertoo, monennella termillään summa $1^3 + 2^3 + 3^3 ...$ ylittää sen.

Sanasto

dokumentaatio Ohjelmiston tai ohjelmointikielen hakuteosta muistuttava asiakirja, joka kertoo yksityiskohtaisesti sen ominaisuuksista. Vaatii yleensä esitietoja. Pythonin dokumentaatio löytyy osoitteesta `https://docs.python.org/3/`. 2

funktio Ohjelmoinnissa sellainen arvo, jota voidaan kutsua antamalla sille nolla tai useampi argumenttia. Funktioilla voi olla paluuarvo, sivuvaikutuksia tai ei kumpaakaan. Esimerkiksi `print` on funktio, joka tulostaa sille annetun argumentin. 6

IDLE (Integrated Development and Learning Environment) Helppokäyttöinen ohjelma Python-koodin käsittelemiseen. 4

komentotulkki Interaktiivinen ohjelma, johon käyttäjä syöttää ohjelmakoodia, joka suoritetaan välittömästi. Python-komentotulkin saa auki `python`-komennolla; myös IDLE:ssä on komentotulkki. 4

koodinvaihtomerkki Koodinvaihtomerkki on jokin merkki (Pythonissa kenoviiva \), jonka avulla voidaan kirjoittaa merkkejä, jotka muuten tulkittaisiin virheellisesti. Esimerkiksi merkkijonojen sisällä lainausmerkin saa kirjoittamalla `\"`, sillä pelkkä lainausmerkki tulkittaisiin merkkijonon päättymiseksi. 7

lause Sellainen pätkä Python-koodia, joka voi esiintyä itsenäisesti. Esimerkiksi `print("kissa")` tai `x=6` ovat lauseita. 6

liukuluku Pythonin vastine desimaaliluvuille. Liukuluvuilla laskeminen on niiden sisäisestä esityksestä johtuen epätarkkaa. 14

merkkijono Merkeistä koostuva pätkä tekstiä. Pythonissa merkkijonoja voi merkitä asettamalla ne lainausmerkkien sisään: esimerkiksi `"kissa"` on merkkijono. 6

muuttuja Lauseke, joka viittaa sille aiempin määriteltyyn arvoon. Muuttujien nimillä on joitakin rajoituksia; `x3` ja `var` ovat sallittuja, mutta `muut)` ja `4y` ovat kiellettyjä. 10

semantiikka Termi sille, mikä merkitys koodin eri osilla on. Vertaa syntaksiin. 21, 26

sisennys Rivin alussa olevien välilyöntien tai tabulaattorien määrä. 16, 21

syntaksi Ohjelmointikielen kielioppi eli se, minkälaisia osia koodista on sallittua käyttää missäkin yhteydessä. Vertaa semantiikkaan. 21, 26

tyyppi Jokaisella arvolla on tyyppi, joka kertoo sen ominaisuudet, kuten sen, mitä operaattoreita ja metodeja sillä on. 11

versiohallintaohjelma Koodin säilyttämiseen ja historiatietojen kirjaamiseen suunniteltu ohjelmisto, joka helpottaa useamman ohjelmoijan yhteistyötä. Suosittuja ovat nykyisin mm. Git ja Mercurial. 8