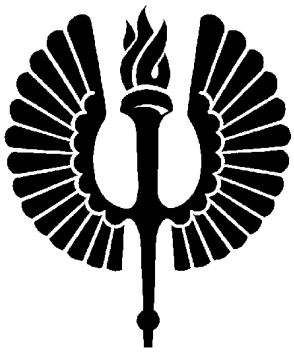


LUKIOKURSSI MATEMAATTISESTA OHJELMOINNISTA PYTHON-KIELELLÄ

Päivi Kulmala

Pro gradu -tutkielma
Joulukuu 2010

MATEMATIIKAN LAITOS
TURUN YLIOPISTO

TURUN YLIOPISTO
Matematiikan laitos

KULMALA, PÄIVI:
Lukiokurssi matemaattisesta ohjelmoinnista Python-kielellä
Pro gradu -tutkielma, 144 s.,40 liites.
Matematiikka
Joulukuu 2010

Tässä tutkielmassa suunnitellaan lukion kolmannen vuoden opiskelijoille tarkoitettu syventävä kurssi, jossa opiskellaan ohjelmoinnin perusteita ja matemaattista ohjelmointia Python-kielellä. Tutkielma on laadittu siten, että sitä voidaan käyttää kurssin oppimateriaalina.

Kurssin ensimmäiset oppitunnit käytetään ohjelmoinnin perustietojen opiskeluun. Tämän jälkeen kurssi painottuu algoritmisen ajattelun ja ohjelmointitaitojen kehittämiseen useita matemaattisia ohjelmia kirjoittamalla. Tutkielmaan on kerätty monipuolisesti lukion pitkän matematiikan oppimäärään sisältyviä menetelmiä ja tehtäviä, joita on kätevää ratkaista ohjelmoimalla. Näiden menetelmien ja tehtävien ohjelmointia havainnollistetaan tutkielmassa useiden esimerkkien avulla. Jokaisen matematiikan aihealueen yhteydessä on tarkoitus oppia jokin uusi asia ohjelmoinnista tai syventää jo opittujen asioiden osaamista. Tutkielma sisältää myös 50 harjoitustehtävää kurssin eri aihealueilta sekä lisämateriaalina lyhyen johdatuksen pelien ohjelmointiin ja täydennystä numeerisiin menetelmiin.

Tutkielmassa suunnitellulla kurssilla käytettäväksi ohjelmointikieleksi on valittu Python, sillä Pythonin yksinkertainen ja selkeä syntaksi helpottaa huomattavasti aloittelijoiden ohjelmoinnin oppimista. Lisäksi Python-tulkki on ladattavissa Internetistä ilmaiseksi.

Asiasanat: lukion pitkän matematiikan syventävä kurssi, matemaattinen ohjelmointi, ohjelmoinnin oppiminen, Python-ohjelmointikieli.

Sisältö

1 Johdanto

Tässä tutkielmassa suunnitellaan lukioon uusi syventävä kurssi, jossa opiskellaan ohjelmoinnin perusteita ja matemaattista ohjelmointia Python-kielellä. Kurssi kuuluu varsinaisesti pitkän matematiikan opintokokonaisuuteen mutta yhdistää sujuvasti pitkän matematiikan oppimäärän ja tietojenkäsittelytieteen. Kurssin päätavoitteet ovat algoritmisen ajattelutavan omaksuminen ja ohjelmoinnin perusasioiden oppiminen. Kurssi olisi hyvä suorittaa vasta lukion viimeisenä vuonna, kun kaikki muut pitkän matematiikan kurssit on suoritettu. Kurssilla ohjelmoidaan useita matemaattisia menetelmiä, ja näiden menetelmien ohjelmoiminen on mielekästä ja järkevää vasta, kun niihin liittyvä teoria on opiskeltu.

Ohjelmoinnin oppiminen jakautuu kurssilla kahteen päävaiheeseen. Ensin käytetään Pythonia laskimen tavoin sekä kirjoitetaan hyvin yksinkertaisia ja lyhyitä ohjelmia, jotta Pythonin syntaksi ja ohjelmoinnin ajatusmaailma tulisivat oppilaille tutuiksi. Tämän jälkeen kehitetään ohjelmointitaitoja edelleen kirjoittamalla useita eritasoisia matemaattisia ohjelmia. Ohjelmiin liittyvä matemaattinen teoria perustuu lukion pitkän matematiikan oppimäärään. Jokaisessa luvussa esitetään kyseisen matematiikan aihealueen teoria lyhyesti kertauksen vuoksi, mutta silti nämä aihealueet oletetaan tarkemmin tunnetuiksi aiempien matematiikan kurssien perusteella.

Tutkielma on jaettu vastaavasti kahteen päälukuun (luvut 3 ja 4) edellä mainittujen oppimisvaiheiden mukaan. Tutkielma on kirjoitettu siten, että sitä voidaan käyttää kurssin oppimateriaalina. Varsinaisissa luvuissa esitettävät ohjelmat on tarkoitettu uusien asioiden oppimista varten. Osa niistä voidaan kirjoittaa oppitunneilla opettajan johdolla, osa taas voidaan antaa oppilaille itsenäisesti mietittäviksi tuntitehtäviksi. Lukujen 3 ja 4 lopussa on harjoitustehtäviä, jotka on tarkoitettu oppilaille kotitehtäviksi. Harjoitustehtäviä voidaan tarvittaessa käyttää myös lisäharjoituksina oppitunneilla.

Kurssimateriaali on tarkoitus laittaa Internet-sivuille, josta löytyy myös sellaista tutkielmassa esitettävää lisämateriaalia, jota ei oppitunneilla ehditä käsittelemään. Tämä lisämateriaali on tarkoitettu asiasta kiinnostuneille ja edistyneille oppilaille ja sisältää täydennystä lukuun 4.7 Numeeriset mene-

telmät sekä lyhyen johdatuksen pelien ohjelmointiin testausesimerkkeineen. Myös tutkielman lopussa olevat liitteet on tarkoitus laittaa kurssin Internet-sivuille.

Ohjelmointikieleksi on valittu Python, sillä se soveltuu ominaisuuksiensa perusteella mainiosti ensimmäiseksi ohjelmointikieleksi. Pythonin yksinkertainen ja selkeä syntaksi helpottaa huomattavasti aloittelijoiden algoritmisen ajattelutavan ja ohjelmoinnin oppimista. Lisäksi ohjelmien kirjoittaminen Pythonilla on nopeaa ja virheiden etsiminen valmiista ohjelmasta vaivatonta. Python-tulkki on myös ladattavissa Internetistä täysin ilmaiseksi. [38]

Kurssi tarjoaa tietojenkäsittelytieteestä ja matematiikasta kiinnostuneille oppilaille mahdollisuuden ohjelmoinnin oppimiseen jo lukiossa. Tästä on varmasti paljon hyötyä useilla eri aloilla jatko-opinnoissa. Kun oppilas osaa ohjelmoinnin perusasiat Pythonilla, hänen on myöhemmissä opinnoissaan huomattavasti helpompi opiskella muita ohjelmointikieliä ja laajentaa osaamistaan tekemällä aiempaa vaativampia ohjelmointitehtäviä.

Luvuissa 3 ja 4 esitettävän oppimateriaalin Pythonin syntaksiin ja teoriaan liittyvät osuudet pohjautuvat lähteeseen [4] M. Dawson: *Python Programming for the Absolute Beginner*, ellei toisin mainita. Tähän lähteeseen en luvuissa 3 ja 4 viittaa erikseen. Lisäksi näiden lukujen sekä koko tutkielman kirjoittamisessa olen käyttänyt lähteinä lähdeluettelossa mainittuja matematiikan ja ohjelmoinnin oppikirjoja, Internet-artikkeleita ja Internet-sivustoja, joihin viittaan kyseisissä luvuissa tai kappaleissa lähdeluettelon numerointiin perustuen. Tutkielmassa esitettävät esimerkit olen suunnitellut itse lähteiden [4, 12, 13] tietojen pohjalta, ellei toisin mainita. Kurssin harjoitustehtävät olen laatinut itse, ellei viittauksia ole mainittu.

2 Ohjelmointi

2.1 Mitä ohjelmointi on?

Ohjelmointi on toimintaohjeiden antamista jonkin asian suorittamista varten. Tietokoneelle voi esimerkiksi antaa ohjeet ja käskyn laskea luvun seitsemän kertotaulu, ja kone tekee sen. Yleensä ohjelmointi yhdistetään juuri tietokoneisiin, mutta ympärillämme on joka päivä myös muita asioita, jotka voidaan ymmärtää ohjelmoinniksi. Arkipäivän ohjelmointia voisi olla vaikka kirjahyllyn kokoaminen kokoamisohjeisiin perustuen tai kakun leipominen reseptin avulla. Myös esimerkiksi mikroaaltouunin käyttö on tavallaan alkeellista ohjelmointia. Siinä uunille annetaan ohjeet siitä, millä teholla ja kuinka kauan sen pitäisi toimia. [16]

Ohjelmointi on systemaattista toimintaa, joka vaatii ajattelua [14]. Ohjelmia kirjoitettaessa on ensin suunniteltava tarkkaan, mitä ohjelman halutaan tekevän. Sitten on ratkaistava ongelma ja ilmaistava se koneelle niin, että kone pystyy sen ymmärtämään. [45]

2.2 Ohjelmoinnin historiallista taustaa

Ohjelmoinnin kehitys voidaan jakaa viiteen eri sukupolveen [23]. Ohjelmointia oli olemassa jonkin verran jo ennen ensimmäistä sukupolveakin. Englantilainen Ada Lovelace suoritti 1800-luvun alkupuolella ensimmäisen algoritmin mekaanisella, analyyttiseksi moottoriksi kutsutulla tietokoneella. Tämän vuoksi häntä kutsutaan maailman ensimmäiseksi tietokoneohjelmoijaksi. [46]

Ohjelmoinnin ensimmäinen varsinainen sukupolvi ajoittuu 1940-luvulle, jolloin ensimmäinen elektroninen tietokone Eniac valmistui (1946). Ensimmäisen sukupolven ohjelmointikieli oli konekieltä eli käytännössä lukuja 0 ja 1 peräkkäin. Tällainen ohjelmointi oli erittäin hankalaa. [18, 23]

Toinen sukupolvi (1950-luvun alkupuoli) ei vielä edusta kovin paljon kehittyneempää ohjelmointia kuin ensimmäinenkään sukupolvi. Konekielen käytöstä pyrittiin luopumaan antamalla numeerisille käskyille kuvaavia nimiä. Ohjelmoijat käyttivät usein tätä merkintätapaa suunnitellessaan ohjelmaa paperille ja käänsivät sen myöhemmin konekieliseen asuun. Pian huo-

mattiin, että myös käännösprosessin voisi antaa tietokoneen tehtäväksi. Syntyi Assembler-kääntäjä ja assembly-kieli. [9, 18]

Kolmannen sukupolven ohjelmointikielet ovat korkeammalla tasolla assembly-kieliin verrattuna. Niiden varaan on helppoa rakentaa ohjelmia ja useimmmat niistä ovat laitteesta riippumatomia. Tunnettuja esimerkkejä kolmannen sukupolven kielistä ovat Fortran, Cobol, Basic, Pascal ja C. Kolmannen sukupolven kielet käyttävät kääntäjää, joka siis kääntää ohjelmakoodin konekieliseen asuun. Myös tulkin käyttöä alkoi esiintyä kolmannen sukupolven kielten yhteydessä. Tulkki eroaa kääntäjästä siten, että se toteuttaa ohjeita sitä mukaa, kun niitä käännetään eikä käännä koko koodia kerralla. [9]

Myös nykyään paljon käytetyt Python ja Java kuuluvat kolmannen sukupolven ohjelmointikieliin. Kuitenkin kehitys on edennyt niin paljon, että voidaan sanoa neljännen ja viidennen sukupolvenkin kieliä olevan olemassa. Neljännen sukupolven kielet (muun muassa tietokantakieli SQL) ovat syntaksiltaan hieman lähempänä luonnollista kieltä kuin kolmannen sukupolven kielet, mutta raja näiden kahden sukupolven välillä on hyvin epätarkka. Viidennen sukupolven kielet liittyvät visuaalisen tai graafisen käyttöliittymän avulla tehtävään ohjelmointiin. [18, 23]

2.3 Miksi opettelisin ohjelmoimaan?

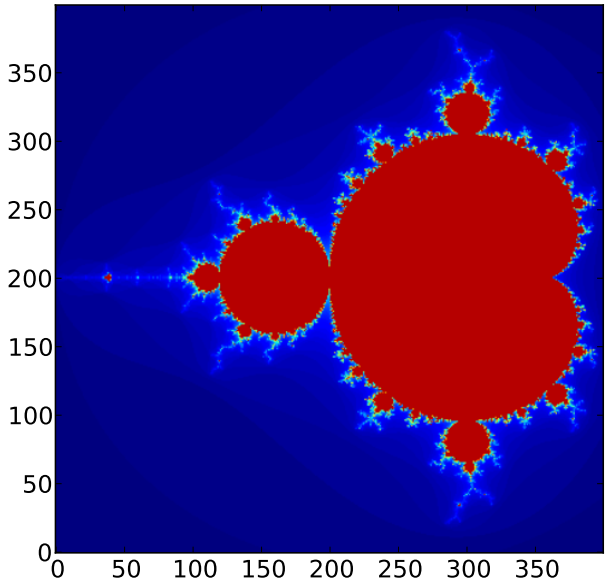
Seuraavassa listassa on lueteltu muutamia syitä, miksi ohjelmoinnin opetteleminen kannattaa.

- Kenties ohjelmointitaidosta on hyötyä tulevassa työelämässä tai jatko-opinnoissa.
- Kuten tällä kurssilla huomataan, ohjelmoimalla on äärimmäisen kätevää ratkaista monia matemaattisia ongelmia.
- Kaikki tietokonepelit on toteutettu ohjelmoimalla. Taitojen kehittyessä jokainen voi tehdä myös omia pelejä.
- Algoritmisen ajattelun oppiminen on hyödyllistä muutenkin kuin vain ohjelmoinnin kannalta.

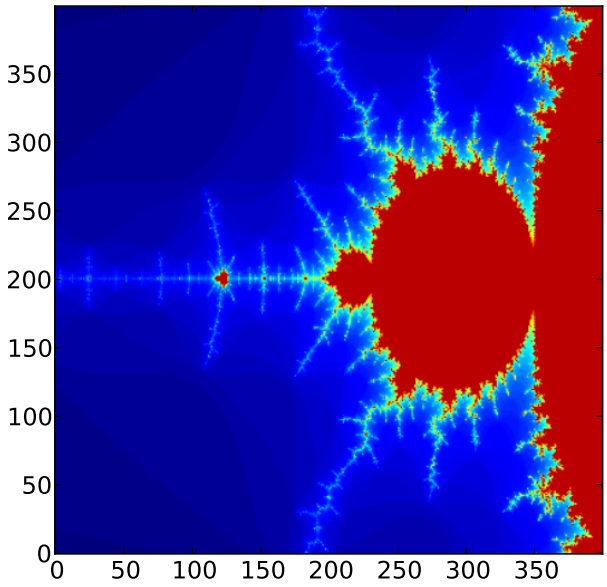
- Yhdysvaltalainen alansa pioneeri Kirby Urner edistää Python-ohjelmoinnin käyttöä kouluopetuksessa. Syy tähän on: "...to let students to experiment and experience math instead of the usual classroom theoretical approach". [28]

- Ohjelmoimalla voi toteuttaa hyvin monipuolista grafiikkaa, muun muassa seuraavanlaisen fraktaalin (Kuvat 1 ja 2). Fraktaalien ohjelmakoodi löytyy liitteestä Kuvaajien lähdekoodeja, sillä fraktaalin määrittelyn ja sen ohjelmoimisen vaikeustaso ylittää tämän kurssin vaatimustason. [6]

Kuva 1: Yksi tunnetuimmista fraktaaleista: Mandelbrotin joukko



Kuva 2: Mandelbrotin joukko: suurennos edellisen kuvan vasemmalla olevasta alueesta



2.4 Millainen on hyvä ohjelmoija?

Ennen ohjelmoinnin opettelun aloittamista on hyvä vielä tutustua lyhyesti muutamiin hyvän ohjelmoijan ominaisuuksiin. Microsoftin Academic Developer Evangelistin Jukka Wallasvaaran mukaan hyvä ohjelmoija on:

- Asioista perillä oleva: Tietää, mitä tekee ja miten tekee.
- Luova: Pystyy löytämään uusia tapoja tehdä asioita.
- Yhteistyökykyinen: Pystyy toimimaan yhteistyössä muiden ohjelmoijien kanssa.
- Sinnikäs: Työskentelee siihen asti, kunnes ongelma on ratkaistu. [45]

3 Johdatus Python-ohjelmointiin

Kurssin osallistujilla ei tarvitse olla ennen kurssin alkua minkäänlaista ohjelmointikokemusta, sillä ohjelmoinnin opettaminen aloitetaan aivan perusasioista. Ensimmäisellä tunnilla opetellaan yksityiskohtaisesti, miten Python-tulkki saadaan käyttöön. Pythonia opetellaan aluksi käyttämään laskimen tavoin erilaisia peruslaskutoimituksia harjoitellen. Tämän jälkeen siirrytään varsinaisiin ohjelmointitehtäviin. Tässä vaiheessa kirjoitetaan hyvin yksinkertaisia muutaman rivin ohjelmia ja opetellaan ohjelmien testaamista.

3.1 Lataaminen

Python-tulkin voi ladata Internetistä osoitteesta

- http://www.python.org/download

Tällä kurssilla käytetään Pythonista versiota python-2.6.5.amd64. Lisäksi kurssilla tarvitaan Pythonin laajennuspaketit Numerical Python ja Scientific Python, jotka tarjoavat useita valmiita moduuleja tieteellistä ja numeerista laskentaa varten sekä Matplotlib, jonka avulla voidaan piirtää esimerkiksi funktioiden kuvaajia. Nämä paketit ovat ladattavissa vastaavassa järjestyksessä Internetistä osoitteista

- http://sourceforge.net/projects/numpy/files
- http://sourceforge.net/projects/scipy/files
- http://matplotlib.sourceforge.net

64-bittisille käyttöjärjestelmille on myös tarjolla useita versioita näistä osoitteessa

- http://www.lfd.uci.edu/~gohlke/pythonlibs

Numerical Pythonista käytetään tällä kurssilla versiota numpy-1.4.1.win-amd64-py2.6-mkl, Scientific Pythonista versiota scipy-0.8.0.win-amd64-py2.6 ja Matplotlibista versiota matplotlib-1.0.0.win-amd64-py2.6. Yksityiskohtaiset lataus- ja asennusohjeet löytyvät työn lopussa olevasta liitteestä.

Toinen vaihtoehto olisi käyttää Pythonin laajennuspakettia Enthought Python Distribution. Paketista on mahdollista ladata ilmainen versio 32-bittisille käyttöjärjestelmille osoitteesta

- http://www.python.org/download

Se sisältää Python-tulkin lisäksi muun muassa juuri Numerical Pythonin, Scientific Pythonin ja Matplotlibin.

3.2 Peruslaskutoimituksia

Nopein tapa suorittaa peruslaskutoimituksia Pythonilla on kirjoittaa laskutehtävät Python-tulkkiin sisältyvän ohjelmointiympäristö IDLE:n interaktiiviseen tilaan tai vaihtoehtoisesti avata Python-tulkki komentorivillä ja kirjoittaa laskutehtävät sinne.

Esimerkiksi Windows 7 -käyttöjärjestelmässä ohjelmointiympäristö IDLE:n interaktiivinen tila löytyy Käynnistä-valikosta seuraavan polun avulla: Kaikki ohjelmat $\rightarrow$ Python2.6 $\rightarrow$ IDLE (Python GUI). Komentorivillä Python-tulkki voidaan avata kahdella tavalla. Ensimmäisen tavan mukaan avataan ensin komentorivi valitsemalla Käynnistä-valikosta Kaikki ohjelmat $\rightarrow$ Apuohjelmat $\rightarrow$ Komentokehote. Kirjoitetaan komentoriville sana python ja painetaan Enter-näppäintä. Toinen tapa on valita Käynnistä-valikosta Kaikki ohjelmat $\rightarrow$ Python 2.6 $\rightarrow$ Python (command line). Tällöin Python-tulkki käynnistyy automaattisesti, eikä sanaa python tarvitse enää kirjoittaa komentoriville erikseen. IDLE:n interaktiiviseen tilaan tai komentoriville on nyt mahdollista kirjoittaa laskutoimituksia tai mitä tahansa ohjelmia ja suorittaa ne painamalla Enter-näppäintä. Python-istunto lopetetaan komentorivillä painamalla ensin samanaikaisesti näppäimiä Ctrl ja Z ja sitten näppäintä Enter.

Yhteen-, vähennys- ja kertolaskutehtävät voidaan suorittaa Pythonilla aivan kuten tavallisella laskimella käyttäen operaattoreita $+$, $-$ ja $*$. Jos laskutehtävät kirjoitetaan IDLE:n interaktiiviseen tilaan tai komentoriville, näkymä laskutehtävien suorittamisen jälkeen on seuraavanlainen.

```
>>> 1+1
2
>>> 100-1
99
>>> 2*(10+5)
30
```

Ennen jakolaskutehtävien kirjoittamista on huomioitava, että Pythonissa on olemassa useita erityyppisiä lukuja. Niistä tärkeimpiä ja useimmin käytettyjä ovat kokonaisluvut ja desimaaliluvut. Kokonaislukuja ovat esimerkiksi 3290, 0 ja -29 eli luvut, joissa ei ole desimaaliosaa. Desimaalilukuja ovat esimerkiksi $4,334$, $-0,39$ ja $2,0$ eli luvut, joissa on desimaaliosa. Kirjoitetaan tästä esimerkkinä jakolaskutehtävä $33/5$ kahdella tavalla.

```
>>> 33/5
6
>>> 33.0/5
6.5999999999999996
```

Ensimmäisessä laskutoimituksessa Python ymmärtää luvun 33 kokonaisluvuksi ja antaa siten vastaukseksi myös kokonaisluvun, joka on tietysti virheellinen, ellei jako mene tasan. Jälkimmäisessä laskutoimituksessa luku $33,0$ on desimaaliluku, joten Python antaa myös vastaukseksi desimaaliluvun.

Potenssiinkorotus merkitään Pythonilla käyttäen operaattoria $**$ [31]. Esimerkiksi 8^2 lasketaan seuraavasti:

```
>>> 8**2
64
```

Pythonilla on helppo laskea erilaisia jakojäännöksiä. Kongruenssiyhtälön $a \equiv x (mod\ b)$ ratkaisu x saadaan operaattorin `%` avulla kirjoittamalla `a%b`. Samoin saadaan ratkaistua esimerkiksi $98020 \equiv x\ (mod\ 10)$ ja $98020 \equiv x\ (mod\ 387)$.

```
>>> 98020%10
0
>>> 98020%387
109
```

Itseisarvo lasketaan käskyllä `abs()` [30].

```
>>> abs(-2)
2
```

3.3 Ensimmäinen ohjelma

Kun peruslaskutoimituksia on harjoiteltu tarpeeksi, voidaan kirjoittaa ensimmäinen varsinainen Python-ohjelma. Kirjoitetaan ohjelma suoraan IDLE:n interaktiiviseen tilaan tai komentoriville. Ohjelma tulostaa `print`-käskyn jälkeisen lainausmerkkien sisään kirjoitetun tekstin. Lainausmerkkien jälkeinen merkintä on kommentti. Pythonissa kommentit kirjoitetaan aina #-merkin perään. Tällöin tulkki lopettaa lukemisen #-merkkiin ja siirtyy lukemaan seuraavaa riviä. Ohjelmia on hyvä opetella kommentoimaan, sillä varsinkin pidemmissä ohjelmissa kommentit helpottavat huomattavasti ohjelmakoodin lukemista.

```
>>> print "Game over!"     #Ensimmainen ohjelma.
Game over!
```

3.4 Skriptien käyttö

Ensimmäinen Python-ohjelma on nyt kirjoitettu. Kaikki tällä kurssilla kirjoitettavat ohjelmat voitaisiin kirjoittaa edellä opitulla tavalla IDLE:n interaktiiviseen tilaan tai komentoriville. Näin ei kuitenkaan kurssilla tästä eteenpäin tehdä, sillä kaikki IDLE:n interaktiiviseen tilaan tai komentoriville kirjoitettu teksti häviää, kun Python-tulkki suljetaan [29].

Sen sijaan ohjelma voidaan kirjoittaa millä tahansa tekstieditorilla, tallentaa se Python-tiedostoon (tiedostonimen pääte .py) ja suorittaa se tästä

tiedostosta milloin tahansa tallennuksen jälkeen. Tällä kurssilla käytetään tekstieditorina Pythonin ohjelmointiympäristö IDLE:n skriptitilaa, joka avataan seuraavasti: avataan ensin IDLE:n interaktiivinen tila luvussa 3.2 opitulla tavalla ja valitaan sitten File → New Window. IDLE:n skriptitilaan kirjoitettu valmis ohjelmakoodi tallennetaan Python-tiedostoon esimerkiksi nimellä tiedostonimi.py. Vastaavasti ohjelmakoodit voitaisiin kirjoittaa millä tahansa muulla tekstieditorilla ja tallentaa ne Python-tiedostoon.

Tällaista tekstieditorilla kirjoitettua ja tiedostoon tallennettua Python-koodia, joka on myöhemmin tarkoitus suorittaa ohjelmana jonkin tietyn tehtävän ratkaisemista varten, kutsutaan skriptiksi [29, 33]. Skriptien käyttö mahdollistaa siis ohjelmakoodin tallentamisen, suorittamisen myöhemmin sekä muokkaamisen. Pythonissa skriptejä on mahdollista muokata myös istunnon aikana käyttäen `reload()`-käskyä [29]. Tämä käsky esitellään tarkemmin luvussa 3.5, sillä ennen sen käyttöä on opiskeltava perustiedot ohjelmien testaamisesta.

Skripti voi olla esimerkiksi tiedostoon tallennettu ja myöhemmin suoritettavissa oleva lyhyt Python-koodi, joka sisältää käskyt viiden luvun keskiarvon laskemiseksi (luvun 3 harjoitustehtävä 4). Skriptejä voidaan kirjoittaa myös pitkien tai monimutkaisten tehtävien suorittamista varten. Tällöin tiedostoon tallennettu skripti voi sisältää myöhemmin kurssilla opittavia funktioiden määrittelyjä sekä pääohjelman, joka kutsuu näitä funktioita [33]. Kun ohjelmakoodi on hyvin pitkä, se voidaan myös jakaa useiksi skripteiksi, joista kukin tallennetaan omaan tiedostoonsa ja tuodaan yhden skriptin (pääohjelman) käyttöön. Kerran kirjoitettua skriptiä voidaan hyödyntää myös muissa skripteissä osatoimintona tuomalla se näiden muiden skriptien käyttöön tiedostosta, johon se on tallennettu [29]. Kurssin kaikki ohjelmat kirjoitetaan tästä eteenpäin skripteinä, joten termeillä ohjelma ja ohjelmakoodi tarkoitetaan tästä eteenpäin skriptejä.

3.5 Ohjelmien testaaminen

Python-tiedostoihin tallennettuja ohjelmakoodeja voidaan testata seuraavilla tavoilla:

1. Jos kyseinen ohjelmakoodi on kirjoitettu IDLE:n skriptitilaan ja tallennettu Python-tiedostoon, joka on testaushetkellä auki, ohjelma voidaan suorittaa suoraan IDLE:ssä valitsemalla Run → Run Module.

2. Millä tahansa tekstieditorilla kirjoitettu ja Python-tiedostoon tallennettu ohjelma voidaan suorittaa komentorivillä seuraavasti: Avataan ensin komentorivi (Komentokehote) luvussa 3.2 opitulla tavalla. Kirjoitetaan komentoriville kirjainyhdistelmä cd ja sen perään hakemistopolku, jossa kyseinen tallennettu tiedosto sijaitsee. Painetaan Enter-näppäintä. Kirjoitetaan tämän jälkeen sanat python tiedostonimi.py ja painetaan jälleen Enter-näppäintä.

3. Ohjelman voi testata komentorivillä myös seuraavalla tavalla: Käynnistetään Python-tulkki komentorivillä jommalla kummalla luvussa 3.2 esitetyllä tavalla. Jos ohjelma, joka halutaan suorittaa, on tallennettu Python-tiedostoon nimellä tiedostonimi.py, kirjoitetaan komentoriville sanayhdistelmä import tiedostonimi ja painetaan Enter-näppäintä.

Kohdissa 1 ja 2 esitetyillä tavoilla suoritettaessa ohjelma tekee ne toiminnot, jotka kyseisessä Python-tiedostossa on määritelty. Ohjelma voi esimerkiksi tulostaa jonkin muuttujan arvon tai laskutoimituksen tuloksen tai suorittaa funktion, jota tiedostossa kutsutaan. Kohdassa 3 esitettyyn tapaan on kiinnitettävä huomiota siksi, että sillä voidaan testata myös sellaisia ohjelmia, joista on kirjoitettu Python-tiedostoon pelkästään jonkin toiminnon suorittava funktio mutta ei funktion kutsua. Tällöin kyseistä funktiota voidaan kutsua (mahdollisesti useita kertoja eri parametrien arvoilla) komentorivillä import-lauseen jälkeen.

Kuten luvussa 3.4 mainittiin, Pythonissa skripteinä kirjoitettuja ohjelmakoodeja voidaan muokata myös istunnon aikana. Tarkastellaan tätä seuraavan esimerkin avulla:

Kirjoitetaan tekstieditorilla seuraavalla sivulla esitetty funktio ja tallennetaan se tiedostoon testitiedosto.py. (Funktioiden käyttöä opiskellaan tarkemmin luvussa 4.3.2.)

```
#tiedostossa testitiedosto.py
def testifunktio():
    a = 1
    print a
```

Suoritetaan ohjelma kirjoittamalla Python-istunnossa komentoriville `import testitiedosto` ja kutsumalla tiedostossa määriteltyä funktiota lauseella `testitiedosto.testifunktio()`. Komentoriville tulostuu numero 1, kuten funktiossa määritellään.

```
>>> import testitiedosto
>>> testitiedosto.testifunktio()
1
```

Muokataan tekstieditorissa ohjelmakoodia siten, että vaihdetaan muuttujan `a` arvoksi `a = 2`. Kutsutaan uudelleen funktiota lauseella `testitiedosto.testifunktio()`. Voisi luulla, että komentoriville tulostuisi nyt funktion määrittelyn mukaan numero 2.

```
>>> testitiedosto.testifunktio()
1
```

Koska testitiedosto ladattiin käyttöön kerran istunnon alussa, editorissa tehty muuttujan `a` arvon muutos `a = 2` ei päivity istuntoon, vaan komentoriville tulostuu edelleen luku 1. Asia voidaan korjata lataamalla testitiedosto uudelleen käyttäen Pythonin sisäänrakennettua `reload()`-funktiota. Funktio saa argumenttina sen tiedoston nimen, joka halutaan ladata uudelleen [30]. Tässä tapauksessa kirjoitetaan komentoriville `reload(testitiedosto)`. Kutsutaan nyt testifunktiota uudelleen lauseella `testitiedosto.testifunktio()` ja huomataan, että komentoriville tulostuu luku 2.

```
>>> reload(testitiedosto)
<module 'testitiedosto' from 'testitiedosto.py'>
>>> testitiedosto.testifunktio()
2
```

Edellä esitetyssä tilanteessa tiedosto on ladattava uudelleen käyttäen nimenomaan `reload()`-funktiota eikä esimerkiksi `import`-lausetta.

Jos riville `reload(testitiedosto)` olisi edellä kirjoitettu uudelleen `import testitiedosto`, Python olisi ensin tarkistanut moduulirekisteristä, onko kyseinen tiedosto jo ladattu aikaisemmin saman istunnon aikana. Jos näin on, Python käyttää tätä aikaisemmin ladattua tiedostoa eikä uutta tiedostoa, johon on tehty muutoksia. [17]

3.6 Tyypit ja muuttujat

Muuttujat ovat välttämättömiä ohjelmoinnin näkökulmasta, sillä niihin voidaan varastoida tietoa ja muokata sitä. Muuttujia tarvitaan lähes kaikissa ohjelmissa myöhemmin tällä kurssilla. Siksi niihin on tutustuttava aivan kurssin alussa.

Harjoitellaan ensin luomaan muuttujia ja antamaan niille arvoja. Muuttujan voi nimetä haluamallaan nimellä, kunhan nimi sisältää vain kirjaimia, numeroita ja alaviivoja, eikä nimen ensimmäinen merkki ole numero. Tietenkin on hyvä kiinnittää huomiota siihen, ettei nimi ole kovin pitkä, mutta se on kuitenkin tarpeeksi kuvaava. Esimerkiksi muuttujalle, joka sisältää kolmion korkeuden, on parempi antaa nimeksi korkeus kuin pelkkä k.

Muuttujaan tallennetaan arvo käyttämällä yhtäsuuruusmerkkiä (=). On huomattava, että kaikki muuttujille annettavat arvot eivät ole samaa tyyppiä. Pythonissa on pelkästään luvuille neljä eri tyyppiä: int (kokonaisluvut), float (desimaaliluvut), long (pitkät kokonaisluvut) ja complex (kompleksiluvut). Lisäksi merkkijonot ovat vielä oma tyyppinsä, string. Python kuitenkin määrittää automaattisesti muuttujan tyypin, kun arvo tallennetaan muuttujaan.

Esimerkki 1.

```
#tiedostossa esimerkki1.py
#Annetaan muuttujille erityyppisia arvoja.
pituus = 173        #tyyppia int
paino = 100.8       #tyyppia float
nimi = "Pertti"     #tyyppia string

print nimi, "on", pituus, "cm pitka ja painaa", paino, "kg."
```

Kuten kommenttirivillä lukee, ohjelmakoodi on kirjoitettu Python-tiedostoon esimerkki1.py. Ohjelma voidaan nyt testata jollain luvussa 3.5 esitetyistä tavoista:

1. Ohjelma on kirjoitettu IDLE:n skriptitilassa. Jos tiedosto on IDLE:ssä auki, valitaan Run → Run Module.

2. Kirjoitetaan komentoriville kirjainyhdistelmä cd ja sen perään hakemistopolku, jossa kyseinen tallennettu tiedosto sijaitsee (tässä cd C:\Users\Päivi\Documents\gradu\python\graduun tulevat ohjelmat). Painetaan Enter-näppäintä. Kirjoitetaan tämän jälkeen sanat python esimerkki1.py ja painetaan Enter-näppäintä.

3. Käynnistetään Python-tulkki komentorivillä, kirjoitetaan sanat import esimerkki1 ja painetaan Enter-näppäintä.

Ruudulle tulostuu seuraava teksti:

```
>>>
Pertti on 173 cm pitka ja painaa 100.8 kg.
```

Jatkossa tällä kurssilla kaikkien ohjelmakoodien alkuun on merkitty kommenttina, minkä nimiseen tiedostoon kyseinen koodi on tallennettu. Jos ohjelman testaamisesta ei ole ohjelmakoodin jälkeen mainittu erikseen, merkki >>> kertoo, että sen jälkeinen osuus on ohjelman testaustulos. Testaaminen tapahtuu aina jollain edellä esitetyistä tavoista, vaikka sitä ei tästä eteenpäin enää erikseen mainita.

Muuttujien tyypeillä saattaa joissain tilanteissa olla suurikin merkitys. Tarkastellaan `raw_input()` -funktiota, jossa käyttäjä saa antaa syötteenä muuttujille arvot. Se on siis funktio, joka palauttaa käyttäjän antaman merkkijonon.

Esimerkki 2.

```
#tiedostossa esimerkki2.py
#Pertti menee pizzeriaan.
hinta1 = raw_input("Anna pizzan hinta euroissa: ")
hinta2 = raw_input("Anna kolan hinta euroissa: ")
```

```
print "Kokonaiskulut ovat", hinta1+hinta2," euroa."

>>>
Anna pizzan hinta euroissa: 10
Anna kolan hinta euroissa: 3
Kokonaiskulut ovat 103  euroa.
```

Tämän esimerkin avulla huomataan, mitä eroa on kokonaislukutyypillä ja merkkijonotyypillä. `raw_input()` -funktio palauttaa muuttujien `hinta1` ja `hinta2` arvoiksi merkkijonot 10 ja 3, ja kun kaksi merkkijonoa lasketaan yhteen, tuloksena saadaan merkkijonot peräkkäin tulostuneina eli 103. Tällä tavalla laskettuna Pertille tulee melko kallis pizzeriakäynti. Jos käyttäjän antama syöte on luku, kannattaa käyttää `input()`-funktiota, kuten esimerkissä 3 tehdään. Tämä funktio palauttaa automaattisesti oikean tyyppisen luvun, jolloin hintojen yhteenlasku toimii oikein.

Esimerkki 3.

```
#tiedostossa esimerkki3.py
hinta1 = input("Anna pizzan hinta euroissa: ")
hinta2 = input("Anna kolan hinta euroissa: ")

print "Kokonaiskulut ovat", hinta1+hinta2," euroa."

>>>
Anna pizzan hinta euroissa: 10
Anna kolan hinta euroissa: 3
Kokonaiskulut ovat 13  euroa.
```

3.7 Taulukot ja listat

Taulukot ja listat ovat sarjoja, joihin on mahdollista asettaa alkioiksi erityyppisiä arvoja, kuten lukuja, merkkijonoja tai jopa toisia taulukoita tai listoja.

Tyhjä taulukko nimetään ja luodaan käskyllä `taulukon_nimi = ()`. Taulukkoon voidaan lisätä alkioita kirjoittamalla alkiot sulkujen sisälle ja erottamalla ne pilkulla: `taulukon_nimi = ("alkio", "toinen", 3, "viimeinen")`. Taulukoilla on useita hyödyllisiä ominaisuuksia. Esimerkiksi taulukon pituus saadaan selville `len()`-funktiolla, ja `in`-operaattorin avulla voidaan tutkia, kuuluuko jokin alkio taulukkoon vai ei. Lisäksi taulukko on indeksoitu eli jokaiseen sen alkioon voidaan viitata tietyn indeksin avulla. Ensimmäisen alkion indeksi on aina 0.

Listoilla on kaikki samat ominaisuudet, jotka taulukoillakin on. Lisäksi listojen kokoa voidaan muokata, toisin kuin taulukoiden. Tyhjä lista luodaan käskyllä `listan_nimi = []` ja siihen voidaan lisätä alkioita samoin kuin taulukkoon.

Esimerkki 4. *Testataan edellä mainittuja ominaisuuksia listaan, joka sisältää alkioina B. Virtasen perheenjäsenet.*

```
#tiedostossa esimerkki4.py
b_virtasen_perhe = ["B", "Armi", "Marko", "Pasi-kissa"]
print b_virtasen_perhe
print "Perheessa on", len(b_virtasen_perhe), "jasenta."
print b_virtasen_perhe[1], "on B:n vaimo."

>>>
['B', 'Armi', 'Marko', 'Pasi-kissa']
Perheessa on 4 jasenta.
Armi on B:n vaimo.
```

On olemassa myös toinen tapa lisätä alkioita listaan, nimittäin `listan_nimi.append()`-käsky. Alustetaan ensin tyhjä lista normaalisti käskyllä `listan_nimi[]`. Listan ensimmäiselle paikalle saadaan lisättyä esimerkiksi alkio 9 käskyllä `listan_nimi.append(9)`. Tämän jälkeen voidaan lisätä esimerkiksi alkio 5 listan toiselle paikalle käskyllä `listan_nimi.append(5)` ja niin edelleen.

```
#tiedostossa listalisays.py
lista=[]
lista.append(9)
print lista
lista.append(5)
print lista
lista.append(1009)
print lista

>>>
[9]
[9, 5]
[9, 5, 1009]
```

Tutkitaan, mitä tapahtuu, kun luodaan kaksi listaa, joiden kaikki alkiot ovat numeroita ja yritetään laskea listat yhteen. Voisi olettaa, että yhteenlaskun `[0,1,2,3] + [0,4,0,6]` tuloksena saataisiin lista `[0,5,2,9]` eli kyseisten kahden listan numerot pareittain yhteenlaskettuna. Todellisuudessa näin ei kuitenkaan ole.

```
#tiedostossa listakatenointi.py
lista1 = [0,1,2,3]
lista2 = [0,4,0,6]
print lista1 + lista2

>>>
[0, 1, 2, 3, 0, 4, 0, 6]
```

Tätä toimintoa kutsutaan listojen katenoinniksi. Kahdesta tai useammasta listasta voidaan tehdä + -operaattorin avulla yksi pidempi lista, joka sisältää alkuperäisten listojen alkiot.

3.8 Ehto- ja toistorakenteet

Tarkastellaan seuraavaksi Pythonin ehto- ja toistorakenteita. Nämä rakenteet ovat merkittävä osa ohjelmointia, sillä niitä tarvitaan hyvin useissa ohjelmis-

sa. Tässä luvussa opetetaan vain ehto- ja toistorakenteiden perusteet, mutta harjoittelua jatketaan tämän työn myöhemmissä luvuissa muiden ohjelmien yhteydessä.

Ennen ehto- ja toistorakenteisiin siirtymistä on syytä kiinnittää huomiota Python-koodin sisennyksiin. Muista ohjelmointikielistä poiketen Pythonissa ohjelmakoodin lohkot erotetaan toisistaan sisennyksin eikä esimerkiksi aaltosulkein. Muissakin ohjelmointikielissä sisennykset parantavat koodin luettavuutta ja kuuluvat hyvään ohjelmointitapaan, mutta Pythonissa sisennyksiä on pakko käyttää, koska muuta tapaa lohkojen erottamiseksi ei ole.

Kuten esimerkissä 5 huomataan, if-lauseen jälkeinen osa ohjelmaa on sisennetty eli if-lause ja sen jälkeinen sisennetty osa koodia muodostavat lohkon. Myöhemmin opitaan, että myös esimerkiksi while- ja for-silmukoiden sekä funktioden sisään kuuluvat ohjelmakoodin osat on sisennettävä. Ohjelmakoodissa voi olla monentasoisia sisennyksiä eli esimerkiksi ehtolauseen sisällä voi olla vielä toinen ehtolause. Sisennysten taso kertoo lohkon suhteen toisiin lohkoihin. Sisennetyt osat ovat tavallaan alisteisia sisentämättömille tai vähemmän sisennetyille riveille. Sisennyksen jälkeen ensimmäinen rivi, joka ei ole sisennetty tai joka on vähemmän sisennetty kuin edellinen rivi, ei enää kuulu samaan lohkoon.

Yleisen tavan mukaan sisennyksen koko on yksi sarkaimen painallus tai neljä välilyöntiä. Sisennyksen koolla ei kuitenkaan ole ohjelman toimivuuden kannalta merkitystä, jos se on koko ohjelmakoodissa vakio. Sarkaimen painalluksia ja välilyöntejä ei myöskään koskaan tulisi sekoittaa, vaikka yksi sarkaimen painallus vastaakin neljää välilyöntiä useimmissa tekstieditoreissa.

3.8.1 Ehtorakenteet

Pythonissa ehtorakenteita voidaan toteuttaa if-lauseilla. Yksinkertaisimmillaan rakenne koostuu if-sanan jälkeisestä ehtolauseesta sekä lauseesta, joka joko suoritetaan tai jätetään suorittamatta sen mukaan, onko ehtolause tosi vai epätosi.

Esimerkki 5. *Kirjoitetaan ohjelma, joka pyytää käyttäjää antamaan luvun ja tulostaa luvun, jos se on pienempi tai yhtäsuuri kuin kymmenen. Tässä*

ehtolause on `x <= 10.0`.

```
#tiedostossa esimerkki5.py
x = input("Anna luku: ")
if x <= 10.0:
    print x, " on pienempi tai yhtasuuri kuin 10.0."
```

Testataan ohjelmaa kahdella eri luvulla. Huomataan, että luku $4,08$ toteuttaa ehtolauseen ja määrätty teksti tulostuu ruudulle. Luku 17 ei toteuta ehtolausetta, joten mitään ei tapahdu.

```
>>>
Anna luku: 4.08
4.08  on pienempi tai yhtasuuri kuin 10.0.
>>>
Anna luku: 17
>>>
```

Käyttäjän kannalta olisi hyvä, että edellinen ohjelma tekisi jotain myös siinä tapauksessa, kun luku on suurempi kuin x. Näitä tilanteita varten if-rakenne voi sisältää vaihtoehto-osan, joka suoritetaan, ellei ehtolause ole voimassa.

Esimerkki 6. *Muokataan edellistä esimerkkiä lisäämällä siihen vaihtoehto-osa.*

```
#tiedostossa esimerkki6.py
x = input("Anna luku: ")
if x <= 10.0:
    print x, "on pienempi tai yhtasuuri kuin 10.0."
else:
    print x, "on suurempi kuin 10.0."
```

Testataan ohjelmaa kolmella eri luvulla.

```
>>>
Anna luku: -37219
-37219 on pienempi tai yhtasuuri kuin 10.0.
```

```
>>>
Anna luku: 10
10 on pienempi tai yhtasuuri kuin 10.0.
>>>
Anna luku: 10.00001
10.00001 on suurempi kuin 10.0.
```

Oikeastaan edellisessä ohjelmassa pitäisi olla kolmen vaihtoehdon monivalintatilanne: luku on pienempi kuin 10, luku on yhtäsuuri kuin 10 tai luku on suurempi kuin 10. Tämä onnistuu käyttäen if-elif-else -rakennetta.

Esimerkki 7. *Muokataan edellistä esimerkkiä lisäämällä siihen elif-lause.*

```
#tiedostossa esimerkki7.py
x = input("Anna luku: ")
if x < 10.0:
    print x, "on pienempi kuin 10.0."
elif x == 10:
    print "Luku on 10.0."
else:
    print x, "on suurempi kuin 10.0."
```

Testataan taas ohjelmaa kolmella eri luvulla.

```
>>>
Anna luku: 0
0 on pienempi kuin 10.0.
>>>
Anna luku: 2*5
Luku on 10.0.
>>>
Anna luku: 92.2-8.23
83.97 on suurempi kuin 10.0.
```

3.8.2 Toistorakenteet

Pythonissa on kaksi toistorakennetta, while ja for. While-silmukkaa käytetään, kun halutaan toistaa jotain lausetta niin kauan kuin tietty ehtolause

on voimassa. For-silmukassa on myös mukana ehtolause, mutta sitä käytetään yleensä silloin, kun käydään läpi esimerkiksi sarja ja siten tiedetään toistojen määrä jo ennakkoon.

While-silmukan yhteydessä on tärkeää oppia kasvattamaan muuttujaa. Seuraavassa ohjelmassa merkintä `i*=10.0` tarkoittaa samaa kuin merkintä `i=i*10.0`, eli silmukkaa läpi käytäessä jokaisella kierroksella muuttuja `i` kerrotaan kymmenellä. Vastaavalla tavalla merkitään myös esimerkiksi `i+=1`, `i-=100` ja `i/=2`. Muuttuja `i` on muistettava alustaa ennen while-silmukkaa, jotta kone ensinnäkin tietää, mennäänkö while-silmukkaan vai ei, ja jotta silmukassa olisi myöhemmin jotain, jota kasvattaa.

```
#tiedostossa toistowhile.py
i=1
while i<=1000.0:
    print 1.0/i
    i*=10.0
```

Testauksessa tulostuvat seuraavat luvut.

```
>>>
1.0
0.1
0.01
0.001
```

For-silmukan yhteydessä opitaan käyttämään `range()`-funktiota.

Esimerkki 8. *Testataan for-silmukkaa ja* `range()`*-funktiota eri parametrien arvoilla.*

```
#tiedostossa esimerkki8.py
for i in range(6):
    print i,"^2 =", i**2
print "----------------"
```

```
for i in range(0,12,2):
    print i
print "----------------"

for i in range(99,93,-1):
    print i+1
```

Ensimmäisessä for-silmukassa lausetta `print i,"^2 =", i**2` toistetaan niin kauan, kun i käy läpi luvut $[0, 1, 2, 3, 4, 5]$, sillä funktio `range(6)` palauttaa listan $[0, 1, 2, 3, 4, 5]$. Toisessa silmukassa funktio `range(0,12,2)` palauttaa listan $[0, 2, 4, 6, 8, 10]$ eli joka toisen luvun väliltä $[0, 11]$. Kolmannen silmukan `range(99,93,-1)`-funktio palauttaa listan lukuja takaperin laskettuna eli $[99, 98, 97, 96, 95, 94]$.

```
0 ^2 = 0
1 ^2 = 1
2 ^2 = 4
3 ^2 = 9
4 ^2 = 16
5 ^2 = 25
----------------
0
2
4
6
8
10
----------------
100
99
98
97
96
95
```

3.9 Valmiit moduulit

Pythonissa, kuten kaikissa muissakin ohjelmointikielissä, on sisäänrakennettuna laaja luokkakirjasto. Se sisältää lukuisan joukon valmiita funktioita erilaisten toimintojen tekemiseen. Suurin osa näistä funktioista on jaettu moduuleihin. Moduulit ovat tiedostoja, jotka sisältävät sellaista koodia, jota pystytään hyödyntämään muissa ohjelmissa. Yksi moduuli sisältää yleensä yhteen tiettyyn aiheeseen liittyviä funktioita ja mahdollisesti myös vakioita.

Yksi tärkeimmistä moduuleista on `math`, joka sisältää useita matemaattisia funktioita ja vakioita, esimerkiksi neliöjuuren, trigonometriset funktiot ja logaritmit sekä usein käytetyt vakiot π ja e [20]. Toinen tällä kurssilla käytettävä moduuli on `random`, joka sisältää satunnaislukujen generoimiseen tarkoitettuja funktioita. Tähän moduuliin palataan luvussa 4.5. Lisäksi Pythonissa on niin sanottuja sisäänrakennettuja funktioita, jotka eivät kuulu mihinkään moduuliin. Näistä muutamia, kuten `len()`, `range()` ja `raw_input()` on jo käytetty kurssin alkuosassa. Pythonin luokkakirjaston tarjonta löytyy kokonaisuudessaan Internet-osoitteesta http://docs.python.org/library/.

Tässä luvussa harjoitellaan muutamien `math`-moduulin funktioiden käyttämistä esimerkin 9 avulla. Jotta funktioita pystytään käyttämään, on `math`-moduuli ensin tuotava käyttöön käskyllä `import math`. Kommenteissa kerrotaan, mitä mikäkin funktio tekee. Logaritmifunktioon on kuitenkin kiinnitettävä huomiota erikseen. Kuten kommenttirivillä lukee, funktio `math.log(x)` laskee luonnollisen logaritmin luvusta x. Jos halutaan laskea logaritmi jonkin muun kantaluvun suhteen, on sulkuihin kirjoitettava myös haluttu kantaluku. Funktio `math.log(x,y)` laskee siis y-kantaisen logaritmin luvusta x. 10-kantainen logaritmi voitaisiin laskea käskyllä `math.log(x,10)`, mutta sille on lisäksi oma funktionsa, `math.log10(x)`, joka on edellistä merkintätapaa tarkempi. [20]

Esimerkki 9.

```
#tiedostossa esimerkki9.py
import math

print math.pi   #Piin likiarvo
```

```
print math.e    #Neperin luvun likiarvo
print "\n"      #tulostaa tyhjan rivin

print math.exp(10)      #e potenssiin 10
print math.log(1)       #luonnollinen logaritmi luvusta 1
print math.log(math.e)  #luonnollinen logaritmi luvusta e
print math.log(64,2)    #2-kantainen logaritmi luvusta 64
print math.log10(10000) #10-kantainen logaritmi luvusta 10000
print "\n"

print math.sqrt(25)           #neliojuuri
print math.sqrt(12190170)
print "\n"

print math.factorial(10)      #luvun 10 kertoma
print "\n"

print math.sin(0)             #sini
print math.cos(math.pi)       #kosini
print math.asin(0.5)          #arkussini
print "\n"

print math.degrees(math.pi/3)    #pii/3 radiaania asteiksi
print math.radians(180)          #180 astetta radiaaneiksi
```

Ohjelman testaus tuottaa seuravat tulokset.

```
>>>
3.14159265359
2.71828182846

22026.4657948
0.0
1.0
6.0
4.0
```

```
5.0
3491.44239534

3628800

0.0
-1.0
0.523598775598

60.0
3.14159265359
```

Tutustutaan vielä kahteen funktioon, jotka eivät välttämättä ole tähän mennessä tuttuja, mutta ovat hyödyllisiä ohjelmoinnin kannalta. Nämä funktiot ovat `math.ceil(x)` ja `math.floor(x)`. Funktio `math.ceil(x)` antaa pienimmän kokonaisluvun, joka on suurempi kuin x. Funktio `math.floor(x)` antaa suurimman kokonaisluvun, joka on pienempi kuin x. Molemmat palauttavat float-tyyppisen arvon. [20]

```
#tiedostossa kattolattiafunktiot.py
import math
print math.ceil(39.3927)
print math.floor(39.3927)

>>>
40.0
39.0
```

3.10 Harjoitustehtäviä

Kirjoita ohjelmat skripteinä eli toisin sanoen kirjoita ohjelmakoodit tekstieditorilla ja tallenna ne Python-tiedostoon nimellä luku3_harjoitus'tehtävänumero'.py (esimerkiksi luvun 3 harjoitustehtävän 2 tiedostonimi on luku3_harjoitus2.py). Suorita ohjelmat tästä tiedostosta.

1. Laske Pythonilla
   a)$2 \cdot 25 + 50$
   b)$10/3$
   c)3^6
   d)$(38909^2 + (766 \cdot 6))/5$
   e)$|5 \cdot 40 - 4 \cdot 60|$
   f)$(\frac{1}{2} + \frac{1}{4} + \frac{1}{6})/(\frac{1}{2} + \frac{5}{12})$.

2. Suorita ja tulosta Pythonilla seuraava laskutoimitus: Ota kolme ensimmäistä numeroa puhelinnumerostasi (esimerkiksi 324), kerro tämä luku luvulla 80 ja lisää siihen luku 1. Kerro saatu luku luvulla 250. Lisää saatuun lukuun neljä viimeistä numeroa puhelinnumerostasi (esimerkiksi 9187). Lisää nämä neljä numeroa uudestaan. Vähennä näin saadusta luvusta luku 250 ja jaa saatu luku kahdella. Mitä huomaat? (Suuntanumeroa ei käytetä tässä laskussa. Voidaksesi suorittaa tehtävän oikein puhelinnumerosi on oltava seitsemän numeron mittainen.) [3]

3. Kirjoita ohjelma, joka kysyy puhelinnumerosi, ottaa sen vastaan merkkijonona ja tulostaa jonkin lauseen, joka sisältää puhelinnumeron.

4. Kirjoita ohjelma, joka pyytää käyttäjältä viisi lukua ja ilmoittaa niiden keskiarvon.

5. Kirjoita ohjelma, joka laskee auton keskinopeuden käyttäjän antamien matkan ja ajan perusteella. Keskinopeus lasketaan kaavalla $v = \frac{s}{t}$, jossa s on matka ja t aika.

6. Alusta tyhjä lista ja nimeä se haluamallasi nimellä. Lisää listaan yksitellen kokonaisluvut yhdestä viiteen ja tulosta valmis lista. Tulosta myös listan pituus ja listan kolmas alkio.

7. Kirjoita ohjelma, joka pyytää käyttäjältä kolme lukua ja tulostaa niistä suurimman.

8. Kirjoita ohjelma, joka pyytää käyttäjältä kaksi kokonaislukua ja ilmoittaa, onko ensimmäinen luku jaollinen toisella. [14]

9. Kirjoita ohjelma, joka laskee käyttäjän antaman luvun kertotaulun, kun kertoja käy läpi luvut $[0, 10]$. Kirjoita ohjelma kahdella eri tavalla, ensin käyttäen while-lausetta ja sitten for-lausetta.

10. Kirjoita ohjelma, joka laskee lukujen $1 - 10$ kertotaulut, joissa kaikissa kertoja käy läpi luvut $[0, 10]$.

11. Laske Pythonilla
    a)ln(83)
    b)$\log_5(15625)$
    c)lg(1021021)
    d)e^e
    e)18!
    f)$\tan(\pi/4)$

12. Muuta $\pi/6$ radiaania asteiksi ja 135 astetta radiaaneiksi.

13. Kirjoita ohjelma, joka laskee ja tulostaa ympyrän pinta-alan, kun käyttäjältä kysytään ympyrän säde.

14. Kirjoita ohjelma, joka laskee ja tulostaa suorakulmaisen kolmion hypotenuusan pituuden, kun käyttäjältä kysytään kateettien pituudet. [14]

4 Matemaattista ohjelmointia

Jatketaan ohjelmoinnin perusteiden opettelemista. Jokaisessa luvussa joko opitaan uusi asia ohjelmoinnista tai kerrataan vanhan osaamista matemaattisia menetelmiä ja laskutoimituksia ohjelmoiden. Matemaattisia osa-alueita ei esitetä siinä järjestyksessä, jossa ne on opiskeltu varsinaisilla pitkän matematiikan kursseilla. Asiat käsitellään sellaisessa järjestyksessä, joka tukee ohjelmoinnin oppimista parhaiten. Esimerkiksi lukujonot käsitellään heti luvun 4 alussa, koska niistä saa kohtalaisen helppoja ohjelmointitehtäviä kurssin alkupuolelle. Geometria taas on luvun 4 viimeisenä, sillä sen avulla on luonnollista opettaa olio-ohjelmointia.

4.1 Lukujonot - Toistorakenteiden kertausta

Tässä luvussa ohjelmoidaan muutamia lukujonoja ja niiden summia sekä kerrataan samalla luvussa 3.7.2 opittujen toistorakenteiden käyttöä. Tässä luvussa esitettävät lukujonojen ja summien määritelmät perustuvat lähteeseen [12].

4.1.1 Lukujono

Lukujonoksi kutsutaan jonoa, jonka jäsenet ovat reaalilukuja. Joskus lukujonolle $a_1, a_2, ..., a_n$ on mahdollista ilmoittaa jonon yleinen termi indeksin n lausekkeena, jolloin kaikki jonon jäsenet saadaan laskettua tästä kaavasta.

Esimerkki 10. *Kirjoitetaan ohjelma, joka laskee ja tulostaa lukujonon* $a_n = 2^n$ $(n = 0, 1, ..., 10)$ *jäsenet.*

Ohjelmassa käytetään for-silmukkaa, koska toistojen määrä tiedetään etukäteen.

```
#tiedostossa esimerkki10.py
for n in range(11):
    print 2**n,

>>>
1 2 4 8 16 32 64 128 256 512 1024
```

Esimerkki 11. *Muokataan edellistä ohjelmaa siten, että se laskee kaikki lukujonon* $a_n = 2^n$ $(n = 0, 1, 2, 3...)$ *jäsenet, jotka ovat pienempiä kuin* 5000.

Nyt käytetään while-silmukkaa, sillä ei tiedetä, kuinka monta kertaa silmukka on käytävä läpi ennen kuin jäsenen 2^n arvo ylittää luvun 5000.

```
#tiedostossa esimerkki11.py
n = 0
while 2**n < 5000:
    print 2**n,
    n += 1


>>>
1 2 4 8 16 32 64 128 256 512 1024 2048 4096
```

4.1.2 Fibonaccin lukujono

Lukujono voidaan määritellä myös rekursiivisesti. Tällöin tiedetään yksi tai useampi jäsen jonon alusta ja kaava, jonka avulla jonon seuraavat jäsenet lasketaan edellisten jäsenten avulla. Yksi tunnetuimmista rekursiivisesti määritellyistä lukujonoista on Fibonaccin lukujono, jossa $a_1 = 1$, $a_2 = 1$ ja $a_{n+1} = a_n + a_{n-1}$.

Esimerkki 12. *Kirjoitetaan ohjelma, joka laskee käyttäjän ilmoittaman määrän Fibonaccin lukujonon jäseniä.*

```
#tiedostossa esimerkki12.py
#kaksi ensimmaista jasenta
a = 1
b = 1
#haluttu jasenien lukumaara
n = input("Anna jonon jasenien maara: ")
while n < 0 or n%1 != 0:
    n = input("Anna jonon jasenien maara: ")
for i in range(1,n+1):
    print a,
    c = a + b   #lasketaan seuraava termi
```

```
    a = b       #kasvatetaan muuttujia
    b = c


>>>
Anna jonon jasenien maara: 17
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

4.1.3 Aritmeettinen lukujono

Aritmeettinen lukujono voidaan määritellä sekä rekursiivisesti että yleisen kaavan avulla. Rekursiivisen määritelmän mukaan aritmeettisen lukujonon jäsenet saadaan kaavasta $a_{n+1} = a_n + d$, jossa d on jonon termien erotusluku eli etäsyys toisistaan.

Esimerkki 13. *Kirjoitetaan aritmeettisen lukujonon termejä laskeva ohjelma, jolle käyttäjä saa antaa jonon ensimmäisen termin ja termien erotusluvun sekä laskettavien termien lukumäärän.*

```
#tiedostossa esimerkki13.py
a = input("Anna jonon ensimmainen termi: ")
d = input("Anna termien erotusluku: ")
n = input("Kuinka monta termia haluat? ")
while n < 0 or n%1 != 0:
    n = input("Kuinka monta termia haluat? ")

for i in range(1,n+1):
    print a,
    b = a + d   #lasketaan seuraava termi
    a = b       #kasvatetaan muuttujaa

>>>
Anna jonon ensimmainen termi: 0
Anna termien erotusluku: 8
Kuinka monta termia haluat? 11
0 8 16 24 32 40 48 56 64 72 80
```

Yleisen kaavan avulla aritmeettisen jonon n:s jäsen a_n voidaan laskea seuraavasti: $a_n = a_1 + (n-1)d$.

Esimerkki 14. *Muokataan edellisen esimerkin for-silmukkaa siten, että se laskee jonon jäsenet nyt yleisen kaavan avulla. Ohjelmaa testatessa huomataan, että molemmilla kaavoilla laskettaessa saadaan sama lopputulos.*

```
#tiedostossa esimerkki14.py
a1 = input("Anna jonon ensimmainen termi: ")
d = input("Anna termien erotusluku: ")
n = input("Kuinka monta termia haluat? ")
while n < 0 or n%1 != 0:
    n = input("Kuinka monta termia haluat? ")

for i in range(1,n+1):
    print a1 + (i - 1)*d,

>>>
Anna jonon ensimmainen termi: 0
Anna termien erotusluku: 8
Kuinka monta termia haluat? 11
0 8 16 24 32 40 48 56 64 72 80
```

4.1.4 Aritmeettinen summa

Päättyvässä lukujonossa $(a_1, a_2, ..., a_n)$ on vain äärellisen monta jäsentä, joten ne voidaan laskea yhteen. Tällöin puhutaan lukujonon summasta $S_n = a_1 + a_2 + ... + a_n$. Jos jonossa olisi ääretön määrä jäseniä, yhteenlasku $a_1 + a_2 + ... + a_i + ...$ ei koskaan päättyisi. Tällaista päättymätöntä lukujonon jäsenten yhteenlaskua kutsutaan sarjaksi. Sarjan summalla tarkoitetaan raja-arvoa, jota edellinen yhteenlasku lähestyy, kun sarjan jäseniä lisätään rajattomasti. Päättyvän aritmeettisen lukujonon $(a_1, a_2, ..., a_n)$ termien summa on $S_n = \sum_{i=1}^{n} a_i = n \cdot \frac{a_1+a_n}{2}$ eli ensimmäisen ja viimeisen termin keskiarvo kerrottuna termien lukumäärällä.

Esimerkki 15. *Lasketaan yhteen tuhat ensimmästä positiivista kokonaislukua eli luvut yhdestä tuhanteen.*

Kyseessä on aritmeettinen summa, jonka ensimmäinen termi on $a_1 = 1$, viimeinen termi on $a_n = 1000$ ja termien lukumäärä on 1000.

```
#tiedostossa esimerkki15.py
a = 1
b = 1000
n = 1000
summa = n*(a+b)/2
print summa

>>>
500500
```

Esimerkki 16. *Kirjoitetaan ohjelma, joka laskee aritmeettisen summan käyttäjän ilmoittamalle lukujonolle. Käyttäjä siis antaa jonon ensimmäisen ja viimeisen termin sekä erotusluvun.*

Summan kaavaan tarvitaan kuitenkin termien lukumäärä n erotusluvun d sijaan. Se voidaan ratkaista jonon yleisen termin kaavasta $a_n = a_1 + (n-1)d$, josta saadaan termien lukumääräksi $n = \frac{a_n - a_1 + d}{d}$.

```
#tiedostossa esimerkki16.py
a = input("Anna ensimmainen termi: ")
b = input("Anna viimeinen termi: ")
d = input("Anna erotusluku: ")
termien_maara = (b-a+d)/d
summa = termien_maara*(a+b)/2
print summa

>>>
Anna ensimmainen termi: 999
Anna viimeinen termi: 0
Anna erotusluku: -3
166833
```

Esimerkki 17. *Lasketaan kolmellatoista jaollisten kolminumeroisten luonnollisten lukujen lukumäärä ja summa.*

```
#tiedostossa esimerkki17.py
import math
ensimmainen =  13 * math.ceil(100.0/13.0)
viimeinen = 13 * math.floor(999.0/13.0)
d = 13
print "Ensimmainen termi on",ensimmainen
print "Viimeinen termi on",viimeinen

termien_maara = (viimeinen-ensimmainen+d)/d
print "Jonossa on",termien_maara,"termia."

summa = termien_maara*(ensimmainen+viimeinen)/2
print "Jonon termien summa on",summa

>>>
Ensimmainen termi on 104.0
Viimeinen termi on 988.0
Jonossa on 69.0 termia.
Jonon termien summa on 37674.0
```

4.2 Vektorilaskentaa - Numerical Python

Tässä luvussa tutustutaan Pythonin Numerical Python -laajennukseen, joka tuo lisää työkaluja numeeriseen laskentaan. Tärkein näistä työkaluista on tällä kurssilla käytettävä niin sanottu vektorointi. Vektoroinnissa luodaan ensin lista, jonka alkioina on numeroita. Tämän jälkeen listalle voidaan tehdä mitä tahansa laskutoimituksia elementeittäin. Laskutoimitus tehdään siis erikseen jokaiselle listan alkiolle. Tuloksena saadaan lista, jonka alkiot on saatu alkuperäisen listan alkioista suorittamalla tietyt laskutoimitukset. [39]

Numerical Python ei sisälly Python-tulkkiin, vaan se on ladattava erikseen Internet-osoitteesta http://sourceforge.net/projects/numpy/files/. Python Enthought Distribution -pakettiin se kuitenkin sisältyy valmiiksi.

Tässä luvussa esitettävä vektorien laskutoimituksiin liittyvä teoria pohjautuu lähteeseen [13].

4.2.1 Vektorien peruslaskutoimituksia

Lukion oppikirjoissa vektorit esitetään muodossa $\bar{a} = x_1\bar{i} + y_1\bar{j} + z_1\bar{k}$. Pythonilla ohjelmoidessa ne on muutettava muotoon $\bar{a} = [x_1, y_1, z_1]$, sillä vektorit alustetaan Pythonissa käskyllä `a = array([x,y,z])` [39]. Python-tulkki itse ei tue tätä käskyä, vaan ennen sen käyttämistä on otettava Numerical Python -kirjasto käyttöön käskyllä `from numpy import *`.

Vektoreilla voidaan suorittaa useita laskutoimituksia, muun muassa yhteenlasku, reaaliluvulla kertominen ja vektorin pituuden laskeminen.

- Vektorien $\bar{a} = x_1\bar{i} + y_1\bar{j} + z_1\bar{k}$ ja $\bar{b} = x_2\bar{i} + y_2\bar{j} + z_2\bar{k}$ summa lasketaan kaavalla $\bar{a} + \bar{b} = (x_1 + x_2)\bar{i} + (y_1 + y_2)\bar{j} + (z_1 + z_2)\bar{k}$.
- Kun vektori kerrotaan reaaliluvulla, suoritetaan kertolasku erikseen vektorin jokaiselle komponentille.
- Vektorin $\bar{b} = x_2\bar{i} + y_2\bar{j} + z_2\bar{k}$ pituus $|\bar{b}|$ lasketaan kaavasta $|\bar{b}| = \sqrt{x_2^2 + y_2^2 + z_2^2}$, jossa neliöjuuri merkitään Pythonissa käskyllä `math.sqrt()`.

Esimerkki 18. *Toteutetaan edellä mainitut laskutoimitukset vektoreille* $\bar{a} = 3\bar{i} + 6\bar{j}$ *ja* $\bar{b} = -\bar{i} + 5\bar{j} - 2\bar{k}$.

```
#tiedostossa esimerkki18.py
from numpy import *
#a = 3i+6j
a = array([3,6,0])
print "a = ",a

#b = -i+5j-2k
b = array([-1,5,-2])
print "b = ",b

#vektoreiden yhteenlasku
print "a+b = ",a+b
```

```
#vektorin kertominen reaaliluvulla
print "10*a = ",10*a

#vektorin pituus
pituus = math.sqrt(b[0]**2+b[1]**2+b[2]**2),
print "Vektorin b pituus on ","%.3f"%pituus
```

Testataan ohjelma.

```
>>>
a =  [3 6 0]
b =  [-1  5 -2]
a+b =  [ 2 11 -2]
10*a =  [30 60  0]
Vektorin b pituus on 5.477
```

Esimerkin 18 ohjelmakoodissa on vektorin pituuden tulostavalla rivillä merkintä `"%.3f"%pituus`. Tässä on käytetty niin sanottua formatoitua tulostusta ja tulostettu vektorin pituus kolmen desimaalin tarkkuudella. Merkintä `"%.6f"%pituus` tarkoittaa vastaavasti tulostusta kuuden desimaalin tarkkuudella ja merkintä `"%.3e"%pituus` kymmenpotenssimuodossa esitettävää lukua kolmen desimaalin tarkkuudella. Vektorin pituuden eteen voisi tulostaa tyhjää tilaa merkitsemälllä esimerkiksi `"%10.3f"%pituus`. Tätä hyödynnetään myöhemmin kurssilla muun muassa tulostettaessa taulukoita. Merkitsemällä `"%3d"%pituus` saataisiin tulostettua vektorin pituus kokonaislukuna niin, että sen eteen jää tyhjää tilaa. [5]

Vaikka esimerkin 18 laskutoimitukset voidaan järkevästi suorittaa kaikille matemaattisille vektoreille, on huomattava, että Numerical Pythonin laskennassa käyttämät vektorit ovat silti eri asia kuin matemaattiset vektorit. Kuten luvun 4.2 alussa mainittiin, Numerical Pythonin käyttämät vektorit ovat vain listoja, joiden avulla sama operaatio pystytään suorittamaan yhtä aikaa kaikille listan alkioille. Niille tehtävät laskutoimitukset tarkoittavat vain elementeittäin suoritettavia listan alkioiden laskutoimituksia. Täten on mahdollista laskea esimerkiksi $\sin(\bar{a}) = [\sin(x_1), \sin(y_1), \sin(z_1)]$, vaikkei se olekaan järkevä laskutoimitus matemaattisille vektoreille. [39]

4.2.2 Vektorien pistetulo

Vektorien $\bar{a} \neq 0$ ja $\bar{b} \neq 0$ pistetulo määritellään kaavalla $\bar{a} \cdot \bar{b} = |\bar{a}| \left|\bar{b}\right| \cos\alpha$, missä $|\bar{a}|$ on vektorin $\bar{a}$ pituus, $\left|\bar{b}\right|$ on vektorin $\bar{b}$ pituus ja α on vektorien $\bar{a}$ ja $\bar{b}$ välinen kulma. Pistetulo $\bar{a} \cdot \bar{b} = 0$ tarkalleen silloin, kun $\bar{a} = 0$, $\bar{b} = 0$ tai $\cos\alpha = 0$ eli $\alpha = 90°$. Koska alussa määriteltiin, että $\bar{a} \neq 0$ ja $\bar{b} \neq 0$, niin pistetulo on tässä tapauksessa nolla tarkalleen silloin, kun $\cos\alpha = 0$ eli kun vektorit ovat kohtisuorassa toisiaan vastaan.

Esimerkki 19. *Kirjoitetaan ohjelma, joka tutkii, ovatko käyttäjän antamat kaksi vektoria kohtisuorassa toisiaan vastaan. Jos eivät ole, niin ohjelma antaa vektorien välisen kulman asteina.*

Vektorien välinen kulma α saadaan laskettua muuttamalla pistetulon kaava muotoon $\cos\alpha = \frac{\bar{a}\cdot\bar{b}}{|\bar{a}|\left|\bar{b}\right|}$. Kolmiulotteisten vektorien tapauksessa pistetulo voidaan kirjoittaa muodossa $\bar{a} \cdot \bar{b} = x_1x_2 + y_1y_2 + z_1z_2$.

```
#tiedostossa esimerkki19.py
from numpy import *

print """Ohjelma tutkii vektorien a = x1*i + y1*j + z1*k ja
b = x2*i + y2*j + z2*k kohtisuoruutta ja vektorien valisen
kulman suuruutta. \n"""

x1,y1,z1 = input("Anna x1, y1 ja z1 (pilkuilla erotettuna):")
x2,y2,z2 = input("Anna x2, y2 ja z2 (pilkuilla erotettuna):")

#alustetaan vektorit
a = array([x1,y1,z1])
b = array([x2,y2,z2])

#tutkitaan vektorien kohtisuoruutta
pistetulo = x1*x2 + y1*y2 + z1*z2
if pistetulo == 0:
    print "Vektorit ovat kohtisuorassa toisiaan vastaan."
```

```
else:
    print "Vektorit eivat ole kohtisuorassa."

    #lasketaan vektorien pituudet
    a_pituus = math.sqrt(x1**2 + y1**2 + z1**2)
    b_pituus = math.sqrt(x2**2 + y2**2 + z2**2)

    #lasketaan kulman alpha suuruus asteina
    cos_alpha = pistetulo / (a_pituus * b_pituus)
    alpha = degrees(math.acos(cos_alpha))
    print "Vektorien valinen kulma on","%.2f"%alpha,"astetta."
```

Testataan ohjelmaa kahdella eri vektoriparilla.

```
>>>
Ohjelma tutkii vektorien a = x1*i + y1*j + z1*k ja
b = x2*i + y2*j + z2*k kohtisuoruutta ja vektorien valisen
kulman suuruutta.

Anna x1, y1 ja z1 (pilkuilla erotettuna):3,0,9
Anna x2, y2 ja z2 (pilkuilla erotettuna):-6,0,2
Vektorit ovat kohtisuorassa toisiaan vastaan.



>>>
Ohjelma tutkii vektorien a = x1*i + y1*j + z1*k ja
b = x2*i + y2*j + z2*k kohtisuoruutta ja vektorien valisen
kulman suuruutta.

Anna x1, y1 ja z1 (pilkuilla erotettuna):12,-4,5
Anna x2, y2 ja z2 (pilkuilla erotettuna):-1,7,3
Vektorit eivat ole kohtisuorassa.
Vektorien valinen kulma on 103.84 astetta.
```

Tässä ohjelmassa vektorien $\bar{a}$ ja $\bar{b}$ pistetulo laskettiin kaavalla $\bar{a} \cdot \bar{b} = x_1x_2 + y_1y_2 + z_1z_2$. Pistetulo on mahdollista laskea myös Numerical Pythonin toiminnolla `dot(a,b)` [25]. Tämä on kannattavaa varsinkin silloin, kun

listan pituus on suurempi kuin kolme. Lasketaan vertailun vuoksi edellisessä esimerkissä käytettyjen vektoreiden pistetulot käyttäen `dot(a,b)` -toimintoa.

```
#tiedostossa pistetulodot.py
from numpy import *

#alustetaan vektorit
a1 = array([3,0,9])
b1 = array([-6,0,2])
print dot(a1,b1)

#alustetaan vektorit
a2 = array([12,-4,5])
b2 = array([-1,7,3])
print dot(a2,b2)

>>>
0
-25
```

4.3 Toisen asteen yhtälö - Funktioiden käyttö

Luvussa 4.3.1 ohjelmoidaan toisen asteen yhtälön ratkaisukaava käyttäen niitä taitoja, joita kurssilla ollaan tähän mennessä opittu. Luvussa 4.3.2 opitaan tekemään omia funktioita ja luvussa 4.3.3 ohjelmoidaan toisen asteen yhtälön ratkaisukaava uudestaan funktioiden avulla.

4.3.1 Toisen asteen yhtälön ratkaisukaava

Toisen asteen yhtälölle $ax^2 + bx + c = 0$ voidaan laskea ratkaisut kaavasta $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

Esimerkki 20. *Kirjoitetaan ohjelma, joka laskee toisen asteen yhtälön ratkaisut käyttäjän antamilla kertomilla a, b ja c.*

Ohjelma perustuu siihen, että diskriminantin $b^2 - 4ac$ arvosta riippuen yhtälöllä on joko yksi reaalijuuri, kaksi reaalijuurta tai ei yhtään reaalijuurta. Jos

toisen asteen termin kerroin a on nolla, yhtälö ei ole toisen asteen yhtälö eikä sitä voi ratkaisukaavalla ratkaista, sillä nimittäjä saisi kaavassa arvon nolla.

```
#tiedostossa esimerkki20.py
import math
print "Ohjelma ratkaisee muotoa ax^2 + bx + c = 0 olevan"
print "toisen asteen yhtalon."

a,b,c = input("Anna yhtalon kertoimet muodossa a,b,c: ")

if a != 0:
    diskriminantti = b**2 - 4*a*c
    if diskriminantti < 0:
        print "Yhtalolla ei ole reaalijuuria."
    elif diskriminantti == 0:
        x = -b / (2*a)
        print "Yhtalon ratkaisu on x =",x,"."
    else:
        x1 = (-b + math.sqrt(diskriminantti)) / (2*a)
        x2 = (-b - math.sqrt(diskriminantti)) / (2*a)
        print "Yhtalolla on kaksi ratkaisua, x1 =",x1
        print "ja x2 =",x2,"."
else:
    print "Yhtalo ei ole toista astetta."
```

Testataan ohjelmaa kolme kertaa eri kertoimien arvoilla.

```
>>>
Ohjelma ratkaisee muotoa ax^2 + bx + c = 0 olevan
toisen asteen yhtalon.
Anna yhtalon kertoimet muodossa a,b,c: -5,8,-1
Yhtalolla on kaksi ratkaisua, x1 = 0.136675041929
ja x2 = 1.46332495807.

>>>
Ohjelma ratkaisee muotoa ax^2 + bx + c = 0 olevan
```

```
toisen asteen yhtalon.
Anna yhtalon kertoimet muodossa a,b,c: 1,2,1
Yhtalon ratkaisu on x = -1 .

>>>
Ohjelma ratkaisee muotoa ax^2 + bx + c = 0 olevan
toisen asteen yhtalon.
Anna yhtalon kertoimet muodossa a,b,c: 3,3,3
Yhtalolla ei ole reaalijuuria.
```

Ohjelma toimii tällä tavalla kirjoitettuna täysin virheettömästi. Kuitenkin siinä on eräs heikkous. Sitä ei nimittäin pysty suoraan hyödyntämään missään muussa ohjelmassa, jossa tarvitaan toisen asteen yhtälön ratkaisua. Kirjoittamalla ratkaisukaava funktiona sitä voidaan hyödyntää myös pidemmissä ohjelmissa osatoimintona.

4.3.2 Johdatus funktioiden käyttöön

Kurssilla ollaan tähän mennessä käytetty jo muutamia Pythonin omia sisäänrakennettuja funktioita, kuten `len()` ja `range()` ja math-moduulin funktioita, kuten `math.sqrt()` ja `math.log()`. Tässä vaiheessa opetellaan kirjoittamaan aivan omia funktioita.

Funktio on käytännössä pala ohjelmakoodia, jota voidaan käyttää myöhemmin uudelleen. Funktioiden uudelleenkäytettävyyteen palataan tarkemmin muutamaa riviä alempana. Funktiolle on mahdollista antaa syötteinä arvoja, joita käyttäen se tekee sille määritellyt toimenpiteet ja joko palauttaa laskemansa tuloksen tai muuttaa saamaansa arvoa. Muutos voi olla myös tulostus ruudulle eli tulostusvirran muuttaminen. Yleensä funktioksi kutsutaan sellaista aliohjelmaa, joka palauttaa jonkin tuloksen. Pythonissa voidaan kutsua kaikkia aliohjelmia funktioiksi, sillä ne aliohjelmat, jotka eivät sisällä myöhemmin tässä luvussa opittavaa `return`-käskyä, palauttavat arvon `None`. [4, 35]

Suurin osa varsinkin yksinkertaisista ohjelmista on mahdollista kirjoittaa käyttämättä funktioita, kuten kurssilla on tähän asti tehty. Kuitenkin niiden

käyttöä on hyödyllistä harjoitella jo tässä vaiheessa, jotta se on hyvin hallinnassa siirryttäessä pidempiin ohjelmakoodeihin. Kuten myöhemmin kurssilla huomataan, tärkein funktioiden ominaisuus on juuri niiden uudelleenkäytettävyysarvo. Tätä voidaan hyödyntää kahdella tavalla. Kun funktio kerran määritellään ohjelmakoodin sisältävässä tiedostossa, sen laskema toiminto saadaan käyttöön useita kertoja vain kutsumalla funktiota myöhemmin samassa tiedostossa.

Vaihtoehtoisesti funktio voidaan myös tallentaa omaan tiedostoonsa (esimerkiksi `tiedostonimi.py`) ja ladata sieltä minkä tahansa muun ohjelman käyttöön kirjoittamalla kyseisen ohjelman alkuun käsky `from tiedostonimi import *`. Nyt ohjelmassa, jonka käyttöön kyseinen `tiedostonimi.py` on ladattu, voidaan kutsua kaikkia `tiedostonimi.py` -tiedostossa määriteltyjä funktioita samalla tavoin kuin niissäkin ohjelmissa, joissa funktiot on kirjoitettu samaan tiedostoon niitä tarvitsevan ohjelman kanssa. On kuitenkin huomattava, että jokaisen myöhemmin käyttöön ladattavan tiedoston tiedostotyyppi on oltava Python File (tiedostonimessä pääte .py) ja tiedoston täytyy sijaita samassa hakemistossa sen tiedoston kanssa, jonka käyttöön se aiotaan ladata. [9]

Tällä kurssilla funktioiden tallentamista omaan tiedostoonsa ja sieltä lataamista ei juuri hyödynnetä, sillä ohjelmakoodit pysyvät aloittelijalle yksinkertaisempina lukea ja ymmärtää, kun funktio on määritelty samassa tiedostossa, jossa sitä kutsutaan. Esimerkki funktioiden lataamisesta eri tiedostosta esitetään luvussa 4.3.3 Toisen asteen yhtälön ratkaisukaava funktiona. Kurssin lisämateriaalista löytyvässä luvussa 4.7.7 Virheet numeerisessa laskennassa tarkastellaan toisen asteen yhtälön ratkaisukaavan mahdollista virhettä tietyillä kertoimien arvoilla. Tässä vaiheessa on erittäin kätevää ladata aiemmin kirjoitettu `ratkaise_yhtalo` -funktion sisältävä tiedosto suoraan pääohjelman käyttöön. Toinen esimerkki vastaavasta asiasta voisi olla seuraava: Aikaisemmin on kirjoitettu nopanheittoa simuloiva funktio (luvun 4 harjoitustehtävä 21) ja myöhemmin halutaan ohjelmoida peli, jossa tarvitaan noppaa. Tällöin nopanheittoa simuloivan funktion tiedosto voidaan ladata pelin käyttöön eikä nopanheiton koodia tarvitse kirjoittaa enää uudelleen.

Funktio määritellään kirjoittamalla `def funktion_nimi():`. Tämä rivi

kertoo tietokoneelle, että rivin jälkeinen sisennetty osa koodia on funktio nimeltään `funktion_nimi`. Nyt funktiota voidaan kutsua omalla nimellään myöhemmin samassa tai jossain toisessa tiedostossa. Funktion nimen jälkeisiin sulkuihin kirjoitetaan tarvittaessa funktiolle annettavat syötteet.

Luodaan ensin hyvin yksinkertainen funktio, joka palauttaa tekstin "Game over again!". Kommentilla `#main` merkitty pääohjelma kutsuu funktiota ja tulostaa sen, mitä funktio palauttaa.

```
#tiedostossa funktio1.py
def ensimmainen_funktio():
    return "Game over again!"

#main
print ensimmainen_funktio()

>>>
Game over again!
```

Harjoitellaan seuraavaksi informaation vastaanottamista parametrin kautta. Parametri `a`, jota laskussa käytetään, on kirjoitettava funktion nimen jälkeisiin sulkuihin. Kun funktiota kutsutaan pääohjelmassa, kirjoitetaan sulkuihin haluttu parametrin arvo, tässä esimerkissä 3. Tällöin luku 3 on informaatio, jonka funktio ottaa vastaan parametrin `a` kautta.

```
#tiedostossa funktio2.py
def toinen_funktio(a):
    return a+2

#main
print "Kun a=3, a+2 =",toinen_funktio(3)

>>>
Kun a=3, a+2 = 5
```

Kun verrataan funktiota `toinen_funktio(a)` ja matematiikasta tuttua funktiota $f(x)$, voidaan huomata joitain yhtäläisyyksiä. Funktiossa $f(x)$ f on funktion nimi, kuten edellisen esimerkin `toinen_funktio`. Suluissa oleva x

on parametri, kuten myös a. Myös funktion arvon palauttaminen ja parametrin arvon antaminen tapahtuvat näillä funktioilla täysin vastaavasti.

```
#tiedostossa funktio3.py
def f(x):
    return x**2 + 5*x - 4

#main
print "Kun x=3, f(x) = x**2 + 5*x - 4 saa arvon",f(3)

>>>
Kun x=3, f(x) = x**2 + 5*x - 4 saa arvon 20
```

Tutkitaan, miten käyttäjä voi antaa funktiolle arvoja parametrien kautta. Tämä toimii aivan samoin kuin edellinen ohjelma, paitsi arvon 3 tilalla on nyt käyttäjän antama arvo. Tehdään lisäksi myös pääohjelmasta funktio, jota kutsutaan ohjelman lopussa.

```
#tiedostossa funktio4.py
def f(x):
    return x**2 + 5*x - 4

def main():
  print "Laske funktion f(x) = x**2 + 5*x - 4 arvo haluamallasi"
  print "x:n arvolla."
  arvo = input("Anna parametrin x arvo: ")
  print "Kun x=",arvo,", f(x) = x**2 + 5*x - 4 saa arvon",f(arvo)

main()

>>>
Laske funktion f(x) = x**2 + 5*x - 4 arvo haluamallasi
x:n arvolla.
Anna parametrin x arvo: -6
Kun x= -6 , f(x) = x**2 + 5*x - 4 saa arvon 2
```

Esimerkki 21. *Kirjoitetaan ohjelma, joka vertailee funktioiden* $f(x) = x^2$ *ja* $g(x) = 2^x$ *kasvunopeuksia.*

Määritellään ensin molemmat funktiot erikseen. Pääohjelmassa annetaan muuttujan saada kaikki kokonaislukuarvot väliltä $[0, 10]$ ja tulostetaan arvoparit.

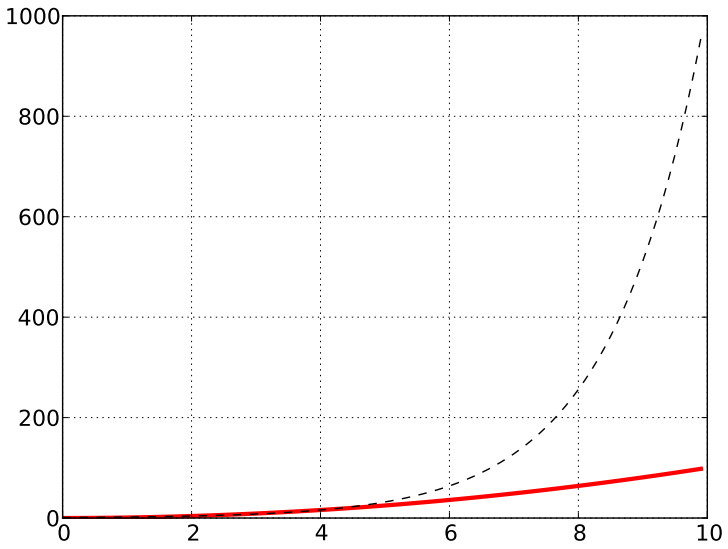
```
#tiedostossa esimerkki21.py
def f(x):
    return x**2

def g(x):
    return 2**x

def main():
    print "f(i)  g(i)"
    for i in range(0,11):
        print "%1d %5d" % (f(i), g(i))
main()
```

Testataan ohjelmaa ja huomataan, että muuttujan x kasvaessa funktion $g(x) = 2^x$ arvo kasvaa paljon nopeammin kuin funktion $f(x) = x^2$ arvo.

```
>>>
f(i)  g(i)
0     1
1     2
4     4
9     8
16    16
25    32
36    64
49    128
64    256
81    512
100   1024
```

Samaa asiaa voidaan havainnollistaa myös piirtämällä funktioiden $f(x)$ ja $g(x)$ kuvaajat samaan kuvaan (Kuva 3).

Kuva 3: Yhtenäinen viiva: $f(x) = x^2$, katkoviiva: $g(x) = 2^x$.



Tässä yhteydessä ei vielä esitetä funktion kuvaajan piirtämiseen vaadittavaa ohjelmakoodia, koska kuvaajien piirtäminen opetetaan vasta myöhemmin kurssilla. Koodi löytyy kuitenkin työn lopussa olevasta liitteestä Kuvaajien lähdekoodeja, joka tulee kurssin Internet-sivuille.

4.3.3 Toisen asteen yhtälön ratkaisukaava funktiona

Ohjelmoidaan toisen asteen yhtälön ratkaisukaava nyt siten, että kirjoitetaan `tayratkaisukaava.py` -nimiseen tiedostoon yhtälön ratkaiseva osa omana funktionaan, joka saa syötteenä yhtälön kertoimien a, b, ja c arvot. Tehdään pääohjelmasta eri tiedostoon toinen funktio, joka kysyy käyttäjältä kertoimien arvot ja kutsuu sen jälkeen yhtälön ratkaisevaa funktiota. Tätä jälkimmäistä tiedostoa voidaan käyttää niin sanottuna testitiedostona, jossa voidaan testata ratkaisukaavaa useilla eri kertoimien a, b ja c arvoilla kirjoittamatta aina koko ratkaisukaavaa uudestaan. Tämän ohjelman alkuun on muistettava kirjoittaa käsky `from tayratkaisukaava import *`, jotta yhtälön ratkaiseva funktio saadaan ohjelman käyttöön.

Jos koko ohjelma kirjoitettaisiin yhteen tiedostoon, tulisi pääohjelman

koodi tähän kyseiseen tiedostoon vain alkuosan koodin perään ilman riviä `from tayratkaisukaava import *`. Tässä tilanteessa pääohjelma kutsuisi yhtälön ratkaisevaa funktiota saman tiedoston alkuosasta eikä toisesta tiedostosta.

Tässä funktioille on määritelty alkuehto AE, loppuehto LE ja kommentti. Vaikka nämä rivit on merkitty kommenttimerkillä ja ne eivät siten vaikuta ohjelman kulkuun, hyvään ohjelmointitapaan kuuluu kirjoittaa alku- ja loppuehto sekä kommentti jokaiselle funktiolle. Kommentti kuvaa lyhyesti ja yksiselitteisesti, mitä funktio tekee. Näin ohjelman luettavuus paranee, ja muidenkin on helpompi lukea ohjelmia. Alkuehto kertoo, mitkä ehdot on oltava voimassa, jotta funktio toimii niin kuin on kuvattu. Alkuehto rajaa tehokkaasti pois ydinongelman kannalta merkityksettömät vaihtoehdot ja tekee ohjelmista yksinkertaisempia. Esimerkiksi funktiossa `ratkaisukaava(a,b,c)` alkuehto rajaa pois kertoimien a, b ja c imaginaariset arvot. Tietysti ohjelmaan voidaan lisätä ylimääräinen if-lause, joka tulostaa virheilmoituksen, jos käyttäjä yrittää syöttää kertoimille imaginaarisia arvoja. Sopimuspohjaisen ohjelmoinnin periaatteiden mukaan alkuehto on kuitenkin riittävä keino ydinongelman kannalta merkityksettömien vaihtoehtojen rajaamiseksi [42]. Loppuehto kertoo ne ehdot, joiden luvataan täyttyvän, jos alkuehto oli voimassa. Funktion `main()` alku- ja loppuehto ovat true, koska funktio toimii kaikissa tilanteissa samalla tavalla ehdoista riippumatta. Jatkossa tällä kurssilla kirjoitetaan funktioille vain kommentti ja alkuehto, sillä loppuehto on joissain tapauksissa melko vaikea muodostaa eikä siten kuulu vielä kurssille, jossa opetellaan ohjelmoinnin perusteita. [35]

Esimerkki 22. *Ohjelmoidaan toisen asteen yhtälön ratkaisukaava funktioita käyttäen.*

```
#tiedostossa tayratkaisukaava.py
import math

#Palauttaa ratkaisut toisen asteen yhtalolle.
#Palauttaa virheilmoituksen, jos a == 0.
#AE: a,b,c kuuluvat reaalilukuihin
```

```
#LE: RESULT: Yhtalon ax^2+bx+c=0 juuret
def ratkaise_yhtalo(a,b,c):
    if a != 0:
        diskriminantti = b**2 - 4*a*c
        if diskriminantti < 0:
            return "Yhtalolla ei ole reaalijuuria."
        elif diskriminantti == 0:
            x = -b / (2*a)
            return x
        else:
            x1 = (-b + math.sqrt(diskriminantti)) / (2*a)
            x2 = (-b - math.sqrt(diskriminantti)) / (2*a)
            return (x1,x2)
    else:
        return "Yhtalo ei ole toista astetta."
```

Pääohjelma:

```
#tiedostossa esimerkki22.py
from tayratkaisukaava import *

#Kysyy yhtalon kertoimet ja tulostaa ratkaisut.
#AE: true
#LE: true
def main():
    print """Ratkaistaan muotoa ax^2 + bx + c = 0 oleva
    toisen asteen yhtalo."""
    a,b,c=input("Anna yhtalon reaaliset kertoimet muodossa a,b,c: ")
    print "Ratkaisu on: "
    print ratkaise_yhtalo(a,b,c)

main()
```

Testataan ohjelmaa.

```
>>>
Ratkaistaan muotoa ax^2 + bx + c = 0 oleva
```

```
toisen asteen yhtalo.
Anna yhtalon reaaliset kertoimet muodossa a,b,c: 3,4,-1
Ratkaisu on:
(0.21525043702153024, -1.5485837703548635)
```

Tällaista ohjelmaa voisi havainnollistaa piirtämällä kyseisen funktion kuvaajan ja tarkastelemalla funktion nollakohtia myös sen avulla. Kuvaajien piirtäminen opitaan luvussa 4.4, jossa esitetään yhtenä esimerkkinä toisen asteen funktion kuvaajan piirtäminen.

4.4 Kuvaajien piirtäminen - Matplotlib

Kuten jo luvun 4.3 lopussa huomattiin, joitakin ohjelmia olisi hyvä havainnollistaa piirtämällä kuvaaja. Kaksiulotteisia kuvaajia voidaan piirtää esimerkiksi Pythonin erillisen piirtämiseen tarkoitetun Matplotlib-kirjaston avulla. Se ei sisälly Pythonin valmiiseen luokkakirjastoon, mutta sen voi ladata ilmaiseksi Internet-osoitteesta http://matplotlib.sourceforge.net. Jotta Matplotlib toimii, on tietokoneessa oltava asennettuna luvussa 4.2 esitelty Numerical Python -laajennus. Python Enthought Distribution -pakettiin nämä molemmat sisältyvät valmiiksi.

Tämä luku perustuu lähteiden [7, 21] tietoihin ja niiden sovelluksiin. Kaikissa funktioiden kuvaajia piirtävissä ohjelmakoodeissa esiintyy tällä kurssilla ohjelmakoodin alussa seuraavat rivit:

```
from fonttikoko import *
ax = fonttikokomuutos()
```

Ensimmäinen rivi kertoo, että ohjelman käyttöön tuodaan fonttikoko.py -niminen tiedosto. Tässä tiedostossa on määritelty funktio `fonttikokomuutos()`, joka suurentaa x- ja y-koordinaattien fonttikokoa. Toisella rivillä asetetaan funktion palauttama arvo muuttujaan `ax`. Kuvaajia piirrettäessä on nyt kirjoitettava käsky `ax.plot()` käskyn `plot()` sijasta, jotta ohjelma ottaa fonttikokomuutokset huomioon piirtäessään kuvaajaa. Funktion `fonttikokomuutos()` lähdekoodi esitetään tutkielman liitteessä Kuvaajien lähdekoodeja.

4.4.1 Piirtämisen perusteita

Piirretään ensimmäiseksi (x, y) -koordinaatistoon yksinkertaisesti pisteiden $(0, 0)$, $(1, 2)$ ja $(2, 4)$ kautta kulkeva suora. Ohjelman alussa on ensin tuotava Matplotlib käyttöön. Tämä onnistuu `from pylab import *` -käskyllä. Käskyn lopussa oleva tähti tarkoittaa, että koko koodiin saadaan käyttöön kaikki pylab-pakkauksen alihakemistot, kuten esimerkiksi Matplotlib, Scientific Python ja Numerical Python. Piirtäminen tapahtuu `plot()` -käskyllä. Haluttujen pisteiden x- ja y-koordinaattien arvot kirjoitetaan listojen muodossa `plot()` -käskyn sulkeisiin, kuten seuraavassa koodissa on tehty. Ohjelman loppuun on lisättävä vielä `show()` -käsky, jotta kuva piirtyisi näkyviin.

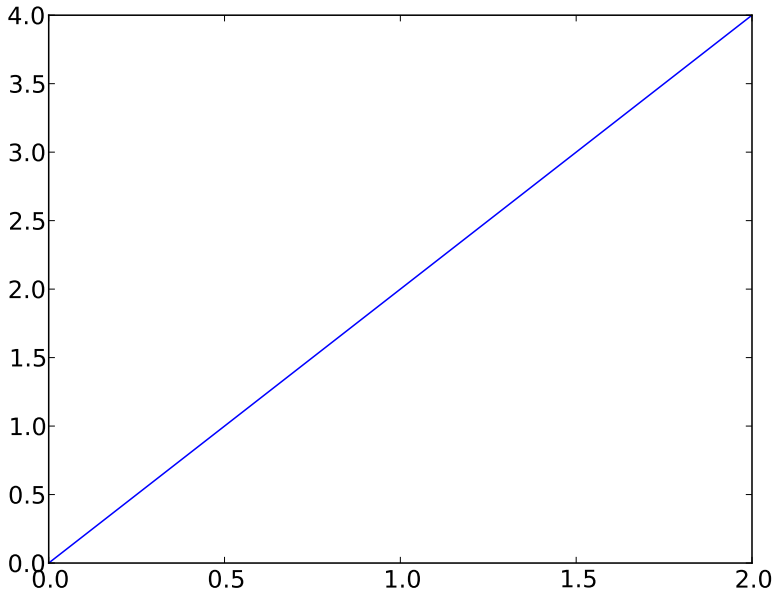
```
#tiedostossa mpl1.py
from pylab import *
from fonttikoko import *
ax = fonttikokomuutos()
ax.plot([0 ,1, 2], [0, 2, 4])
show()
```

Ohjelma piirtää seuraavalla sivulla esitettävän kuvan (Kuva 4).

4.4.2 Toisen asteen funktion kuvaaja

Toisen asteen yhtälön nollakohtia on järkevää havainnollistaa piirtämällä vastaavan funktion kuvaaja. Tarkastellaan tässä funktion $f(x) = 3x^2+4x-1$ kuvaajaa. Esimerkkikoodissa tuodaan ensin pylab-pakkaus käyttöön ja määritellään funktio aiemmin opitulla tavalla. Tällä kurssilla jätetään Matplotlib-sovellusten yhteydessä funktioiden kommentit ja alkuehdot kirjoittamatta, jotta ohjelmakoodista olisi helpompaa erottaa varsinaiset piirtämiseen tarvittavat käskyt. `arange()` -käskyllä määrätään tässä, että muuttuja x saa arvoja väliltä $[-2,3; 1,0]$ $0,1$ yksikön välein. Luku $0,1$ on siis niin sanottu kuvaajan tarkkuusparametri. `plot()` -käsky on tässä ilmaistu hieman eri tavalla kuin edellisessä esimerkissä. Periaate on kuitenkin sama. Piirretään kuvaaja sellaisten pisteiden kautta, joissa muuttuja x käy läpi edellä mainitut arvot ja kunkin pisteen y-koordinaatti määräytyy aina funktion $f(x) = 3x^2 + 4x - 1$

Kuva 4: Pisteiden $(0, 0)$, $(1, 2)$ ja $(2, 4)$ kautta kulkeva suora.

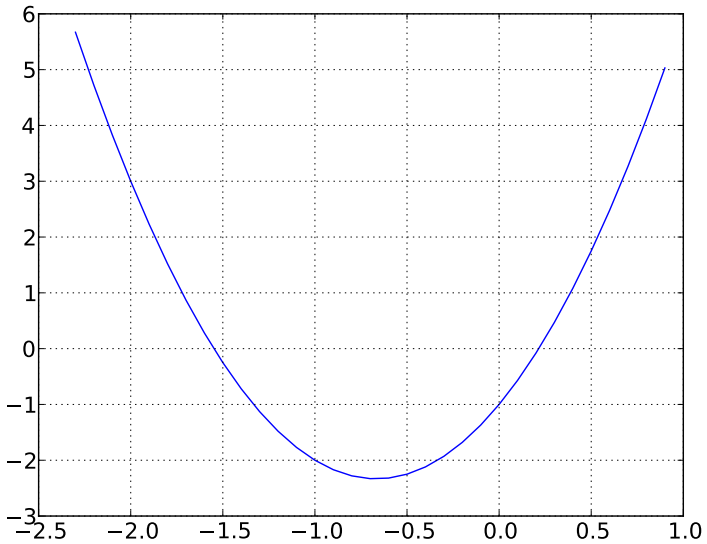


perusteella. Koodin lopussa on vielä uusi `grid()` käsky. Sen avulla saadaan kuvaan ruudukko, jotta kuvaajasta olisi helpompi lukea funktion arvoja x-akselin eri pisteissä.

```
#tiedostossa mpl2.py
from pylab import *
from fonttikoko import *
ax = fonttikokomuutos()
def f(x):
    return 3*x**2+4*x-1
x = arange(-2.3, 1.0, 0.1)
ax.plot(x, f(x))
grid()
show()
```

Ohjelman tulostama kuva (Kuva 5) esitetään seuraavalla sivulla. Vastaavalla tavalla voidaan piirtää minkä tahansa funktion kuvaaja.

Kuva 5: Funktion $f(x) = 3x^2 + 4x - 1$ kuvaaja.



4.4.3 Kaksi käyrää samaan kuvaan

Matplotlib-kirjaston avulla on mahdollista piirtää kahden tai useamman funktion kuvaajat samaan kuvaan. Tämä on kätevää, kun halutaan esimerkiksi tarkastella kuvaajien leikkauspisteitä tai vertailla vaikka jakson pituuden vaikutusta sinikäyrän muotoon. Toteutetaan tässä jälkimmäinen: piirretään funktioiden $f(x) = \sin(x)$ ja $g(x) = \sin(2x)$ kuvaajat samaan kuvaan. On huomattava, että luvussa 3.8 opittua funktiota `math.sin(x)` ei voida käyttää tässä tehtävässä, sillä se ei pysty käsittelemään listoja, ja tässä tehtävässä väli, jonka muuttuja x käy läpi, annetaan listan muodossa. Koska käskyllä `from pylab import *` saadaan käyttöön myös Numerical Pythonin funktiot, kuvaaja voidaan piirtää käyttäen Numerical Pythonin funktiota `sin(x)`. .

Ohjelmakoodissa määritellään ensin funktiot ja x-akseli kuten toisen asteen funktion kuvaajan yhteydessä opittiin. `plot()` -käskylle on nyt annettava neljä parametria, sillä ensin piirretään funktio $f(x)$ muuttujan x suhteen ja sitten funktio $g(x)$ muuttujan x suhteen. Tässä on kuvaajille annettu nimet `sin1` ja `sin2`, jotta kummankin piirtyvän viivan ulkonäköä voitaisiin muokata. Tämä onnistuu `setp()` -käskyllä. Käsky saa parametreikseen ky-

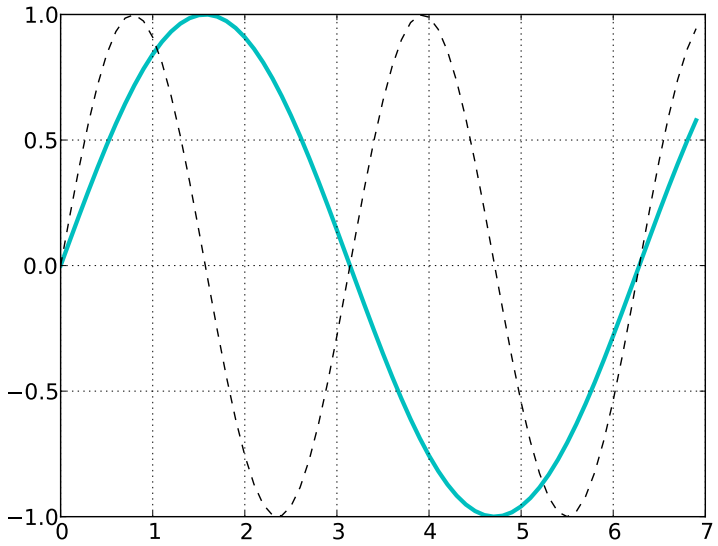
seiselle kuvaajalle annetun nimen, viivan tyylin (esimerkiksi yhtenäinen viiva tai katkoviiva), leveyden (float-tyyppinen luku) ja värin (tässä vaaleansininen ja musta). Kaikki mahdolliset värivaihtoehdot löytyvät osoitteesta http://matplotlib.sourceforge.net/api/colors_api.html.

```
#tiedostossa mpl3.py
from pylab import *
from fonttikoko import *
ax = fonttikokomuutos()
def f(x):
    return sin(x)
def g(x):
    return sin(2*x)
x = arange(0.0, 7.0, 0.1)
sin1, sin2 = ax.plot(x, f(x), x, g(x))
setp(sin1, linestyle='-', linewidth=3.0, color='c')
setp(sin2, linestyle='--', linewidth=1.0, color='k')
grid()
show()
```

Saadaan seuraavalla sivulla esittävä kuva (Kuva 6).

Myöhemmissä luvuissa jatketaan vielä kuvaajien piirtämisen harjoittelua aihekohtaisten esimerkkien avulla. Tilastosovelluksen yhteydessä opetellaan piirtämään piirakkakuvio ja Newtonin menetelmää täydennetään ja havainnollistetaan funktion kuvaajan avulla.

Kuva 6: Yhtenäinen viiva: $f(x) = \sin(x)$, katkoviiva: $g(x) = \sin(2x)$



4.5 Todennäköisyyslaskenta ja tilastot - Moduuli random

Tässä luvussa opitaan käyttämään moduulia `random`, joka sisältää satunnaislukujen generoimiseen tarkoitettuja funktioita. Satunnaisfunktioita voidaan käyttää monissa todennäköisyyslaskentaan ja tilastotieteeseen liittyvissä ohjelmissa. Ne helpottavat myös useiden pelien ohjelmointia mahdollistamalla esimerkiksi nopanheiton simuloinnin.

4.5.1 Moduuli random

Moduuli `random` sisältyy valmiina Pythonin luokkakirjastoon, kuten aikaisemmin kurssilla käsitelty `math`-moduulikin. Täten riittää ottaa se käyttöön ohjelman alussa käskyllä `import random`. Moduuli sisältää useita erilaisia satunnaislukuja generoivia funktioita. Tutustutaan esimerkkien avulla muutamiin tärkeimpiin moduulin funktioihin.

Funktio `random.randrange()` palauttaa satunnaisesti valitun alkion funktion `range()` palauttamasta listasta. Esimerkiksi funktio

`random.randrange(101)` palauttaa satunnaisen kokonaisluvun väliltä $[0, 100]$, sillä funktio `range()` palauttaa kaikki kokonaisluvut tältä väliltä. Vastaavasti funktio `random.randrange(10,21)` palauttaa satunnaisen kokonaisluvun väliltä $[10, 20]$ ja `random.randrange(0,101,2)` palauttaa satunnaisen parillisen luvun väliltä $[0, 100]$.

```
#tiedostossa moduulirandom1.py
import random
print random.randrange(101)
print random.randrange(10,21)
print random.randrange(0,101,2)

>>>
29
18
92
```

Funktion `random.randrange(a,b)` sijasta voidaan myös käyttää funktiota `random.randint(a,b)`, joka palauttaa satunnaisen kokonaisluvun väliltä $[a, b]$ [37].

```
#tiedostossa moduulirandom2.py
import random
print random.randint(10,21)

>>>
13
```

Funktio `random.random()` palauttaa satunnaisen desimaaliluvun väliltä $[0.0, 1.0)$ [37].

```
#tiedostossa moduulirandom3.py
import random
print random.random()

>>>
0.961495205109
```

Funktio `random.choice(listan_nimi)` palauttaa satunnaisesti valitun alkion listasta `listan_nimi`.

```
#tiedostossa moduulirandom4.py
import random
lista = [3,6,4,8,10,45,2,765,23,10086,"viimeinen_alkio"]
print random.choice(lista)


>>>
45
```

Huomautuksena mainittakoon, että Pythonin generoimat luvut eivät todellisuudessa ole satunnaisia vaan deterministisen algoritmin tuottamia lukuja. Deterministinen algoritmi käyttäytyy ennustettavasti ja tuottaa aina samalla alkusyötteellä tulokseksi saman sarjan. Mikään aritmeettiseen operaatioon perustuva yksittäinen luku ei ole täysin satunnainen, mutta deterministisen algoritmin tuottamien lukujen tilastolliset ominaisuudet vastaavat jonkin etukäteen annetun todennäköisyysjakauman ominaisuuksia. Tarvittava algoritmin kutsujen määrä, jonka aikana algoritmi käy kaikki syötteet läpi ja jonka jälkeen sama tulossarja toistuu, on Pythonin satunnaislukugeneraattorissa suuruudeltaan $2^{19937} - 1$. Tällä tavoin generoidut luvut soveltuvat useimpiin ohjelmointitehtäviin, joissa tarvitaan satunnaislukuja. [37, 41, 47]

4.5.2 Kolikonheittosimulaattori

Perinteistä kolikonheittoa on helppo soveltaa todennäköisyyslaskentaan ja tilastotieteeseen.

Esimerkki 23. *Kirjoitetaan ohjelma, joka generoi satunnaisia kokonaislukuja väliltä* $[0, 1]$ *eli heittää kolikkoa. Jos heittotulos on numero* 0*, tallennetaan ohjelman alussa luotuun listaan alkio kruuna ja jos heittotulos on numero* 1*, tallennetaan listaan alkio klaava. Lopuksi pääohjelma tulostaa heittotulokset sisältävän listan, kun heittojen lukumäärä on kymmenen.*

```
#tiedostossa esimerkki23.py
import random
```

```
#Palauttaa heittotulokset sisaltavan listan.
#AE: heittojen_maara > 0 ja heittojen_maara on kokonaisluku
def kolikonheitto(heittojen_maara):
    heitot = []                             #alustetaan lista
    for i in range(heittojen_maara):
        heittotulos = random.randrange(2) #satunn. luku 0 tai 1
        if heittotulos == 0:
            heitot.append("kruuna")       #lisataan alkio listaan
        else:
            heitot.append("klaava")       #lisataan alkio listaan
    return heitot

#Tulostaa 10 heiton tulokset.
#AE:true
def main():
    print kolikonheitto(10)

main()
```

Testataan ohjelmaa kolme kertaa ja huomataan, kuinka heittotulokset vaihtelevat eri testauskerroilla.

```
>>>
['klaava', 'klaava', 'kruuna', 'klaava', 'klaava', 'klaava',
'kruuna', 'klaava', 'klaava', 'kruuna']
>>>
['klaava', 'kruuna', 'kruuna', 'kruuna', 'klaava', 'kruuna',
'klaava', 'kruuna', 'klaava', 'klaava']
>>>
['klaava', 'kruuna', 'kruuna', 'kruuna', 'klaava', 'klaava',
'kruuna', 'klaava', 'klaava', 'klaava']
```

Ensimmäisellä kerralla saatiin siis 3 kruunaa ja 7 klaavaa, toisella kerralla 5 kruunaa ja 5 klaavaa ja kolmannella kerralla 4 kruunaa ja 6 klaavaa. Luonnollisesti 5 kruunaa ja 5 klaavaa on todennäköisin vaihtoehto, mutta mikä on sen todennäköisyys? Eri yhdistelmien todennäköisyydet saadaan laskettua

binomikaavalla P(saadaan k kruunaa)= $\binom{n}{k}p^k q^{n-k}$, missä n on toistojen eli heittojen määrä, p on kruunan todennäköisyys yhdellä heitolla ja $q = 1 - p$ eli klaavan todennäköisyys yhdellä heitolla [12].

Lasketaan esimerkkinä edellisten tulosten todennäköisyydet eli P(saadaan 3 kruunaa), P(saadaan 5 kruunaa) ja P(saadaan 4 kruunaa). Todennäköisyydet muille kruunien määrille lasketaan vastaavasti.

- P(saadaan 3 kruunaa)=$\binom{10}{3}0,5^3 0,5^{10-3} \approx 0,12$
- P(saadaan 5 kruunaa)=$\binom{10}{5}0,5^5 0,5^{10-5} \approx 0,25$
- P(saadaan 4 kruunaa)=$\binom{10}{4}0,5^4 0,5^{10-4} \approx 0,21$

Jos ohjelmaa testattaisiin hyvin monta kertaa, lähestyttäisiin todennäköisesti edellä laskettuja todennäköisyyksien arvoja. Tällöin noin 25 prosenttia testauksista tuottaisi 5 kruunaa ja 5 klaavaa, noin 21 prosenttia tuottaisi 4 kruunaa ja 6 klaavaa ja niin edelleen.

Esimerkki 24. *Muokataan kolikonheitto-ohjelman pääohjelmaa siten, että käyttäjä saa määrätä heittojen lukumäärän.*

Nyt, kun käyttäjän on mahdollista antaa heittojen määräksi kuinka suuri luku tahansa, on järkevää antaa ohjelman laskea saatujen kruunien ja klaavojen lukumäärät. Tätä varten kirjoitetaan laskuri-funktio, joka käy läpi heittotulokset sisältävän listan jokaisen alkion ja kasvattaa aina joko kruunien tai klaavojen määrän kertovaa muuttujaa. Funktio palauttaa kahden alkion listan, jonka alkiot ovat kruunien määrä ja klaavojen määrä. Nyt pääohjelma voi tulostaa listan ensimmäisen ja toisen alkion, jolloin saadaan tietää nämä määrät.

Kruunien ja klaavojen määrien lisäksi olisi mielenkiintoista tietää myös niiden prosenttiosuudet kaikista heittotuloksista. Annetaan pääohjelman laskea ja tulostaa ne. Kruunien prosenttiosuus kaikista heitoista lasketaan kaavalla $100 \cdot \frac{kruunien\_maara}{heittojen\_maara}$. Klaavojen prosenttiosuus lasketaan vastaavasti. Lisäksi pääohjelma voi laskea binomikaavalla todennäköisyyden, jolla saadaan juuri kyseinen määrä kruunia ja klaavoja. Binomikaavassa esiintyvää tulon termiä $\binom{n}{k}$ varten Scientific Python -laajennuspaketissa

on olemassa valmis funktio `comb()` [40]. Scientific Python ei sisälly automaattisesti Python-tulkkiin, vaan se on ladattava Internetistä osoitteesta http://sourceforge.net/projects/scipy/files/. Python Enthought Distribution -pakettiin se kuitenkin sisältyy valmiiksi. Jotta funktiota `comb()` voidaan käyttää, on Scientific Python otettava käyttöön ohjelman alussa käskyllä `from scipy import *`.

```
#tiedostossa esimerkki24.py
from scipy import *
import random

#Palauttaa heittotulokset sisaltavan listan.
#AE: heittojen_maara > 0 ja heittojen_maara on kokonaisluku
def kolikonheitto(heittojen_maara):
    heitot = []                           #alustetaan lista
    for i in range(heittojen_maara):
        heittotulos = random.randrange(2) #satunn. luku 0 tai 1
        if heittotulos == 0:
            heitot.append("kruuna")       #lisataan alkio listaan
        else:
            heitot.append("klaava")       #lisataan alkio listaan
    return heitot

#Palauttaa listan, jonka alkiot ovat kruunien maara ja klaavojen
#maara.
#AE: heitto_lista != none
def laskuri(heitto_lista):
    kruuna_lkm = 0
    klaava_lkm = 0
    for i in range(len(heitto_lista)):
        if heitto_lista[i] == "kruuna":
            kruuna_lkm += 1
        else:
            klaava_lkm += 1
    return [kruuna_lkm, klaava_lkm]         #lkm_lista
```

```
#Tulostaa heittotulokset sisaltavan listan ja kruunien ja klaavojen
#maarat.
#AE: true
def main():
    heittojen_lkm = input("\nAnna heittojen lukumaara: ")
    heitot = kolikonheitto(heittojen_lkm)
    print heitot
    lkm_lista = laskuri(heitot)
    print "\nKruunia saatiin ",lkm_lista[0]
    print "ja klaavoja ",lkm_lista[1],"."
    print "\nKruunia saatiin ",100*lkm_lista[0]/heittojen_lkm,"%."
    print "\nKlaavoja saatiin ",100*lkm_lista[1]/heittojen_lkm,"%."
    print "\nTodennakoisyys saada ",lkm_lista[0]
    print "kruunaa ",heittojen_lkm," heitolla on"
    n_yli_k = comb(heittojen_lkm,lkm_lista[0])
    tn = n_yli_k*0.5**lkm_lista[0]*0.5**lkm_lista[1]
    print "%.4f"%tn
main()
```

Testataan ohjelmaa.

```
>>>

Anna heittojen lukumaara: 40
['klaava', 'kruuna', 'klaava', 'kruuna', 'kruuna', 'kruuna',
'klaava', 'klaava', 'kruuna', 'klaava', 'klaava', 'kruuna',
'klaava', 'kruuna', 'kruuna', 'kruuna', 'klaava', 'kruuna',
'kruuna', 'kruuna', 'klaava', 'klaava', 'klaava', 'kruuna',
'klaava', 'kruuna', 'klaava', 'kruuna', 'kruuna', 'kruuna',
'kruuna', 'kruuna', 'klaava', 'klaava', 'klaava', 'klaava',
'kruuna', 'klaava', 'kruuna', 'kruuna']

Kruunia saatiin  22
ja klaavoja  18.
```

```
Kruunia saatiin  55 %.

Klaavoja saatiin  45 %.

Todennakoisyys saada  22
kruunaa 40 heitolla on
0.1031
```

4.5.3 Piirakkakuvion piirtäminen

Luku perustuu lähteen [21] tietoihin ja niiden sovelluksiin. Piirretään piirakkakuvio havainnollistamaan edellisessä luvussa esitetyn kolikonheitto-ohjelman tuloksia. Kuten aiemmin on opittu, tuodaan koodin alussa pylab-pakkaus kokonaisuudessaan käyttöön. Piirakkakuvio saadaan piirrettyä `pie()` -käskyllä. Tämä käsky vaatii useita parametreja ja saattaa siten näyttää aluksi monimutkaisemmalta kuin todellisuudessa on. Sille on ensiksi annettava listan muodossa tieto siitä, kuinka monta kappaletta esimerkiksi tässä kruunia ja klaavoja on saatu. Ohjelma osaa laskea näiden kappalemäärien avulla kruunien ja klaavojen prosenttiosuudet yhteissummasta ja sovittaa lohkojen koot sopiviksi tämän mukaan.

Kuvaan piirtyy lohkoja yhtä monta kuin edellä mainitussa listassa on alkioita (tässä siis kaksi). Jos haluttaisiin havainnollistaa vaikka nopanheiton tulosjakaumaa, kirjoitettaisiin listaan kuusi alkiota ja saataisiin piirakkakuvio, jossa on kuusi eri lohkoa. Samalla täytyy myös seuraavassa kappaleessa esiteltyjen parametrien sulkeisiin lisätä määre kaikkia kuutta alkiota varten.

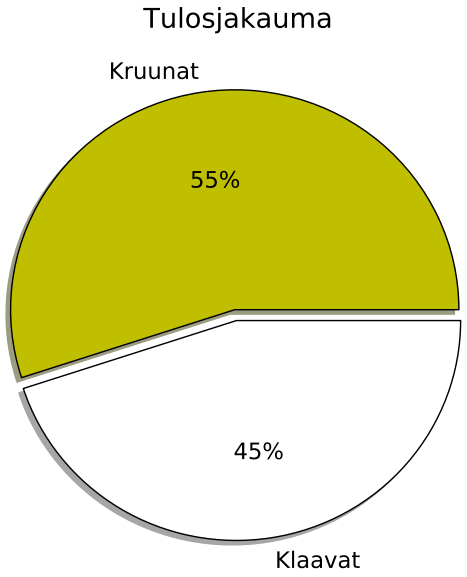
Muita parametreja ovat lohkojen värien määritys (tässä vihreänkeltainen ja valkoinen), `explode` eli kunkin lohkon etäisyys piirakan keskipisteestä, `labels` eli selitys, mitä kukin lohko kuvaa, `autopct='%1.f%%'`, jolla saadaan prosenttiosuudet lohkojen sisään näkyviin sekä `shadow` eli varjo piirakan ulkoreunalle. Näiden kaikkien arvoiksi olisi myös mahdollista asettaa yksinkertaisesti None, jolloin saataisiin piirakkakuvio oletusväreillä (vihreä ja sininen) ilman tekstejä ja muita selvennyksiä. `title()` -käskyn avulla voidaan kirjoittaa kuvalle vielä otsikko.

```
#tiedostossa mpl4.py
```

```
from pylab import *
x = [22, 18] #kruunien ja klaavojen kappalemaarat
rcParams['font.size'] = 16  #fonttikoko
pie(x, colors=('y', 'w'), explode=(0.0, 0.05),
    labels=('Kruunat', 'Klaavat'),
    autopct='%1.f%%', shadow=True)
title('Tulosjakauma')
show()
```

Tulokseksi saadaan seuraava kuva (Kuva 7).

Kuva 7: Kolikonheiton tulokset.



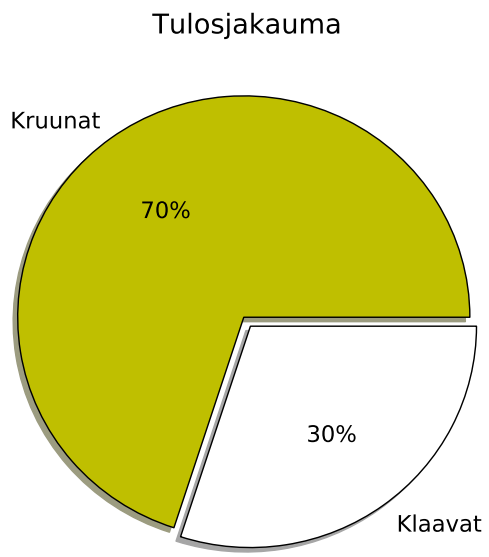
Vastaava kuva voidaan toteuttaa myös niin, että ei määritetä kruunien ja klaavojen kappalemääriä valmiiksi, vaan annetaan ohjelman heittää kolikkoa juuri ennen kuvan piirtämistä. Nyt on muistettava tuoda moduuli random käyttöön ohjelman alussa, sillä Pythonin valmiit moduulit eivät sisälly pylab-pakkaukseen.

```
#tiedostossa mpl5.py
from pylab import *
```

```
import random
tulokset = []
for i in range(100):       #sata heittoa
    tulokset.append(random.randrange(2))
kruunat = 0
for i in range(len(tulokset)):
    if tulokset[i] == 0:
        kruunat += 1
klaavat = 100 - kruunat
x = [kruunat, klaavat]     #kruunien ja klaavojen lkm
rcParams['font.size'] = 16 #fonttikoko
pie(x, colors=('y', 'w'), explode=(0.0, 0.05),
    labels=('Kruunat', 'Klaavat'),
    autopct='%1.f%%', shadow=True)
title('Tulosjakauma')
show()
```

Tulokseksi voidaan saada vaikka tällainen kuva (Kuva 8). Melko epätodennäköistä, mutta mahdollista!

Kuva 8: Kolikonheiton tulokset.



4.6 Kertoman ohjelmoiminen - Rekursio

Luku perustuu lähteen [9] tietoihin ja niiden sovelluksiin. Algoritmiset ongelmat (myös rekursiivisesti määritellyt kuten Fibonaccin lukujono) on kurssilla tähän mennessä ratkaistu iteratiivisesti eli päättyviä toistorakenteita käyttäen. Toinen tapa ratkaista niitä on käyttää rekursiota. Rekursiivinen funktio sisältää itsensä kutsun, tosin tietenkin alkuperäisistä poikkeavin parametrein. Funktiossa on siis kutsusilmukka, minkä seurauksena se palaa aina takaisin itseensä ja käyttää omaa informaatiotaan yhä uudelleen, kunnes kohtaa rekursion pohjan. Rekursiota käytettäessä on huolehdittava, että rekursion pohja on saavutettavissa. Muuten kone joutuu jumiin loputtomaan rekursiosilmukkaan.

Tässä luvussa tutustutaan rekursiofunktioon luvun n kertoman avulla sekä vertaillaan rekursiota ja iteraatiota keskenään. Luvun n kertoma määritellään iteratiivisesti kaavalla $n! = 1 \cdot 2 \cdot 3 \cdots n$ ja rekursiivisesti seuraavasti:

$$n! = \begin{cases} 1 & \text{jos } n = 0 \\ n \cdot (n-1)! & \text{muulloin.} \end{cases}$$

Luvun n kertoma $n!$ saadaan siis helposti laskettua kaavalla $n \cdot (n-1)!$, kun tiedetään $(n-1)!$. Lukua $(n-1)!$ ei kuitenkaan vielä tiedetä, mutta se saadaan laskettua kaavalla $(n-1) \cdot (n-2)!$, kun tiedetään $(n-2)!$. Luku $(n-2)!$ saadaan taas kaavalla $(n-2) \cdot (n-3)!$, kun tiedetään $(n-3)!$ ja niin edelleen. Palauttamista jatketaan, kunnes saavutetaan rekursion pohja $(n = 0)$. Tähän mennessä ei ole vielä saatu tulokseksi muita numeroarvoja kuin 0!. Vasta nyt lasku alkaa purkautua takaisin päin, ja saadaan askel kerrallaan numeroarvot kaikille kertomille (viimeisenä myös siis $n!$).

Esimerkki 25. *Ohjelmoidaan luvun n kertoma ensin iteratiivisena ja sitten rekursiivisena funktiona.*

Kertoma ohjelmoidaan pelkästään harjoituksen vuoksi, sillä Pythonissa luvun n kertoman arvon voi laskea suoraan `math`-moduulin käskyllä `math.factorial(n)`. Ohjelman main-osiossa käytetään Pythonin `time`-moduulia, jonka avulla voidaan vertailla iteratiivisen ja rekursiivisen funktion suoritusaikoja.

```
#tiedostossa esimerkki25.py
from time import *      #otetaan time-moduuli kayttoon

#Palauttaa luvun n kertoman.
#AE: n on pos. kokonaisluku
def kertoma_iter(n):
    kertoma = 1
    for i in range(1,n+1):
        kertoma = kertoma * i
    return kertoma

#Palauttaa luvun n kertoman.
#AE: n on pos. kokonaisluku
def kertoma_rek(n):
    if n == 0:
        return 1
    else:
        return (n * kertoma_rek(n-1))

#Tulostaa suoritusajat.
#AE: true
def main():
    k = int(input("Minka luvun kertoman haluat laskea? "))
    alkuaika_iter = 10**9 * time()           #ajanotto alkaa
    kertoma_iter(k)
    loppuaika_iter = 10**9 * time()          #ajanotto loppuu
    aika_iter = loppuaika_iter - alkuaika_iter
    print "Iterativisen aika: ",aika_iter
    alkuaika_rek = 10**9 * time()            #ajanotto alkaa
    kertoma_rek(k)
    loppuaika_rek = 10**9 * time()           #ajanotto loppuu
    aika_rek = loppuaika_rek - alkuaika_rek
    print "Rekursiivisen aika: ",aika_rek

main()
```

Testaustuloksia eri luvuilla:

```
>>>
Minka luvun kertoman haluat laskea? 30
Iteratiivisen aika (ns):  0.0
Rekursiivisen aika (ns):  999936.0

>>>
Minka luvun kertoman haluat laskea? 100
Iteratiivisen aika (ns):  999936.0
Rekursiivisen aika (ns):  1000192.0

>>>
Minka luvun kertoman haluat laskea? 450
Iteratiivisen aika (ns):  1999872.0
Rekursiivisen aika (ns):  4000000.0

>>>
Minka luvun kertoman haluat laskea? 900
Iteratiivisen aika (ns):  11000064.0
Rekursiivisen aika (ns):  17000192.0
```

Ohjelman main-osiossa ei tarkoituksella tulosteta kertoman arvoa, sillä suurilla syötteen arvoilla se on erittäin suuri ja pienillä syötteen arvoilla suoritusajoissa ei ole nähtävissä mitään eroa. Kertoman tulostuksen voisi lisätä main-osioon asettamalla esimerkiksi iteratiivisen funktion palauttaman arvon kertoma-nimiseen muuttujaan (`kertoma = kertoma_iter(k)`) ja lisäämällä ohjelman loppuun rivin `print kertoma`.

Suoritusajat ovat kertoman tapauksessa melko pieniä, joten ne on tulostettu ohjelmassa nanosekunteina (1 sekunti = 10^9 nanosekuntia). Suoritusajat vaihtelevat tietokoneesta riippuen samallakin syötteellä eri testauskerroilla, mutta kaikilla eri testauskerroilla ja kaikilla syötteillä suuruusjärjestys pysyy aina samana: iteratiivinen funktio on aina nopeampi kuin rekursiivinen. Ero ei kuitenkaan ole merkittävä ja joka tapauksessa rekursio on hyvin lyhyt tapa kirjoittaa ohjelmia.

4.7 Numeerisia menetelmiä

Tässä luvussa opitaan ohjelmoimaan numeerisen matematiikan kurssilla esille tulleita menetelmiä. Vaikka näiden menetelmien laskeminen kynällä ja paperilla sekä laskimella onnistuu, se on varsin työlästä ja siksi menetelmät onkin hyvä toteuttaa ohjelmoimalla. Ohjelmissa ei tule esille varsinaista uutta asiaa, paremminkin vain funktioiden käytön ja toistorakenteiden kertausta. Tärkeintä tässä vaiheessa on se, että opitaan muotoilemaan kirjallisesti kuvattu laskentamenetelmä toimivaksi ohjelmakoodiksi.

Luvussa ohjelmoidaan seuraavat menetelmät: numeerinen derivointi, Newtonin menetelmä, puolisuunnikassääntö sekä Simpsonin sääntö. Ohjelmoidessa käytetään esimerkkeinä näitä menetelmiä koskevia vanhoja ylioppilastehtäviä. Ylioppilastehtävien tehtävänannot ovat peräisin lähteestä [19]. Harjoitustehtäviksi jätetään välinpuolitusmenetelmän, kiintopisteiteraation sekä keskipistesäännön ohjelmoiminen. Kurssin lisämateriaalista löytyy esimerkit Eulerin menetelmästä, (Eulerin) keskipistemenetelmästä, kertaluvun 4 Runge-Kutta -menetelmästä (ohjelmointi lisätehtävänä) ja Gaussin eliminoinnista kolmen muuttujan tapauksessa sekä tietoa numeerisen laskennan virhelähteistä.

Lukujen 4.7.1-4.7.4 menetelmiin liittyvä teoria perustuu lähteen [10] tietoihin, ellei toisin mainita. Luvuissa 4.7.5-4.7.7 viittaukset mainitaan erikseen.

4.7.1 Numeerinen derivaatta

Funktion $f(x)$ derivaatta pisteessä $x = a$ määritellään erotusosamäärän raja-arvona $f'(a) = \lim_{h \to 0} \frac{f(a+h)-f(a)}{h}$. Funktion $f(x)$ derivaatalle pisteessä $x = a$ saadaan siis numeerinen likiarvo $f'(a) \approx \frac{f(a+h)-f(a)}{h}$ asettamalla luku h riittävän pieneksi. Tämä tapa ei välttämättä ole kuitenkaan paras mahdollinen, sillä se ei ota huomioon funktion toispuolisia raja-arvoja, eli toisin sanoen se ei huomioi funktion käyttäytymistä kohdan $x = a$ eri puolilla. Parempi arvio numeeriselle derivaatalle saadaan tarkastelemalla funktion arvoja kohdan $x = a$ molemmilla puolilla pisteissä $x = a + h$ ja $x = a - h$ ja asettamalla nyt luku h riittävän pieneksi. Tällöin $f'(a) \approx \frac{f(a+h)-f(a-h)}{2h}$. Tätä lauseketta

kutsutaan keskeisdifferenssiksi [15].

Ylioppilastehtävä 1. *(Syksy 2004, tehtävä 10) Määritä funktion* $f(x) = \frac{1}{x}$ *derivaatta pisteessä* $x = 2$ *laskemalla erotusosamäärän raja-arvo.*

Käytetään tässä kuitenkin keskeisdifferenssin lauseketta.

```
#tiedostossa ylioppilasteht1.py
#Palauttaa funktion f(x)=1/x arvon.
#AE: x!=0
def f(x):
    return 1/x

#Tulostaa keskeisdifferenssin arvot.
#Saa parametreina pisteen a, jossa derivaatta
#lasketaan seka askeleen h pituuden.
#AE: a kuuluu reaalilukuihin ja h>0
def numder(a, h):
    while h>10**(-5):
        print "%1.9f" % ((f(a+h)-f(a-h))/(2*h))
        h*=0.1
    return " "

#main
numder(2.0, 1.0)

>>>
-0.333333333
-0.250626566
-0.250006250
-0.250000062
-0.250000001
-0.250000000
```

Kun lukua h pienennetään tarpeeksi, huomataan, että $\lim_{h\to 0} \frac{f(2+h)-f(2-h)}{2h} = -0,25$. Siis $f'(2) = -0,25$. Kurssin lisämateriaalissa (luku 4.7.7) tarkastellaan, kuinka paljon on tämä tarpeeksi sekä tutkitaan, voiko luvun h arvoa

pienentää liikaa. Numeerisesti saatu tulos voidaan tässä tapauksessa tarkistaa ratkaisemalla sama tehtävä analyyttisesti: $f'(x) = -\frac{1}{x^2}$ ja $f'(2) = -\frac{1}{2^2} = -0,25$.

4.7.2 Yhtälön nollakohdat - Newtonin menetelmä

Newtonin menetelmä (jota kutsutaan joskus myös Newtonin ja Raphsonin menetelmäksi) on yleinen tapa etsiä numeerisesti ratkaisua yhden muuttujan yhtälöille. Menetelmää käytettäessä yhtälö muutetaan aina muotoon, jossa kaikki termit ovat yhtälön vasemmalla puolella ja etsitään näin saadun funktion nollakohtia. Menetelmän toimivuuden ehtona on, että nollakohdassaan sekä sen lähellä funktio on derivoituva ja derivaatan arvo on nollasta poikkeava.

Newtonin menetelmää käytettäessä tarvitaan aina alkuarvaus (olkoon se kohta x_0). Kohtaan x_0 piirretään funktiolle tangentti. Tangentin ja x-akselin leikkauskohta x_1 otetaan uudeksi tarkastelupisteeksi ja piirretään siihen kohtaan funktiolle uusi tangentti, joka leikkaa x-akselin kohdassa x_2 ja niin edelleen.

Kohtaan x_0 piirretyn tangentin yhtälö on $y - f(x_0) = f'(x_0)(x - x_0)$. Tangentti kulkee pisteen $(x_1, 0)$ kautta. Näin ollen edellä oleva tangentin yhtälö voidaan kirjoittaa muotoon $-f(x_0) = f'(x_0)(x_1 - x_0)$. Uuden tarkastelukohdan x_1 arvo ratkaistuna edellisestä yhtälöstä on siis $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$.

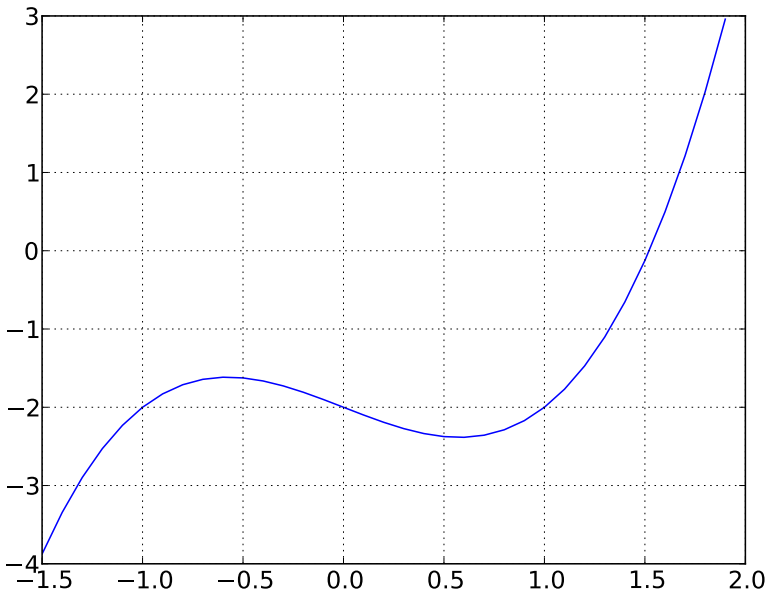
Yleisesti Newtonin menetelmä voidaan määritellä rekursiivisena jonona: Lähdetään liikkeelle alkuarvauksesta x_0. Seuraavat arvot saadaan kaavasta $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$, kun $i = 0, 1, 2, 3, ...$ Tällä tavoin muodostettu jono $x_0, x_1, x_2, x_3, ...$ suppenee yleensä hyvin kohti yhtälön $f(x) = 0$ ratkaisua. Onnistuminen riippuu kuitenkin alkuarvauksesta. Jos alkuarvaus x_0 on liian kaukana funktion nollakohdasta tai jos kohdan x_0 ja funktion todellisen nollakohdan välissä funktio ei ole derivoituva tai funktion derivaatta saa arvon nolla, haluttuun tulokseen ei päästä.

Ylioppilastehtävä 2. *(Kevät 2009, tehtävä 12) Määritä Newtonin menetelmällä yhtälön $x^3 = x + 2$ juuri kahden desimaalin tarkkuudella. Osoita, että yhtälöllä on täsmälleen yksi juuri välillä $[1, \infty[$.*

Muutetaan yhtälö ensin muotoon $x^3 - x - 2 = 0$. Ennen varsinaisen ohjelman kirjoittamista piirretään funktion $f(x) = x^3 - x - 2$ kuvaaja Matplotlibin avulla (Kuva 9), jotta alkuarvauksen määrittäminen helpottuisi.

```
#tiedostossa ylioppilasteht2kuva.py
from pylab import *
from fonttikoko import *
ax = fonttikokomuutos()
def f(x):
    return x**3-x-2
x = arange(-1.5, 2.0, 0.1)
ax.plot(x, f(x))
grid()
show()
```

Kuva 9: $f(x) = x^3 - x - 2$



Kuvaajan perusteella huomataan, että hyvä alkuarvaus olisi esimerkiksi $x_0 = 1,5$.

Seuraava ohjelma perustuu siihen, että etsittäessä funktion nollakohtaa tietyllä tarkkuudella $2h$ lasketaan uusia tarkastelukohtia x_i ja funktion arvoja tarkastelukohdissa, kunnes $f(x_i-h)\cdot f(x_i+h) < 0$. Kun taulukkoon tulostuu ensimmäinen negatiivinen lausekkeen $f(x_i - h) \cdot f(x_i + h)$ arvo, on löydetty kohta $x = x_i$, jolle funktion arvot $f(x_i - h)$ ja $f(x_i + h)$ ovat erimerkkiset. Funktion nollakohdan on oltava avoimella välillä $]x_i - h, x_i + h[$. Tässä tehtävässä halutaan laskea funktion nollakohta kahden desimaalin tarkkuudella, joten sijoitetaan $h = 0,005$. Kun funktion arvot $f(x_i-0,005)$ ja $f(x_i+0,005)$ ovat erimerkkiset, nollakohta kahden desimaalin tarkkuudella laskettuna on avoimella välillä $]x_i - 0,005, x_i + 0,005[$.

```
#tiedostossa ylioppilasteht2.py
#Palauttaa funktion f(x)=x**3-x-2 arvon.
#AE: x kuuluu reaalilukuihin
def f(x):
    return x**3-x-2

#Palauttaa derivaattafunktion f'(x)=3*x**2-1 arvon.
#AE: x kuuluu reaalilukuihin
def df(x):
    return 3*x**2-1

#Tulostaa likiarvoja funktion f nollakohdaksi.
#AE: x_i kuuluu reaalilukuihin.
def newton():
    i = 0
    x_i = 2.0                           #alkuarvaus
    h = 0.005                           #haluttu tarkkuus
    print "i  x_i        f(x_i)     f(x_i-h)*f(x_i+h)"
    taul = (i,x_i,f(x_i),f(x_i-h)*f(x_i+h))
    print "%1d %9.6f %10.6f %12.4e"%taul
    while f(x_i - h)*f(x_i + h) >= 0:
        x_i = x_i-(f(x_i)/df(x_i))      #uusi tarkastelukohta
        i += 1
        taul = (i,x_i,f(x_i),f(x_i-h)*f(x_i+h))
```

```
        print "%1d %9.6f %10.6f %12.4e"%taul
    return " "
#main
newton()

>>>
i  x_i        f(x_i)     f(x_i-h)*f(x_i+h)
0  1.500000  -0.125000   1.4770e-02
1  1.521739   0.002137  -8.7913e-04
```

Huomataan, että hyvän alkuarvauksen ansiosta haluttu tulos saatiin jo kahden askeleen jälkeen. Funktion $f(x) = x^3 - x - 2$ nollakohta kahden desimaalin tarkkuudella on $x \approx 1,52$, joka on siis samalla yhtälön $x^3 = x + 2$ juuri. Lasketaan vielä funktion f arvo sijoittamalla muuttujan x paikalle kaksidesimaalinen likiarvo $1,52$ ja verrataan tuloksen tarkkuutta ohjelman laskeman jälkimmäisen ratkaisun avulla saatuun funktion arvoon `0.002137`.

```
>>> print "%.6f"%(1.52**3-1.52-2)
-0.008192
```

Testataan vielä ratkaisujonon suppenemista suorittamalla sama ohjelma alkuarvauksen arvolla $x_0 = 2.0$.

```
>>>
i  x_i        f(x_i)     f(x_i-h)*f(x_i+h)
0  2.000000   4.000000   1.5998e+01
1  1.636364   0.745304   5.5442e-01
2  1.530392   0.053939   2.0139e-03
3  1.521441   0.000367  -8.8316e-04
```

Jono suppenee edelleen hyvin kohti yhtälön ratkaisua, mutta tällä kerralla tarvittiin neljä askelta.

Tehtävänannon jälkimmäisen lauseen suorittaminen ei vaadi ohjelmointia. Sen voi ratkaista osoittamalla, että funktion $f(x) = x^3 - x - 2$ derivaatta on positiivinen aina, kun muuttuja x saa kaikki arvot väliltä $[1, \infty[$. Siis $f'(x) = 3x^2 - 1 > 0$, kun $x \in [1, \infty[$. Funktio on aidosti kasvava tällä välillä, joten $x \approx 1,52$ on välin ainoa juuri.

4.7.3 Numeerinen integrointi - Keskipistesääntö ja puolisuunnikassääntö

Epänegatiivisen funktion f kuvaajan ja x-akselin väliin jäävälle pinta-alalle välillä $[a, b]$ voidaan laskea likiarvoja eri tavoin numeerisesti integroimalla. Jaetaan ensin väli $[a, b]$ yhtä pitkiin osaväleihin (olkoon näitä n kappaletta). Kunkin osavälin pituus on $d = \frac{b-a}{n}$. Keskipistesäännössä korvataan kysytty pinta-ala suorakulmioilla, joiden leveys on yhtäsuuri kuin jakovälin pituus ja korkeus on yhtäsuuri kuin funktion arvo välin keskipisteessä. Suorakulmioiden alojen summa on ylä- ja alasumman välissä, joten se lähenee kysyttyä pinta-alaa, kun jakovälejä tihennetään rajatta. Keskipistesäännön ohjelmoiminen jätetään harjoitustehtäväksi (luvun 4 harjoitustehtävä 28).

Keskipistesäännöllä ei kuitenkaan saada helposti kovin tarkkaa likiarvoa. Koska tarkkaan likiarvoon pyrittäessä termejä tarvitaan paljon, saattavat pyöristysvirheet alkaa vaikuttaa tulokseen jo niin paljon, ettei tarkkaan tulokseen päästä edes jakovälejä lisäämällä. Tulosta voidaan yrittää parantaa korvaamalla suorakulmiot puolisuunnikkailla. Tällöin kunkin puolisuunnikkaan yhdensuuntaisten sivujen pituudet ovat samat kuin funktion arvot välin päätepisteissä (merkitään y_i ja y_{i+1}) ja x-akselilla olevan sivun pituus on sama kuin jakovälin pituus $d = \frac{b-a}{n}$. Tällöin puolisuunnikkaan ala on $\frac{y_i+y_{i+1}}{2} \cdot d$.

Merkitään funktion arvoja jakovälien päätepisteissä $y_0, y_1, y_2, ..., y_n$. Puolisuunnikkaiden alojen summa on näin ollen

$$\frac{y_0+y_1}{2} \cdot d + \frac{y_1+y_2}{2} \cdot d + ... + \frac{y_{n-1}+y_n}{2} \cdot d = d \cdot (\frac{y_0}{2} + y_1 + y_2 + ... + y_{n-1} + \frac{y_n}{2}).$$

Ylioppilastehtävä 3. *(Kevät 2007, tehtävä 12) Laske integraalin* $\int_1^3 \frac{1}{x} dx$ *tarkka arvo. Laske sille myös viisidesimaalinen likiarvo puolisuunnikassäännöllä käyttämällä neljää jakoväliä. Mikä on likiarvon suhteellinen virhe prosentteina?*

```
#tiedostossa ylioppilasteht3.py
import math

#Palauttaa kysytyn integraalin tarkan arvon.
#AE: true
```

```
def tarkka():
    return math.log(3)-math.log(1)

#Palauttaa funktion f(x)=1/x arvon.
#AE: x!=0
def f(x):
    return 1/x

#Palauttaa puolisuunnikkaiden pinta-alojen summan.
#Saa parametreina alkupisteen, loppupisteen seka jakovalien maaran.
#AE: x0<xn and n>0
def puolisuunnikas(x0, xn, n):
    d=(xn-x0)/(float)(n)        #jakovalin pituus
    ala=0.0                     #puolisuunnikkaiden alojen summa
    for j in range((int)(n)):
        xi=j*d
        xiseur=(j+1)*d
        ala = ala + (f(x0+xi)+f(x0+xiseur))*d/2.0
    return ala

#Palauttaa alan likiarvon suhteellisen virheen.
#Saa parametreina alkupisteen, loppupisteen seka jakovalien maaran.
#AE: x0<xn and n>0
def suhtvirhe(x0, xn, n):
    return 100*(puolisuunnikas(x0, xn, n)-tarkka())/tarkka()

#Tulostaa pinta-alan tarkan arvon, likiarvon ja suht.virheen.
#AE: true
def main():
    print "Tarkka arvo =","%1.5f" % tarkka()
    print "Arvio alalle =","%1.5f" % puolisuunnikas(1.0,3.0,4.0)
    print "Suhteellinen virhe =","%1.2f"%suhtvirhe(1.0,3.0,4.0),"%"

main()
```

```
>>>
Tarkka arvo = 1.09861
Arvio alalle = 1.11667
Suhteellinen virhe = 1.64 %
```

Saatiin siis integraalin $\int_1^3 \frac{1}{x}dx$ tarkaksi arvoksi $1,09861$, likiarvoksi viiden desimaalin tarkkuudella puolisuunnikassäännöllä $1,11667$ sekä suhteelliseksi virheeksi $1,64\%$.

Jos samaa ohjelmaa ajetaan asettamalla jakovälien määräksi tuhat, eli siis antamalla puolisuunnikas-funktiolle (sekä vastaavasti myös suhtvirhe-funktiolle) parametreiksi `puolisuunnikas(1.0, 3.0, 1000.0)`, saadaan huomattavasti tarkempi tulos. Nyt suhteellisen virheen tulostava käsky on hyvä muuttaa muotoon `print "Suhteellinen virhe =","%1.2e"%suhtvirhe(1.0,3.0,4.0),"%"`, jotta ohjelma tulostaa virheen arvon kymmenpotenssimuodossa. Kuten seuraavasta huomataan, tällöin pinta-alan arvio on viiden desimaalin tarkkuudella sama kuin tarkka arvokin.

```
>>>
Tarkka arvo = 1.09861
Arvio alalle = 1.09861
Suhteellinen virhe = 2.70e-05 %
```

4.7.4 Numeerinen integrointi - Simpsonin sääntö

Keskipistesäännön ja puolisuunnikassäännön hyvät puolet saadaan käyttöön Simpsonin säännössä. Siinä suorakulmioiden ja puolisuunnikkaiden pinta-alat korvataan niiden painotetuilla keskiarvoilla siten, että puolisuunnikkaan pinta-ala saa painon $\frac{1}{3}$ ja suorakulmion pinta-ala painon $\frac{2}{3}$. Simpsonin säännön mukaan arvio pinta-alalle on $\frac{d}{3} \cdot (f(a) + 4S_1 + 2S_2 + f(b))$, missä $S_1 = y_1 + y_3 + y_5 + ...$ ja $S_2 = y_2 + y_4 + y_6 + ...$, d on jakovälin pituus, a ja b ovat välin alku- ja loppupisteet ja y_i $(i = 1, 2, 3, ...)$ ovat funktion arvot kunkin jakovälin päätepisteissä. Simpsonin sääntöä käytettäessä jakovälien määrän on oltava parillinen.

Ylioppilastehtävä 4. *(Syksy 2002, tehtävä 15) Normaalijakauman kertymäfunktio on*

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_{0}^{x} e^{-t^2/2} dt.$$

Laske $\Phi(1)$ *Simpsonin säännön avulla jakamalla integroimisväli a) neljään, b) kahdeksaan osaväliin. Esitä näihin laskuihin perustuva arvio tuloksen tarkkuudesta.*

```
#tiedostossa ylioppilasteht4.py
import math

#Palauttaa funktion f(t)=math.exp(-t**2/2.0) arvon.
#AE: t kuuluu reaalilukuihin
def f(t):
    return math.exp(-t**2/2.0)

#Palauttaa likiarvon funktion f(t) maaratylle integraalille.
#Saa parametreiksi integrointivalin [a,b] ja osavalien maaran.
#AE: a ja b kuuluvat reaalilukuihin ja n on parillinen
def simpson(a, b, n):
    d = (b-a)/(float)(n)
    arvio = 0.0
    for i in range(1, (int)(n), 2):
        xi = a + d*i
        arvio = arvio + 4*f(xi)
    for i in range(2, (int)(n), 2):
        xi = a + d*i
        arvio = arvio + 2*f(xi)
    arvio = (arvio + f(a) + f(b))*d/3.0
    return arvio

#Palauttaa tehtavanannon phifunktion arvon.
#Saa parametreiksi integrointivalin [a,b] ja osavalien maaran.
#AE: a ja b kuuluvat reaalilukuihin ja n on parillinen
def phifunktio(a, b, n):
```

```
    return 0.5 + (1/math.sqrt(2*math.pi))*simpson(a, b, n)

#Tulostaa integraalin likiarvon ja phifunktion arvon.
#AE: true
def main():
    print "%1.6f" % simpson(0.0, 1.0, 4.0)
    print "%1.6f" % phifunktio(0.0, 1.0, 4.0)

main()
```

a) Tulokset käyttäen neljää osaväliä:

```
>>>
0.855651
0.841355
```

b) Tulokset käyttäen kahdeksaa osaväliä:

```
>>>
0.855626
0.841345
```

Edellä suoritettujen laskutoimitusten perusteella voidaan olettaa, että lopputuloksen neljä ensimmäistä desimaalia ovat kummassakin tapauksessa oikeita, sillä ne ovat molemmissa tuloksissa samat. Siis $\Phi(1) \approx 0,8413$. Myös Maol-taulukkokirja antaa saman tuloksen.

Tehtävän ratkaisun tarkistaminen analyyttisesti on melko vaikeaa. Sen sijaan tuloksen tarkkuutta voidaan arvioida esimerkiksi muokkaamalla edellistä ohjelmaa siten, että se tulostaa useilla eri osavälien määrillä sekä arvon $\Phi(1)$ että tämän arvon muutoksen edelliseen arvoon verrattuna. Näin saadaan tietää, parantaako rajaton osavälien määrän kasvattaminen tuloksen tarkkuutta. Jätetään tämä tarkastelu kurssin lisämateriaaliin lukuun 4.7.7 Virheet numeerisessa laskennassa.

4.7.5 Lisämateriaali: Differentiaaliyhtälöiden ratkaiseminen numeerisesti

Differentiaaliyhtälöitä käytetään kuvaamaan suureiden muutosta esimerkiksi ajan tai paikan suhteen. Yksinkertaisimmat differentiaaliyhtälöt voi ratkaista analyyttisesti, mutta käytännön sovelluksissa tämä ei yleensä ole mahdollista. Tämän vuoksi niiden ratkaisemiseen on yleensä käytettävä numeerisia menetelmiä. Näistä menetelmistä esitetään tässä luvussa Eulerin menetelmä, keskipistemenetelmä eli niin sanottu modifioitu Eulerin menetelmä sekä kertaluvun 4 Runge-Kutta -menetelmä. Numeerista menetelmää käytettäessä ratkaisufunktiosta saadaan selville vain arvot, ei funktion lauseketta. Useissa sovelluksissa se kuitenkin riittää. [10, 27]

Differentiaaliyhtälö $y'(t) = f(t, y(t)),\ y(t_0) = y_0$ voidaan integroida puolittain välillä $[t_i, t_{i+1}]$, jolloin saadaan $y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))dt$. Differentiaaliyhtälöiden ratkaisemisessa käytettävät numeeriset menetelmät perustuvat integraalin $\int_{t_i}^{t_{i+1}} f(t, y(t))dt$ approksimointiin. Väli $[a, b]$, jolla yhtälö halutaan ratkaista, jaetaan ensin yhtä pitkiin osaväleihin. Merkitään osavälien pituutta eli askelpituutta h ja jakopisteitä $t_0(= a), t_1, t_2, t_3, ...$ Jos jako on tarpeeksi tiheä, ratkaisufunktion $y(t)$ kuvaaja eroaa jokaisella osavälillä vain vähän tangentistaan, jonka kulmakerroin on laskettu menetelmästä riippuen esimerkiksi välin alku- tai keskipisteessä. [10, 27]

Eulerin menetelmä

Eulerin menetelmässä kunkin osavälin alkupisteessä t_i piirretään ratkaisufunktion kuvaajalle tangentti ja edetään tämän tangentin suuntaisesti askelpituuden h verran. Eulerin menetelmä voidaan esittää rekursiivisesti seuraavalla kaavalla: (käytetään tästä lähtien merkintää $y_i = y(t_i)$) [10]

$$\begin{cases} y_0 = y(t_0) \\ y_{i+1} = y_i + f(t_i, y_i) \cdot h \\ t_{i+1} = t_i + h, \quad i = 0, 1, 2, ..., n \end{cases}$$

Ylioppilastehtävä 5. *(Syksy 2003, tehtävä 15) a) Totea, että differentiaaliyhtälön* $y' + 2\sin(x) = y$ *ratkaisu alkuehdolla* $y(0) = 1$ *on* $y(x) = \sin(x) + \cos(x)$. *b) Määritä Eulerin menetelmällä kyseisen ratkaisun likiarvot* y_i $(\approx y(x_i))$ *välillä* $[0, 2]$ *askelpituudella* $h = 0,5$ *sekä laadi taulukko, jossa esiintyvät* $x_i, y(x_i), y_i$ *ja virhe* $y_i - y(x_i)$.

Tässä tehtävässä merkinnät eroavat hieman luvun alussa opituista. Muuttuja x on vastaava kuin muuttuja t. Tässä tehtävässä y_i ja $y(x_i)$ eivät esiinnykään täysin samassa merkityksessä, vaan y_i tarkoittaa ratkaisun likiarvoa ja $y(x_i)$ tarkkaa arvoa.

Tehtävän a-kohdan ratkaisuksi riittää todeta seuraavat asiat: $y'(x) = \cos(x) - \sin(x)$, joten $y'(x) + 2\sin(x) = \cos(x) - \sin(x) + 2\sin(x) = \cos(x) + \sin(x) = y(x)$ sekä $y(0) = \sin(0) + \cos(0) = 1$. Ratkaistaan b-kohta seuraavan ohjelman avulla. Ennen ohjelman kirjoittamista muutetaan vielä tehtävän differentiaaliyhtälö muotoon $y' = f(x, y)$ eli $y' = y - 2\sin(x)$.

```
#tiedostossa ylioppilasteht5.py
import math

#Palauttaa funktion f(ex,yy)=yy-2*math.sin(ex) arvon.
#AE: ex ja yy kuuluvat reaalilukuihin
def f(ex,yy):
    return yy-2*math.sin(ex)

#Tulostaa diff.yht. ratkaisulle likiarvot, tarkat arvot ja virheet.
#Saa parametreina alkuarvot x0, y0, askelpituuden ja valin
#loppupisteen.
#AE: x,y,b kuuluvat reaalilukuihin ja h>0
def euler(x, y, h, b):
    print "xi    yi_tarkka  yi_likiarvo  virhe"
    tarkka = math.cos(x)+math.sin(x)
    virhe = y - tarkka
    print "%.2f %9.6f %10.6f %10.4f"%(x, tarkka, y, virhe)
    n = b/(float)(h) #askelten maara
```

```
    for i in range((int)(n)):
        y = y + f(x,y)*h
        x = x + h
        tarkka = math.cos(x)+math.sin(x)
        virhe = y - tarkka
        print "%.2f %9.6f %10.6f %10.4f"%(x, tarkka, y, virhe)
    return " "

#main
euler(0.0, 1.0, 0.5, 2.0)
```

Tulokset:

```
>>>
xi    yi_tarkka  yi_likiarvo  virhe
0.00  1.000000   1.000000     0.0000
0.50  1.357008   1.500000     0.1430
1.00  1.381773   1.770574     0.3888
1.50  1.068232   1.814391     0.7462
2.00  0.493151   1.724091     1.2309
```

Suoritetaan vielä vertailun vuoksi sama ohjelma antamalla askelpituudelle arvo $h = 0,01$ eli koodissa `euler(0.0, 1.0, 0.01, 2.0)`. Tuloksista on otettu näkyviin vain pieni osa, sillä niitä on yhteensä $2/0,01 = 200$ kappaletta. Koska virhe on tässä huomattavasti pienempi kuin edellä, muutetaan ohjelman tulostuskäsky muotoon `print "%.2f %9.6f %10.6f %14.4e"%(x, tarkka, y, virhe)` eli tulostetaan virheen arvo kymmenpotenssimuodossa.

```
>>>
xi    yi_tarkka  yi_likiarvo  virhe
0.00  1.000000   1.000000     0.0000e+00
0.01  1.009950   1.010000     5.0166e-05
0.02  1.019799   1.019900     1.0133e-04
0.03  1.029546   1.029699     1.5350e-04
```

```
......
1.98  0.519559   0.557317     3.7758e-02
1.99  0.506380   0.544542     3.8162e-02
2.00  0.493151   0.531719     3.8568e-02
```

Tuloksista voidaan huomata, että Eulerin menetelmän antamien arvojen tarkkuus riippuu suuresti askelpituudesta. Pienellä askelpituudella ratkaisufunktion arvot ovat vielä välin päätepisteen $2,0$ lähelläkin melko lähellä oikeita arvoja. Sen sijaan suurella askelpituudella lukua $2,0$ lähestyttäessä arvot poikkeavat jo hyvin paljon oikeasta arvosta. Eulerin menetelmän heikkoutena on, että ratkaisufunktiota korvaavan tangentin kulmakerroin lasketaan kunkin osavälin alkupisteessä, joten se ei välttämättä kuvaa oikein funktion muutosta koko osavälillä. Tämän vuoksi pienen askelpituuden käyttäminen ohjelmassa antaa paremmat tulokset kuin suuren askelpituuden käyttö. [10]

Keskipistemenetelmä eli modifioitu Eulerin menetelmä

Keskipistemenetelmässä yritetään ottaa Eulerin menetelmään verrattuna paremmin huomioon ratkaisufunktion koko osavälillä tapahtuva muutos. Siinä tangentin kulmakerroin $y' = f(t_i, y_i)$ lasketaan nimensä mukaisesti kunkin osavälin keskipisteessä. Keskipistemenetelmä voidaan esittää rekursiivisena kaavana: [10]

$$\begin{cases} y_{i+1} = y_i + f(t_i + \frac{h}{2}, y_i + f(t_i, y_i) \cdot \frac{h}{2}) \cdot h \\ t_{i+1} = t_i + h \end{cases}$$

Esimerkki 26. *Muokataan Eulerin menetelmän yhteydessä kirjoitettua ohjelmaa siten, että se laskee ja tulostaa differentiaaliyhtälön* $y' + 2\sin(x) = y$, $y(0) = 1$ *ratkaisut välillä* $[0, 2]$ *askelpituudella* $h = 0,5$ *sekä keskipistemenetelmällä että Eulerin menetelmällä laskettuna. Vertaillaan lopuksi tulosten tarkkuutta.*

```
#tiedostossa esimerkki26.py
```

```
import math

#Palauttaa funktion f(ex,yy)=yy-2*math.sin(ex) arvon.
#AE: ex ja yy kuuluvat reaalilukuihin
def f(ex,yy):
    return yy-2*math.sin(ex)

#Tulostaa diff.yht. ratkaisulle tarkan arvon ja likiarvot (kp,euler).
#Saa parametreina alkuarvot x0, y0, askelpituuden ja valin
#loppupisteen.
#AE: x,y,b kuuluvat reaalilukuihin ja h>0
def keskipiste_ja_euler(x, y, h, b):
    print "xi    yi_tarkka  yi_kp     yi_euler"
    n = b/(float)(h) #askelten maara
    y_kp = y
    y_euler = y
    for i in range((int)(n)):
        y_kp = y_kp + f(x+h/2.0,y_kp+f(x,y_kp)*h/2.0)*h
        y_euler = y_euler + f(x,y_euler)*h
        x = x + h
        tarkka = math.cos(x)+math.sin(x)
        print "%.2f %9.6f %10.6f %9.6f"%(x, tarkka, y_kp, y_euler)
    return " "

#main
keskipiste_ja_euler(0.0, 1.0, 0.5, 2.0)
```

Seuraavista tuloksista huomataan, että jo suurella askelpituudella $h = 0,5$ keskipistemenetelmä antaa huomattavasti parempia tuloksia kuin Eulerin menetelmä. Siltikään sen avulla ei päästä suurella askelpitudella edes yhden desimaalin tarkkuudella oikeaan tulokseen.

```
>>>
xi    yi_tarkka  yi_kp     yi_euler
0.50  1.357008   1.377596  1.500000
1.00  1.381773   1.437098  1.770574
```

```
1.50   1.068232    1.175933   1.814391
2.00   0.493151    0.677531   1.724091
```

Sama ohjelma antaa askelpituudella $h = 0,01$ testattuna kohdassa $2,0$ tuloksen

```
2.00   0.493151    0.493251   0.531719.
```

Keskipistemenetelmän antama tulos on nyt jo hyvin lähellä ratkaisun tarkkaa arvoa.

Kertaluvun 4 Runge-Kutta -menetelmä

Edellä kuvatuista menetelmistä Eulerin menetelmä on ensimmäistä kertalukua eli siinä virhe on suoraan verrannollinen askelvälin pituuteen. Keskipistemenetelmässä virhe on suoraan verrannollinen askelvälin pituuden neliöön. Käytännössä askelvälin pienentäminen kymmenesosaan tuo siis karkeasti arvioiden tulokseen kaksi oikeaa numeroa lisää. Tässä esiteltävä Runge-Kutta -menetelmä on neljättä kertalukua. Se on paljon käytetty menetelmä ja yleensä riittävän tarkka käytännön laskentaan. [10, 27]

Kertaluvun 4 Runge-Kutta -menetelmässä ratkaisufunktiota approksimoivan tangentin kulmakerroin lasketaan jokaisella osavälillä neljässä kohdassa (merkitään näitä K_1, K_2, K_3, K_4). Varsinainen y_{i+1}-arvon laskemisessa käytettävä kulmakerroin on näiden neljän kulmakertoimen arvon painotettu keskiarvo. Menetelmä voidaan esittää kaavana seuraavasti: [10, 27]

$$y_{i+1} = y_i + \tfrac{h}{6} \cdot (K_1 + 2K_2 + 2K_3 + K_4),$$

missä

$$\begin{cases} K_1 = f(t_i, y_i), \\ K_2 = f(t_i + \frac{h}{2}, y_i + K_1 \cdot \frac{h}{2}), \\ K_3 = f(t_i + \frac{h}{2}, y_i + K_2 \cdot \frac{h}{2}), \\ K_4 = f(t_i + h, y_i + K_3 \cdot h). \end{cases}$$

Menetelmän ohjelmoiminen onnistuu samalla tavoin kuin edellä esitettyjen Eulerin menetelmän ja keskipistemenetelmän ohjelmoiminenkin. Vain seuraavan y-arvon laskemista varten oleva kaava on muutettava Runge-Kutta -menetelmää vastaavaksi. Jätetään tämän menetelmän ohjelmoiminen Harjoitustehtävät-osioon lisätehtäväksi (luvun 4 harjoitustehtävä 35).

4.7.6 Lisämateriaali: Gaussin eliminointi

Gaussin eliminointi on yleisesti käytetty menetelmä lineaaristen yhtälöryhmien ratkaisemisessa. Sen ymmärtäminen vaatii matriisin käsitteen tuntemista, joten seuraavassa kappaleessa esitetään lyhyesti perustiedot matriiseista.

Perustietoa matriiseista

Matriiseja koskevat tiedot perustuvat lähteisiin [39, 50] ja niiden pohjalta laadittuihin esimerkkeihin. Matriisi on alkioita (yleensä reaalilukuja) sisältävä matemaattinen taulukko. Matriisi koostuu vaakasuorista riveistä ja pystysuorista sarakkeista. Jos matriisissa on m kappaletta rivejä ja n kappaletta sarakkeita, sitä kutsutaan $m \times n$-matriisiksi. Jos $m = n$, matriisia kutsutaan neliömatriisiksi. Neliömatriisin alkioita a_{ii}, $i = 1, ..., n$ (alkiot vasemmasta yläkulmasta oikeaan alakulmaan) kutsutaan matriisin lävistäjäalkioiksi. Alla on kuvattuna yleinen $m \times n$-matriisi. Jatkossa tyydytään kuitenkin tarkastelemaan vain 3×3-matriiseja.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Matriiseille voidaan tehdä laskutoimituksia kuten luvuillekin. Kahden matriisin yhteenlasku tapahtuu yksinkertaisesti laskemalla kummankin matriisin vastinalkiot yhteen, kuten alla olevassa esimerkissä on tehty.

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{pmatrix}$$

Matriisien on oltava samaa tyyppiä $m \times n$, jotta yhteenlasku olisi mahdollista suorittaa. Matriisien vähennyslasku tapahtuu vastaavasti. Matriisien kertolaskun voisi loogisesti kuvitella tapahtuvan kertomalla kummankin matriisin vastinalkiot keskenään. Näin ei kuitenkaan ole. Kahden matriisin A ja B tulomatriisi C saadaan seuraavasti:

- Matriisin C ensimmäisen rivin ensimmäinen alkio on matriisin A ensimmäisen rivin ja matriisin B ensimmäisen sarakkeen pistetulo.
- Matriisin C ensimmäisen rivin toinen alkio on matriisin A ensimmäisen rivin ja matriisin B toisen sarakkeen pistetulo.
  $\vdots$
- Matriisin C toisen rivin ensimmäinen alkio on matriisin A toisen rivin ja matriisin B ensimmäisen sarakkeen pistetulo.
- Matriisin C toisen rivin toinen alkio on matriisin A toisen rivin ja matriisin B toisen sarakkeen pistetulo.
  $\vdots$

Matriisien A ja B tulo AB on määritelty vain, jos matriisin A sarakkeiden lukumäärä on sama kuin matriisin B rivien lukumäärä. Havainnollistetaan edellä olevaa selitystä. Matriisitulo

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{pmatrix}$$

tuottaa tulokseksi matriisin

$$= \begin{pmatrix} 1 \cdot 10 + 2 \cdot 13 + 3 \cdot 16 & 1 \cdot 11 + 2 \cdot 14 + 3 \cdot 17 & 1 \cdot 12 + 2 \cdot 15 + 3 \cdot 18 \\ 4 \cdot 10 + 5 \cdot 13 + 6 \cdot 16 & 4 \cdot 11 + 5 \cdot 14 + 6 \cdot 17 & 4 \cdot 12 + 5 \cdot 15 + 6 \cdot 18 \\ 7 \cdot 10 + 8 \cdot 13 + 9 \cdot 16 & 7 \cdot 11 + 8 \cdot 14 + 9 \cdot 17 & 7 \cdot 12 + 8 \cdot 15 + 9 \cdot 18 \end{pmatrix}$$

$$= \begin{pmatrix} 84 & 90 & 96 \\ 201 & 216 & 231 \\ 318 & 342 & 366 \end{pmatrix}$$

On kuitenkin huomattava, että matriisit eivät yleensä kommutoi eli tulomatriisi AB ei yleensä ole sama matriisi kuin tulomatriisi BA. Lasketaan tästä esimerkkinä edellinen matriisitulo uudestaan asettamalla kerrottavat matriisit eri järjestykseen ja huomataan tulomatriisin poikkeavan edellisestä.

$$\begin{pmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 138 & 171 & 204 \\ 174 & 216 & 258 \\ 210 & 261 & 312 \end{pmatrix}$$

Tämän kurssin kannalta oleellista on tietää, että matriiseja voidaan käyttää apuna muun muassa lineaaristen yhtälöryhmien ratkaisussa. Tarkastellaan kolmen muuttujan lineaarista yhtälöryhmää

$$\begin{cases} x - y + z = 2 \\ x - 2y + 3z = 0 \\ 3x - 4y + 2z = -2 \end{cases}$$

Matriisien tulon laskusääntöön perustuen yhtälöryhmä voidaan kirjoittaa matriisimuodossa seuraavasti:

$$\begin{pmatrix} 1 & -1 & 1 \\ 1 & -2 & 3 \\ 3 & -4 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ -2 \end{pmatrix}$$

Gaussin eliminointi

Rajoitutaan tällä kurssilla tarkastelemaan Gaussin eliminointia vain kolmen muuttujan tapauksessa. Siinä on tarkoituksena ratkaista kolmen muuttujan lineaarinen yhtälöryhmä muuttamalla se ensin matriisimuotoon kuten yllä olevassa esimerkissä on tehty. Matriisin lävistäjän alapuolella olevat alkiot nollataan laskemalla matriisin rivejä yhteen. Näin matriisista tulee niin kutsuttu yläkolmiomatriisi. [27]

Gaussin eliminointi sisältää kaksi vaihetta. Eliminointivaiheessa matriisin ensimmäisen rivin alkiot lisätään muihin riveihin kerrottuna sellaisella luvulla, että matriisin ensimmäisen sarakkeen lävistäjän alapuoliset alkiot nollautuvat. Vastaavalla tavalla nollataan myös lävistäjän alapuoliset alkiot muissa sarakkeissa. Myös pystyvektorin $\overline{b}$ alkiot on muistettava päivittää. Takaisinsijoitusvaiheessa lasketaan tuntemattomien muuttujien arvot lopusta alkuun päin. [27]

Tämä menettely on käytännössä sama asia kuin analyyttisen geometrian kurssilla opittu yhteenlaskukeino lineaaristen yhtälöryhmien ratkaisemiseksi. Siinä kerrotaan yhtälöitä sopivilla luvuilla ja lisätään niitä toisiinsa muuttujien eliminoimiseksi. Lopuksi lasketaan muuttujien arvot yksi kerrallaan aivan samoin kuin Gaussin eliminoinnin takaisinsijoitusvaiheessa.

Esimerkki 27 on laadittu lähteen [27] antamien ideoiden ja algoritmin pohjalta.

Esimerkki 27. *Ratkaistaan lineaarinen yhtälöryhmä*

$$\begin{cases} x - y + z = 2 \\ x - 2y + 3z = 0 \\ 3x - 4y + 2z = -2 \end{cases}$$

käyttäen Gaussin eliminointia.

Funktion `gauss(a,b)`parametrit a ja b on annettava vektorimuodossa seuraavasti:
`a = array([[1.0,-1.0,1.0], [1.0,-2.0,3.0], [3.0,-4.0,2.0]])`,
`b = array([2.0,0.0,-2.0])`. Jotta `array()`-käsky toimii, on Numerical Python -pakkaus tuotava käyttöön ohjelman alussa.

```
#tiedostossa esimerkki27.py
from numpy import *

#Tulostaa matriisin a ja vektorin b eri vaiheissa seka
#yhtaloryhman ratkaisun.
#AE: len(a[i]):t yhtasuuria kaikilla i ja len(a)==len(b)
def gauss(a,b):
```

```
    n = len(b)
    #Eliminointivaihe:
    for k in range(n-1):  #kaydaan matriisin sarakkeet lapi
        for i in range(k+1,n): #nollataan alakolmion alkiot
            r = a[i,k]/a[k,k]
            for l in range(k,n):
                a[i,l] = a[i,l] - r*a[k,l]
            b[i] = b[i] - r*b[k]     #paivitetaan vektori b
        print "a =",a
        print "=================="
        print "b =",b
        print "=================="
    #Takaisinsijoitusvaihe:
    x = array([0.0,0.0,0.0]) #alustetaan lista x muuttujille
    x[n-1] = b[n-1]/a[n-1,n-1]     #viimeisen muuttujan arvo
    print "[k, x[k]]"
    print [n-1,x[n-1]]
    for k in range(n-2,-1,-1):      #muiden muuttujien arvot
        s = 0.0
        for l in range(k,n):
            s = s + a[k,l]*x[l]
        x[k] = (b[k]-s)/a[k,k]
        print [k,x[k]]
    return " "

#Tulostaa matriisin a ja vektorin b eri vaiheissa seka
#yhtaloryhman ratkaisun.
#AE: true
def main():
    a = array([[1.0,-1.0,1.0], [1.0,-2.0,3.0], [3.0,-4.0,2.0]])
    b = array([2.0,0.0,-2.0])
    gauss(a,b)

main()
```

Tulokset:

```
>>>
a = [[ 1. -1.  1.]
     [ 0. -1.  2.]
     [ 0. -1. -1.]]
==================
b =  [ 2. -2. -8.]
==================
a = [[ 1. -1.  1.]
     [ 0. -1.  2.]
     [ 0.  0. -3.]]
==================
b =  [ 2. -2. -6.]
==================
[k, x[k]]
[2, 2.0]
[1, 6.0]
[0, 6.0]
```

Tulosten perusteella yhtälöryhmässä tapahtuu eliminoinnin aikana seuraavat muutokset.

$$\begin{cases} x - y + z = 2 \\ x - 2y + 3z = 0 \\ 3x - 4y + 2z = -2 \end{cases} \Leftrightarrow \begin{cases} x - y + z = 2 \\ -y + 2z = -2 \\ -y - z = -8 \end{cases} \Leftrightarrow \begin{cases} x - y + z = 2 \\ -y + 2z = -2 \\ -3z = -6 \end{cases}$$

Takaisinsijoitusvaiheessa ratkaistaan ensin yhtälö $-3z = -6$. Ratkaistaan sitten yhtälö $-y + 2z = -2$ sijoittamalla siihen tulos $z = 2$ ja lopuksi yhtälö $x - y + z = 2$ sijoittamalla siihen edellisen lisäksi myös tulos $y = 6$. Saadaan siis yhtälöryhmän ratkaisuksi

$$\begin{cases} x = 6 \\ y = 6 \\ z = 2 \end{cases}$$

Tulos voidaan tarkistaa sijoittamalla ratkaisut alkuperäisiin yhtälöihin

$$\begin{cases} x - y + z = 2 \\ x - 2y + 3z = 0 \\ 3x - 4y + 2z = -2 \end{cases}$$

```
>>> 6-6+2
2
>>> 6-2*6+3*2
0
>>> 3*6-4*6+2*2
-2
```

Tässä Gaussin eliminoinnin algoritmi ohjelmoitiin oikeastaan harjoituksen vuoksi. Käytännössä lineaariset yhtälöryhmät on kätevintä ratkaista käyttämällä Numerical Pythonin `linalg.solve(a,b)` -toimintoa [25]. Se saa parametreikseen listat `a` ja `b`, kuten edellisen esimerkin `gauss(a,b)` -funktiokin ja palauttaa suoraan lineaarisen yhtälöryhmän ratkaisun. Testataan `linalg.solve(a,b)` -toimintoa ratkaisemalla sen avulla esimerkin 27 lineaarinen yhtälöryhmä.

```
#tiedostossa linearsolve.py
from numpy import *
a = array([[1.0,-1.0,1.0], [1.0,-2.0,3.0], [3.0,-4.0,2.0]])
b = array([2.0,0.0,-2.0])
x = linalg.solve(a,b)
print x

>>>
[ 6.  6.  2.]
```

4.7.7 Lisämateriaali: Virheet numeerisessa laskennassa

Tämän luvun teoria perustuu lähteeseen [24] ja eri menetelmien virhetarkastelut ovat sen sovelluksia.

Numeeriset tulokset eivät yleensä ole tarkkoja. Virheitä aiheuttavat esimerkiksi lukujen katkaisu ja pyöristäminen, merkitsevien numeroiden kumoutuminen sekä kokeellisissa sovelluksissa mittaustulosten epätarkkuus.

Reaaliluvut esitetään tietokoneissa liukulukuina. Liukuluku koostuu kahdesta kokonaisluvusta: mantissasta m ja eksponentista e. Molemmat sisältävät etumerkin. Kaavana liukuluku voidaan esittää muodossa $f = m \cdot 2^e$. Liukuluvut ovat siis vain approksimaatioita reaaliluvuista. Tämä aiheuttaa virhettä laskujen tuloksiin, sillä reaaliluvut muutetaan liukuluvuiksi joko pyöristämällä tai katkaisemalla. Jos ohjelmoija esimerkiksi antaa syötteenä reaaliluvun, joka on tarkkuudeltaan suurempi kuin tietokoneen laskutarkkuus, tietokone pyöristää tai katkaisee luvun siihen tarkkuuteen, jota se pystyy käsittelemään. Samoin tapahtuu myös laskutoimitusten aikana, jos esimerkiksi kertolaskun jälkeen luvun tarkkuus kasvaa lähtöarvojen tarkkuuteen verrattuna.

Tämän kurssin kannalta oleellista on keskittyä tarkastelemaan kumoutumisvirheitä hieman tarkemmin. Kumoutumisvirheitä syntyy pääasiassa kahdenlaisissa tapauksissa. Kahden liukuluvun vähennyslaskussa voi aiheutua virhettä, jos luvut ovat samanmerkkisiä ja itseisarvoiltaan hyvin suuria erotukseensa nähden. Tämä tarkoittaa käytännössä sitä, että luvut ovat itseisarvoiltaan suuria ja hyvin lähellä toisiaan. Riittävän tarkkaa lopputulosta ei välttämättä saada, sillä luvuista ikään kuin häviää merkitseviä numeroita laskun aikana. Kumoutumisvirhettä voi aiheutua myös laskettaessa yhteen itseisarvoiltaan hyvin suuri ja hyvin pieni luku. Tällöin saattaa taas käydä niin, että kumoutumisen vuoksi itseisarvoltaan pienemmällä luvulla ei ole mitään vaikutusta laskun lopputulokseen.

Kumoutumisvirheitä saattaa esiintyä tällä kurssilla ohjelmoiduista numeerisista menetelmistä ainakin numeerista derivaattaa laskettaessa. Myös toisen asteen yhtälön ratkaisukaava on altis kumoutumisvirheille tietyillä kertoimien a, b, ja c arvoilla. Näiden lisäksi tarkastellaan myös Simpsonin säännön alttiutta pyöristysvirheille sekä selvitetään koneen laskutarkkuus eli kone-epsilonin arvo.

Numeerinen derivointi

Tarkastellaan aikaisemmin tällä kurssilla ohjelmoidun likimääräisen keskeisdifferenssin alttiutta virheille ratkaisemalla luvussa 4.7.1 esitetty ylioppilastehtävä 1 uudestaan hieman eri tavoin. Tällä kerralla pienennetään lukua h

arvoon 10^{-16} asti ja lasketaan jokaisella kierroksella likimääräisen ratkaisun virhe tarkkaan arvoon $f'(2) = -0,25$ verrattuna. Lopuksi päätellään, mikä luvun h arvo tuottaa parhaan tuloksen.

Kun funktio $f(x) = \frac{1}{x}$ derivoidaan numeerisesti pisteessä $x = 2$, pitäisi derivaatan arvon tässä pisteessä lähestyä tarkkaa arvoa $f'(2) = -0,25$, kun luvun h arvoa pienennetään. Oheisen ohjelman tulostamasta taulukosta huomataan, että aluksi derivaatan arvo lähestyykin lukua $-0,25$, mutta tarpeeksi pienillä luvun h arvoilla virhettä alkaa syntyä eli derivaatan arvo alkaa yhä enemmän poiketa luvusta $-0,25$. Taulukon lisäksi ohjelma havainnollistaa virheen käyttäytymistä kuvaajan avulla (Kuva 10).

Virhe aiheutuu siitä, että luvun h ollessa tarpeeksi pieni osoittajassa oleva erotus $f(a+h) - f(a-h)$ on itseisarvoltaan hyvin pieni lukujen $f(a+h)$ ja $f(a-h)$ suuruuteen verrattuna. Lisäksi tämä erotus vielä jaetaan itseisarvoltaan pienellä luvulla, minkä vuoksi tuloksen tarkkuus pienenee entisestään. Se, kuinka pienillä luvun h arvoilla tulokseen alkaa syntyä virhettä, riippuu siitä, kuinka jyrkästi funktio kasvaa tai vähenee tarkasteltavan pisteen ympäristössä ja kuinka suuri funktion arvo tässä pisteessä on. Erotus $f(a+h) - f(a-h)$ on itseisarvoltaan tietysti sitä pienempi, mitä loivempi funktion kuvaaja on pisteen $x = a$ ympäristössä. Lisäksi, mitä suurempi funktion arvo $f(a)$ on itseisarvoltaan pisteessä $x = a$, sitä pienempi on erotus $|f(a+h) - f(a-h)|$ lukuihin $|f(a+h)|$ ja $|f(a-h)|$ verrattuna, ja täten, sitä suurempi on myös syntyvä virhe.

```
#tiedostossa numdervirhe.py
from pylab import *
from fonttikoko import*
ax = fonttikokomuutos()

#Palauttaa funktion f(x)=1/x arvon.
#AE: x!=0
def f(x):
    return 1/x

#Palauttaa derivaattafunktion f'(x)=-1.0/(x**2) arvon.
#AE: x!=0
```

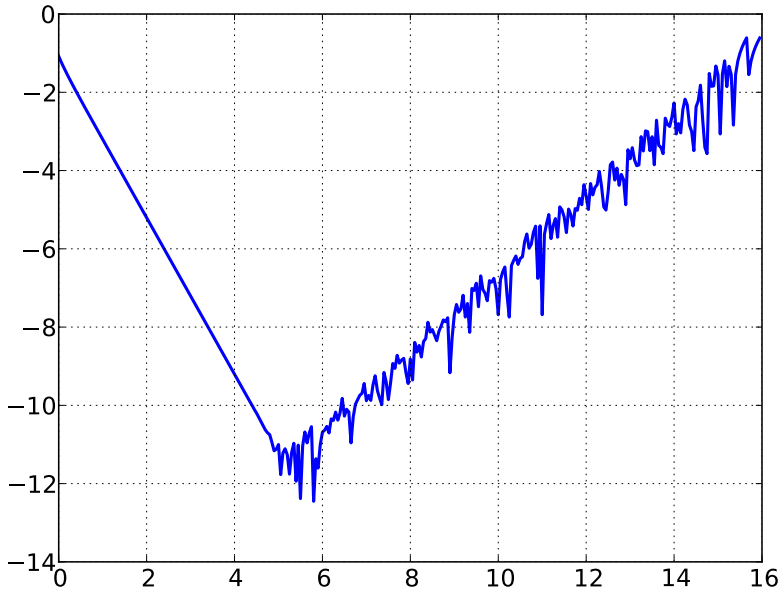
```
def df(x):
    return -1.0/(x**2)

#main
a=2
print "Tarkka arvo: df(a)=",df(a)
print "luku h    derivaatta           virheen itseisarvo"
h=1.0
while 10**(-16)<=h<=1.0:
    derivaatta=(f(a+h)-f(a-h))/(2*h)
    virhe = abs(df(a) - derivaatta)
    print "%1.1e  %10.17f  %10.4e" % (h,derivaatta,virhe)
    h*=0.1
x = arange(0, 16, 0.05)
h = 10**(-x)
kuva = ax.plot(x, log10(abs(df(a) - (f(a+h)-f(a-h))/(2*h))))
setp(kuva, linewidth = 2)
grid()
show()

>>>
Tarkka arvo: df(a)= -0.25
luku h    derivaatta           virheen itseisarvo
1.0e+00  -0.33333333333333337  8.3333e-02
1.0e-01  -0.25062656641604009  6.2657e-04
1.0e-02  -0.25000625015624828  6.2502e-06
1.0e-03  -0.25000006249997758  6.2500e-08
1.0e-04  -0.25000000062558309  6.2558e-10
1.0e-05  -0.25000000000996436  9.9644e-12
1.0e-06  -0.24999999997943326  2.0567e-11
1.0e-07  -0.24999999986841093  1.3159e-10
1.0e-08  -0.24999999848063212  1.5194e-09
1.0e-09  -0.25000002068509264  2.0685e-08
1.0e-10  -0.25000002068509264  2.0685e-08
1.0e-11  -0.25000002068509258  2.0685e-08
```

```
1.0e-12  -0.25002222514558509  2.2225e-05
1.0e-13  -0.24980018054066006  1.9982e-04
1.0e-14  -0.25535129566378578  5.3513e-03
1.0e-15  -0.27755575615628891  2.7556e-02
1.0e-16   0.00000000000000000  2.5000e-01
```

Tuloksista huomataan, että virheen arvo pienenee luvun h arvoon $h = 10^{-5}$ asti. Tämän jälkeen luvun h pienentyessä virhe kasvaa jokaisella kierroksella. Merkitsevien numeroiden kumoutuminen alkaa siis vaikuttaa tuloksiin luvun alussa kuvatulla tavalla, joten lukua h rajattomasti pienentämällä ei päästä aiempaa tarkempaan tulokseen. Päinvastoin, luvun h arvoilla $h = 10^{-15}$ ja $h = 10^{-16}$ derivaatan likiarvo poikkeaa jo hyvin paljon tarkasta arvosta. Paras likiarvo funktion $f(x) = \frac{1}{x}$ derivaatalle saadaan siis, kun luku h on suuruusluokkaa $h = 10^{-5}$.

Virheen edellä kuvattu käyttäytyminen on nähtävissä myös seuraavalla sivulla esitettävästä kuvaajasta (Kuva 10). Edellä esitetyn ohjelman piirtämässä kuvassa 10 esitetään kymmenkantainen logaritmi virheen suuruudesta luvun x funktiona, kun $h = 10^{-x}$ ja luku x kuuluu välille $[0, 16]$. Kuvaaja tukee taulukossa esitettyjä tuloksia: Aluksi lukua h pienennettäessä virheen arvo pienenee nopeasti. Kuitenkin hyvin pienillä luvun h arvoilla ($h < 10^{-7}$) virheen arvo alkaa kasvaa. Virhe on pienimmillään, kun luku h on suuruusluokkaa $h = 10^{-5}$.

Kuva 10: Kymmenkantainen logaritmi virheen suuruudesta luvun x funktiona, kun $h = 10^{-x}$ ja $x \in [0, 16]$



Simpsonin sääntö

Muokataan luvussa 4.7.4 esitettyä ylioppilastehtävän 4 ratkaisua hieman ja tarkastellaan, parantaako rajaton osavälien määrän kasvattaminen saadun tuloksen tarkkuutta. Tulosta on vaikea tarkistaa analyyttisesti, mutta käytetään tuloksen tarkentumisen kriteerinä tuloksen yhä pienenevää etäisyyttä edellisestä tuloksesta. Tuodaan ohjelman käyttöön luvussa 4.7.4 esitetty ohjelmakoodi tiedostosta ylioppilasteht4.py.

```
#tiedostossa simpsonvirhe.py
#Tulostaa listaan osavalien maarat, phifunktion arvot ja
#phifunktion arvojen muutoksen edelliseen verrattuna.
#AE: true
from ylioppilasteht4 import *
def main():
    phi1 = phifunktio(0.0, 1.0, 2)
    print "2 osavalia: phi(1) = ","%1.6f"%phi1
    print "valit    phi(1)             muutos"
```

```
    for i in range(4,1002,2):
        phi2 = phifunktio(0.0, 1.0, i)
        muutos = phi2 - phi1
        phi1 = phi2
        print "%4d %18.12f %14.4e"%(i, phi1, muutos)
main()
```

Testataan ohjelmaa asettamalla osavälien määräksi tuhat:

```
>>>
2 osavalia: phi(1) =  0.841529
valit    phi(1)             muutos
   4     0.841355487857    -1.7356e-04
   6     0.841346841359    -8.6465e-06
   8     0.841345406139    -1.4352e-06
  10     0.841345015888    -3.9025e-07
......
 442     0.841344746069    -1.4433e-15
 444     0.841344746069    -1.1102e-15
 446     0.841344746069    -1.1102e-15
 448     0.841344746069    -8.8818e-16
......
 992     0.841344746069    -1.1102e-16
 994     0.841344746069     6.6613e-16
 996     0.841344746069    -7.7716e-16
 998     0.841344746069    -1.1102e-16
1000     0.841344746069     2.2204e-16
```

Huomataan, että alussa tulos tarkentuu varsin nopeasti eli Simpsonin sääntöä käyttämällä saadaan jo melko pienellä osavälien määrällä useisiin tarkoituksiin riittävän tarkka tulos. Osavälien määrän 446 jälkeen tulos ei osavälien määrää lisäämällä enää huomattavasti muutu. Kun lähestytään tuhatta osaväliä, tuloksen arvo on pysynyt 12 desimaalin tarkkuudella samana. Lievää epävakautta on kuitenkin huomattavissa, sillä välillä tuloksen muutos edelliseen verrattuna kasvaa ja välillä taas pienenee. Tämä saattaa johtua siitä, että suurta osavälien määrää käytettäessä laskussa tarvitaan paljon termejä,

jolloin pyöristysvirheet alkavat vaikuttaa tulokseen niin paljon, ettei täysin tarkkaan tulokseen päästä edes jakovälejä rajattomasti lisäämällä.

Toisen asteen yhtälö

Toisen asteen yhtälön $ax^2+bx+c=0$ ratkaisukaavaa $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ käytettäessä esiintyy merkitsevien numeroiden kumoutumista silloin, kun kertoimet a ja c ovat itseisarvoiltaan hyvin pieniä verrattuna kertoimen b itseisarvoon. Tällöin diskriminantti $b^2-4ac\approx b^2$ ja $\sqrt{b^2-4ac}\approx|b|$.

Testataan virheen syntymistä ratkaisemalla yhtälö $x^2+4\cdot10^6x+0,001=0$. Käytetään tässä hyödyksi aiemmin kurssilla ohjelmoitua toisen asteen yhtälön ratkaisevaa funktiota `ratkaise_yhtalo(a,b,c)`, joka on tallennettu tiedostonimellä `tayratkaisukaava.py`

```
#tiedostossa ratkaisukaavavirhe.py
from tayratkaisukaava import *
a = 1.0
b = 4*(10**6)
c = 0.001
print ratkaise_yhtalo(a,b,c)

>>>
(-2.3283064365386963e-10, -4000000.0)
```

Saatiin siis ratkaisut $x=-2,3283064365386963\cdot10^{-10}$ tai $x=-4000000,0$. Jos ratkaisut sijoitetaan alkuperäiseen yhtälöön, tulokseksi pitäisi tietysti tulla nolla.

```
>>> 0.001+4*(10**6)*(-2.3283064365386963e-10)
    +(-2.3283064365386963e-10)**2
6.8677425384521559e-05
>>> 0.001+4*(10**6)*(-4000000.0)+(-4000000.0)**2
0.001953125
```

Molemmat tulokset poikkeavat nollasta, luku $0,001953125$ jopa melko paljon. Voidaan päätellä, että ratkaisuissa esiintyy jonkin verran virhettä merkitsevien numeroiden kumoutumisen vuoksi.

Kone-epsilon

Koneen laskutarkkuutta kuvaa vakio nimeltä kone-epsilon. Se on pienin luku, joka lisättäessä ykköseen antaa suuremman arvon kuin yksi. Lasketaan kone-epsilonille arvo seuraavan ohjelman avulla.

```
#tiedostossa koneepsilon.py
epsilon = 1.0
while 1.0 + epsilon > 1.0:
    epsilon /= 2
    talletus = epsilon
print "%1.4e" % talletus

>>>
1.1102e-16
```

Saatiin siis kone-epsilonin arvoksi luku $1.1102 \cdot 10^{-16}$.

4.8 Geometria - Johdatus olio-ohjelmointiin

Perehdytään kurssin lopuksi vielä hieman olio-ohjelmointiin. Tällä kurssilla on tavoitteena muodostaa yleiskuva siitä, mitä olio-ohjelmointi on sekä oppia käyttämään sitä yksinkertaisissa ohjelmissa.

4.8.1 Mitä olio-ohjelmointi on?

Luku 4.8.1 perustuu lähteen [14] tietoihin ja niiden sovelluksiin. Olio-ohjelmointi saattaa aluksi vaikuttaa monimutkaiselta, mutta todellisuudessa se on kuitenkin paras ja yksinkertaisin tapa kirjoittaa ohjelmia. Olio-ohjelmointi mahdollistaa sen, että ohjelmissa asiat voidaan esittää lähes samalla tavoin kuin reaalimaailmassakin. Jos halutaan kirjoittaa esimerkiksi minun sinisen Volvoni toimintaa kuvaava ohjelma, voidaan reaalimaailman esine ”minun sininen Volvoni” korvata ohjelmassa ohjelmisto-oliolla, jolla on ominaisuuksia eli attribuutteja sekä toimintoja eli metodeja.

Olio-ohjelmointi on siis käytännössä reaalimaailman ilmiöiden mallintamista tietokoneen avulla. Reaalimaailmassa kaikki esineet kuuluvat johonkin luokkaan. Esimerkiksi minun sininen Volvoni on auto, Frutti di Mare on

pizza, Pertti on ihminen ja niin edelleen. Luokan käsite on keskeinen myös olio-ohjelmoinnissa. Ohjelmassa voidaan määritellä esimerkiksi luokka Auto ja sille yksi tai useampia ominaisuuksia ja toimintoja. Tälle Auto-luokalle voidaan nyt luoda yksi tai useampia olioita, esimerkiksi minun sininen Volvoni, valkoinen Fiat ja punainen Toyota, jotka ovat kaikki autoja mutta eroavat yksityiskohdiltaan toisistaan. Oliot siis edustavat luokkaansa eli ne ovat luokkansa ilmentymiä.

4.8.2 Neliöolio, ympyräolio ja pallo-olio

Tässä luvussa opitaan määrittelemään luokkia ja luomaan niille olioita. Esimerkkeinä käytetään geometriasta tuttuja yksinkertaisia kuvioita ja kappaleita (neliö, ympyrä ja pallo).

Esimerkki 28. *Ohjelmoidaan ensin neliön ja ympyrän pinta-alat sekä pallon tilavuus jo opittujen taitojen avulla, olioita käyttämättä.*

```
#tiedostossa esimerkki28.py
import math

#Palauttaa nelion pinta-alan.
#AE: sivunpituus >= 0
def nelio(sivunpituus):
    return sivunpituus*sivunpituus

#Palauttaa ympyran pinta-alan.
#AE: y_sade >= 0
def ympyra(y_sade):
    return math.pi*y_sade**2

#Palauttaa pallon tilavuuden.
#AE: p_sade >= 0
def pallo(p_sade):
    return (4.0/3.0)*math.pi*p_sade**3

#main
```

```
print nelio(2)
print ympyra(1)
print pallo(1)

>>>
4
3.14159265359
4.18879020479
```

Tässä vaiheessa saattaa herätä kysymys, miksi vastaava ohjelma sitten täytyisi toteuttaa olioiden avulla. Näin lyhyessä ohjelmassa olioiden käyttö ei välttämättä olekaan tarpeellista, mutta neliö-, ympyrä- ja pallo-olion luominen toimii hyvänä ja yksinkertaisena esimerkkinä olio-ohjelmoinnin perusasioista. Pidemmissä ohjelmissa olio-ohjelmoinnin hyödyt pääsevät todella oikeuksiinsa. Oliopohjaisesti toteutettuun ohjelmaan on helppo tehdä pieniä muutoksia joutumatta kuitenkaan muokkaamaan koko ohjelmaa. Lisäksi esimerkiksi oliopohjaiselle pelille on helppo tehdä jatko-osia suoraan alkuosan luokkiin pohjautuen.

Esimerkki 29. *Ohjelmoidaan neliöolio, ympyräolio ja pallo-olio.*

Koodin jälkeen on selitetty yksityiskohtaisesti, mitä mikäkin koodirivi tekee. Tässä (kuten myöhemmissäkin luvun 4.8 ohjelmissa) on koodin lyhentämiseksi alkuehto jätetty kirjoittamatta kaikille niille metodeille, joille se olisi true.

```
#tiedostossa esimerkki29.py
import math

class Nelio(object):
    """Nelio-olion konstruktori"""

    #Nelion alustaja.
    #AE: sivunpituus >= 0
    def __init__(self, sivunpituus):
        self.__sivunpituus = sivunpituus
```

```
    #Palauttaa nelion sivunpituuden.
    def anna_sivu(self):
        return self.__sivunpituus

    #Palauttaa nelion pinta-alan.
    def pinta_ala(self):
        ala = self.__sivunpituus*self.__sivunpituus
        return ala

    #Palauttaa nelion lavistajan pituuden.
    def lavistaja(self):
        lavistaja = math.sqrt(2)*self.__sivunpituus
        return lavistaja

class Ympyra(object):
    """Ympyraolion konstruktori"""

    #Ympyran alustaja.
    #AE: sade >= 0
    def __init__(self, sade):
        self.__sade = sade

    #Palauttaa ympyran sateen.
    def anna_sade(self):
        return self.__sade

    #Palauttaa ympyran pinta-alan.
    def pinta_ala(self):
        return math.pi*self.__sade**2

    #Palauttaa ympyran piirin pituuden.
    def piiri(self):
        return 2*math.pi*self.__sade
```

```
class Pallo(object):
    """Pallo-olion konstruktori"""

    #Pallon alustaja.
    #AE: sade >= 0
    def __init__(self, sade):
        self.__sade = sade

    #Palauttaa pallon sateen.
    def anna_sade(self):
        return self.__sade

    #Palauttaa pallon pinta-alan.
    def pinta_ala(self):
        return 4*math.pi*self.__sade**2

    #Palauttaa pallon tilavuuden.
    def tilavuus(self):
        return math.pi*self.__sade**3*(4/3.0)

#main
print "Nelio:"
sivunpituus = int(raw_input("Anna sivunpituus: "))
nelioolio = Nelio(sivunpituus)
print "Nelion pinta-ala on ", nelioolio.pinta_ala(),
print " ja nelion lavistajan pituus on ", nelioolio.lavistaja(),".\n"

print "Ympyra:"
sade = float(raw_input("Anna sade: "))
ympyraolio = Ympyra(sade)
print "Ympyran pinta-ala on ", ympyraolio.pinta_ala(),
print " ja ympyran kehan pituus on ", ympyraolio.piiri(),".\n"

print "Pallo:"
sade = float(raw_input("Anna sade: "))
```

```
palloolio = Pallo(sade)
print "Pallon pinta-ala on ", palloolio.pinta_ala(),
print " ja pallon tilavuus on ", palloolio.tilavuus(),".\n"
```

Tulokset:

```
>>>
Nelio:
Anna sivunpituus: 2
Nelion pinta-ala on  4  ja nelion lavistajan pituus on
2.82842712475 .

Ympyra:
Anna sade: 1
Ympyran pinta-ala on  3.14159265359  ja ympyran kehan pituus on
6.28318530718 .

Pallo:
Anna sade: 1
Pallon pinta-ala on  12.5663706144  ja pallon tilavuus on
4.18879020479 .
```

Alla on selitetty neliöolioon liittyvä teoria asia kerrallaan. Ympyrä- ja pallooliot luodaan vastaavasti.

1. Olion luomiseksi on ensin määriteltävä sille luokka. Yllä olevassa ohjelmassa luodaan ensimmäisenä neliöluokka rivillä `class Nelio(object):`. Tässä vaiheessa ei siis vielä luoda oliota. Luokan nimi on tapana kirjoittaa isolla alkukirjaimella. Luokan nimen jälkeisissä sulkeissa oleva sana `object` kertoo sen, että luokka perustuu Pythonin sisäänrakennettuun tyyppiin. Sulkeissa voisi olla myös minkä tahansa edeltävän luokan nimi, johon uusi luokka perustuu, mutta tähän palataan luvussa 4.8.3. Rivi `"""Nelio-olion konstruktori"""` on kommentti, joka kuvailee, minkälaisten olioiden luomiseen luokkaa voidaan käyttää.

2. Tämän jälkeen koodissa tuleekin vastaan ensimmäinen metodi:

```
def __init__(self, sivunpituus):
    self.__sivunpituus = sivunpituus
```

`__init__` -metodi on konstruktoriksi kutsuttu erityismetodi. Myöhemmin ohjelmassa sitä kutsutaan automaattisesti heti, kun luokalle luodaan uusi olio. Huomataan, että `__init__` -metodilla on kaksi parametria: `self` ja `sivunpituus`. Näistä ensimmäinen (parametri `self`) on oltava jokaisella oliometodilla, sillä se antaa metodille mahdollisuuden viitata olioon itseensä. Metodin jälkimmäisellä rivillä lause `self.__sivunpituus = sivunpituus` luo uudelle oliolle attribuutin `sivunpituus` (lauseen vasen puoli) ja asettaa sen arvoksi parametrin `sivunpituus` (lauseen oikea puoli). Lauseen oikeanpuoleinen `sivunpituus` on siis sama kuin metodin nimen jälkeisissä sulkeissa oleva `sivunpituus`. Metodin jälkimmäinen rivi voitaisiin kirjoittaa myös ilman kahta alaviivaa: `self.sivunpituus = sivunpituus`. Syy, miksi näin ei kuitenkaan tässä tehdä, selviää kohdassa 6.

Parametrien `self` ja `sivunpituus` lisäksi `__init__` -metodi voisi saada parametrikseen vielä kuvion tai kappaleen sijainnin koordinaatistossa. Tämän parametrin arvo voisi olla ympyrällä ja pallolla keskipiste ja neliöllä esimerkiksi keskipiste tai vasen alanurkka. Tässä vaiheessa ohjelmassa ei kuitenkaan käytetä näitä parametreja, jotta ohjelma pysyisi mahdollisimman yksinkertaisena ymmärtää.

3. Muut Nelio-luokan metodit näyttävät ulkoasultaan hyvin samanlaisilta kuin aiemmin kurssilla opitut funktiot. Itse asiassa metodit ovat käytännössä olioihin liitettyjä funktioita. Niitä kirjoitettaessa on muistettava, että ne saavat aina vähintään yhden parametrin, `self`.

4. Tähän mennessä ei siis ole vielä luotu yhtään oliota. Siirrytään tarkastelemaan ohjelman main-osiota. Asetuslauseella `nelioolio = Nelio(sivunpituus)` luokasta Nelio luodaan neliöolio, jonka nimeksi eli tunnisteeksi annetaan `nelioolio`. Käytännössä muuttujasta nimeltä `nelioolio` luodaan viittaus neliöolioon, joka syntyy tietokoneen muistiin asetuslauseen yhtey-

dessä. Tässä tulee esille tärkeä ero oliotyyppisten muuttujien ja perustyyppisten muuttujien välillä: perustyyppinen muuttuja sisältää itse muuttujan arvon, mutta oliotyyppinen muuttuja sisältää viittauksen olioon. Asetuslauseen seurauksena uusi neliöolio saa attribuutikseen sen sivunpituuden arvon, jonka käyttäjä rivillä `sivunpituus = int(raw_input("Anna sivunpituus: "))` antaa.

5. Neliöoliolla on nyt kaikki Nelio-luokkaan kirjoitetut metodit. Esimerkiksi metodia `def pinta_ala(self)` voidaan kutsua seuravasti: `nelioolio.pinta_ala()`. Tällöin ohjelma tekee sen, mitä metodissa `def pinta_ala(self)` määrätään eli palauttaa neliön pinta-alan tietyllä sivunpituuden arvolla.

6. Palataan kohdassa 2 esitettyyn kysymykseen: Miksi `__init__` -metodin jälkimmäinen rivi kirjoitetaan nimenomaan muodossa `self.__sivunpituus = sivunpituus` käyttäen kahta alaviivaa? Tähän liittyy kaksi uutta käsitettä: tiedon kapselointi ja suojausmääreet. Tiedon kapselointi tarkoittaa rajoitettua pääsyä olion sisäisiä ominaisuuksia koskeviin tietoihin. Käyttäjällä on siis lupa kommunikoida olion kanssa vain olion metodien parametrien ja palautusarvojen kautta. Esimerkiksi neliön sivunpituuden pyytäminen komennolla `nelioolio.sivunpituus` on huono tapa ohjelmoida. Sen sijaan sama asia tehdään oikein pyytämällä sivunpituutta metodin `def anna_sivu(self)` kautta komennolla `nelioolio.anna_sivu()`.

   Suojausmääreiden avulla pystytään estämään edellä mainitun väärän ohjelmointitavan käyttö. Kun esimerkiksi attribuutti `sivunpituus` suojataan private-määreellä, se näkyy enää vain sen luokan sisällä, jossa se on määritelty. Kutsuttaessa Nelio-luokan ulkopuolella sivunpituutta komennolla `nelioolio.sivunpituus`, tuloksena on virheilmoitus: `AttributeError:'Nelio' object has no attribute 'sivunpituus'`. Pythonin syntaksissa private-määre saadaan kirjoitettua laittamalla kaksi alaviivaa attribuutin nimen eteen eli `__sivunpituus`.

Luvussa 4.8.1 opittiin, että olio on luokkansa ilmentymä. Esimerkin 29 ohjelmassa kullakin luokalla on vain yksi ilmentymä eli jokaisesta luokasta luotiin vain yksi olio. Kuitenkin osa olio-ohjelmoinnin voimasta piilee siinä, että luokalla voi olla useita ilmentymiä eli jokaisesta luokasta voidaan luoda useita olioita.

Esimerkki 30. *Luodaan kolme oliota samasta luokasta.*

Koodin jälkeen selitetään taas teoriaa tarkemmin.

```
#tiedostossa esimerkki30.py
import math

class Pallonmuotoinen(object):
    """Pallonmuotoisten olioiden konstruktori"""

    #Pallonmuotoisten olioiden alustaja.
    #AE: sade >= 0 ja vari on jarkevasti nimetty
    def __init__(self, sade, vari):
        self.__sade = sade
        self.__vari = vari

    #Palauttaa olion varin.
    def anna_vari(self):
        return self.__vari

    #Palauttaa olion tilavuuden.
    def tilavuus(self):
        tilavuus = math.pi*self.__sade**3*(4/3.0)
        return tilavuus

#main
puolukkaolio = Pallonmuotoinen(0.2, "punainen")
mustikkaolio = Pallonmuotoinen(0.3, "sininen")
appelsiiniolio = Pallonmuotoinen(3.0, "oranssi")
```

```
print "Puolukan vari ja tilavuus (cm**3):"
print puolukkaolio.anna_vari(), ",", puolukkaolio.tilavuus()
print "Mustikan vari ja tilavuus (cm**3):"
print mustikkaolio.anna_vari(), ",", mustikkaolio.tilavuus()
print "Appelsiinin vari ja tilavuus (cm**3):"
print appelsiiniolio.anna_vari(), ",", appelsiiniolio.tilavuus()

>>>
Puolukan vari ja tilavuus (cm**3):
punainen , 0.0335103216383
Mustikan vari ja tilavuus (cm**3):
sininen , 0.113097335529
Appelsiinin vari ja tilavuus (cm**3):
oranssi , 113.097335529
```

Ohjelmassa määritellään jo opitulla tavalla luokka Pallonmuotoinen ja sille konstruktorimetodi kolmella parametrilla (`self`, `sade` ja `vari`) sekä kaksi muuta metodia. Main-osion ensimmäisellä rivillä luodaan puolukkaolio luokasta Pallonmuotoinen. Samalla annetaan parametreille `sade` ja `vari` arvot `0.2` ja `punainen`. Main-osion kahdella seuraavalla rivillä tulee esille varsinainen uusi asia. Rivillä `mustikkaolio = Pallonmuotoinen(0.3, "sininen")` luodaan mustikkaolio samasta Pallonmuotoinen-luokasta kuin puolukkaoliokin. Vain parametreilla on eri arvot. Vastaavasti rivillä `appelsiiniolio = Pallonmuotoinen(3.0, "oranssi")` luodaan myös uusi olio, appelsiiniolio, yksityiskohtaisilla parametrien arvoilla luokasta Pallonmuotoinen.

4.8.3 Platonin kappaleet olio-ohjelmoinnilla

Platonin kappaleet ovat kuperia säännöllisiä monitahokkaita eli niiden tahkot ovat keskenään yhteneviä säännöllisiä monikulmioita ja niiden kaikissa kärjissä kohtaa yhtä monta tahkoa. Niitä on olemassa vain viisi kappaletta: tetraedri, heksaedri eli kuutio, oktaedri, dodekaedri ja ikosaedri. [13]

Esimerkki 31. *Lasketaan Platonin kappaleiden pinta-aloja ja tilavuuksia olio-ohjelmointia hyödyntäen.*

Teoria ja uudet asiat on taas selitetty koodin jälkeen.

```
#tiedostossa esimerkki31.py
import math

class Monitahokas(object):
    """Monitahokasolion konstruktori"""

    #Monitahokkaan alustaja.
    #AE: sarma >= 0
    def __init__(self, sarma):
        self.__sarma = sarma

    #Palauttaa sarman pituuden.
    def anna_sarma(self):
        return self.__sarma

    #Palauttaa tasasivuisen kolmion pinta-alan.
    def kolmion_pinta_ala(self):
        return self.__sarma**2*math.sqrt(3)/4

    #Palauttaa nelion pinta-alan.
    def nelion_pinta_ala(self):
        return self.__sarma**2

    #Palauttaa saannollisen viisikulmion pinta-alan.
    def saann_viisikulmion_pinta_ala(self):
        return self.__sarma**2*math.sqrt(25+10*math.sqrt(5))/4

    #Palauttaa monitahokkaan pinta-alan.
    def pinta_ala(self): abstract

    #Palauttaa monitahokkaan tilavuuden.
```

```
    def tilavuus(self): abstract

class Tetraedri(Monitahokas):
    """Tetraedriolion konstruktori"""

    #Palauttaa tetraedrin pinta-alan.
    def pinta_ala(self):
        return 4*self.kolmion_pinta_ala()

    #Palauttaa tetraedrin tilavuuden.
    def tilavuus(self):
        return self.anna_sarma()**3*math.sqrt(2)/12

class Heksaedri(Monitahokas):
    """Heksaedriolion konstruktori"""

    #Palauttaa heksaedrin pinta-alan.
    def pinta_ala(self):
        return 6*self.nelion_pinta_ala()

    #Palauttaa heksaedrin tilavuuden.
    def tilavuus(self):
        return self.anna_sarma()**3

class Oktaedri(Monitahokas):
    """Oktaedriolion konstruktori"""

    #Palauttaa oktaedrin pinta-alan.
    def pinta_ala(self):
        return 8*self.kolmion_pinta_ala()

    #Palauttaa oktaedrin tilavuuden.
    def tilavuus(self):
        return self.anna_sarma()**3*math.sqrt(2)/3
```

```
class Dodekaedri(Monitahokas):
    """Dodekaedriolion konstruktori"""

    #Palauttaa dodekaedrin pinta-alan.
    def pinta_ala(self):
        return 12*self.saann_viisikulmion_pinta_ala()

    #Palauttaa dodekaedrin tilavuuden.
    def tilavuus(self):
        return self.anna_sarma()**3*(15+7*math.sqrt(5))/4

class Ikosaedri(Monitahokas):
    """Ikosaedriolion konstruktori"""

    #Palauttaa ikosaedrin pinta-alan.
    def pinta_ala(self):
        return 20*self.kolmion_pinta_ala()

    #Palauttaa ikosaedrin tilavuuden.
    def tilavuus(self):
        return 5*self.anna_sarma()**3*(3+math.sqrt(5))/12

#main
print """Valitse monitahokas:

         0 = Tetraedri
         1 = Heksaedri
         2 = Oktaedri
         3 = Dodekaedri
         4 = Ikosaedri
         5 = Lopeta ohjelma

         """

choice = raw_input("Valinta on : ")
```

```
while choice != "5":

    if choice == "0":
        print "Lasketaan tetraedrin pinta-ala ja tilavuus."
        sarma = float(raw_input("Anna sarman pituus: "))
        monitahokas = Tetraedri(sarma)
        print "Pinta-ala: ", monitahokas.pinta_ala()
        print "Tilavuus: ", monitahokas.tilavuus()
        choice = raw_input("\nValinta on: ")

    elif choice == "1":
        print "Lasketaan heksaedrin pinta-ala ja tilavuus."
        sarma = float(raw_input("Anna sarman pituus: "))
        monitahokas = Heksaedri(sarma)
        print "Pinta-ala: ", monitahokas.pinta_ala()
        print "Tilavuus: ", monitahokas.tilavuus()
        choice = raw_input("\nValinta on: ")

    elif choice == "2":
        print "Lasketaan oktaedrin pinta-ala ja tilavuus."
        sarma = float(raw_input("Anna sarman pituus: "))
        monitahokas = Oktaedri(sarma)
        print "Pinta-ala: ", monitahokas.pinta_ala()
        print "Tilavuus: ", monitahokas.tilavuus()
        choice = raw_input("\nValinta on: ")

    elif choice == "3":
        print "Lasketaan dodekaedrin pinta-ala ja tilavuus."
        sarma = float(raw_input("Anna sarman pituus: "))
        monitahokas = Dodekaedri(sarma)
        print "Pinta-ala: ", monitahokas.pinta_ala()
        print "Tilavuus: ", monitahokas.tilavuus()
        choice = raw_input("\nValinta on: ")
```

```
    elif choice == "4":
        print "Lasketaan ikosaedrin pinta-ala ja tilavuus."
        sarma = float(raw_input("Anna sarman pituus: "))
        monitahokas = Ikosaedri(sarma)
        print "Pinta-ala: ", monitahokas.pinta_ala()
        print "Tilavuus: ", monitahokas.tilavuus()
        choice = raw_input("\nValinta on: ")

    else:
        print "Virheellinen valinta."
        choice = raw_input("\nValinta on: ")
```

Testataan ohjelmaa:

```
>>>
Valitse monitahokas:

         0 = Tetraedri
         1 = Heksaedri
         2 = Oktaedri
         3 = Dodekaedri
         4 = Ikosaedri
         5 = Lopeta ohjelma


Valinta on : 0
Lasketaan tetraedrin pinta-ala ja tilavuus.
Anna sarman pituus: 2
Pinta-ala:  6.92820323028
Tilavuus:  0.942809041582

Valinta on: 1
Lasketaan heksaedrin pinta-ala ja tilavuus.
Anna sarman pituus: 2
Pinta-ala:  24.0
Tilavuus:  8.0
```

```
Valinta on: 2
Lasketaan oktaedrin pinta-ala ja tilavuus.
Anna sarman pituus: 2
Pinta-ala:  13.8564064606
Tilavuus:  3.77123616633

Valinta on: 3
Lasketaan dodekaedrin pinta-ala ja tilavuus.
Anna sarman pituus: 2
Pinta-ala:  82.5829152283
Tilavuus:  61.304951685

Valinta on: 4
Lasketaan ikosaedrin pinta-ala ja tilavuus.
Anna sarman pituus: 2
Pinta-ala:  34.6410161514
Tilavuus:  17.453559925

Valinta on: 5
```

1. Ohjelman ensimmäinen luokka Monitahokas näyttää melko samanlaiselta kuin muutkin tähän mennessä määritellyt luokat. Sen sijaan muut luokat Tetraedrista eteenpäin poikkeavat määrittelyriviltään ja metodeiltaan edellisistä. Ne perustuvat Pythonin sisäänrakennetun tyypin `object` sijaan yliluokalle Monitahokas, mikä ilmaistaan ohjelmassa rivin `class Tetraedri(Monitahokas):` avulla (heksaedrille ynnä muille vastaavasti). Tätä kutsutaan perinnäksi. Kun esimerkiksi aliluokka Tetraedri perii yliluokan Monitahokas, se saa käyttöönsä kaikki Monitahokas-luokan metodit. Luokalla Tetraedri ei tarvitse olla esimerkiksi `__init__` -metodia ollenkaan, sillä se perii vastaavan metodin Monitahokas-luokalta. Aliluokalle voidaan kirjoittaa perittyjen metodien lisäksi myös omia metodeja. Perintä on erittäin kätevä keino yksinkertaistaa pitkiä ohjelmakoodeja. Tässäkään ohjelmassa ei tarvitse kirjoittaa esimerkiksi tasasivuisen kolmion pinta-alan laskevaa metodia

erikseen useaan aliluokkaan.

2. Yliluokan Monitahokas kaksi viimeistä metodia ovat abstrakteja metodeja. Abstraktista metodista esitellään aina vain otsikko, ei toteutusta. Abstrakti luokka on sellainen luokka, jossa on ainakin yksi abstrakti metodi. Tässä Monitahokas on siis abstrakti luokka. Abstraktilla luokalla ei voi olla ilmentymää. Ne ovat kuitenkin käteviä useiden ohjelmien yliluokkina. Ne toimivat ikään kuin luokkapohjina, joille voidaan helposti määritellä konkreettisia aliluokkia, joissa abstraktit metodit on toteutettu ja joilla voi olla ilmentymiä. [14]

3. Luokassa Tetraedri, kuten muissakin aliluokissa, on metodit `pinta_ala(self)` ja `tilavuus(self)`. Nämä ovat täsmälleen samannimiset kuin Monitahokas-luokan abstraktit metodit. Aliluokissa on siis toteutettu abstraktit metodit ylikirjoittamalla ne. Ylikirjoittaminen tarkoittaa, että yliluokan metodi kirjoitetaan aliluokassa uudelleen samalla otsikolla, mutta toteutetaan se eri tavalla. Aliluokassa voi ylikirjoittaa kaikkia yliluokan metodeja, muitakin kuin abstrakteja.

4. Ohjelman jokaisesta aliluokasta luodaan olio aiemmin opitulla tavalla. Perintä ei siis mitenkään vaikuta olion luomiseen. Huomion arvoista on, että jokainen olio on luotu käskyllä `monitahokas = Luokan_nimi(sarma)`. Miten monitahokas-nimiselle tetraedrioliolle käy, kun luodaan monitahokas-niminen heksaedriolio? Entä miten tälle heksaedrioliolle käy, kun luodaan monitahokas-niminen oktaedriolio?

   Pythonissa on sisäänrakennettuna niin sanottu automaattinen roskienkeruu -toiminto, joka mahdollistaa saman muuttujan nimen kierrättämisen. Roskienkeruu toimii seuraavasti: Rivillä `monitahokas = Tetraedri(sarma)` luodaan viittaus monitahokas-nimisestä muuttujasta tetraedriolioon. Kun rivillä `monitahokas = Heksaedri(sarma)` luodaan viittaus monitahokas-nimisestä muuttujasta heksaedriolioon, samalla edellinen viittaus

poistuu. Pythonissa edellinen muuttujasta olioon luotu viittaus siis poistuu aina automaattisesti, kun samasta muuttujasta luodaan viittaus johonkin toiseen olioon. Nyt tetraedriolioon ei ole osoitettu yhtään viittausta (olettaen, että siihen ei ollut aiemmin luotu viittauksia muista muuttujista kuin monitahokkaasta). Muistitilan vapauttamiseksi Pythonin automaattinen roskienkeruu hävittää nyt ”vapaana kelluvan” tetraedriolion. Heksaedrioliolle käy vastaavasti, kun monitahokas-nimisestä muuttujasta luodaan viittaus oktaedriolioon. [32, 35]

4.9 Harjoitustehtäviä

Kirjoita ohjelmat skripteinä eli toisin sanoen kirjoita ohjelmakoodit tekstieditorilla ja tallenna ne Python-tiedostoon nimellä luku4_harjoitus'tehtävänumero'.py (esimerkiksi luvun 4 harjoitustehtävän 2 tiedostonimi on luku4_harjoitus2.py). Suorita ohjelmat tästä tiedostosta.

1. Kirjoita ohjelma, joka laskee ja tulostaa lukujonon $a_n = \frac{\sqrt{n}}{n}$ $(n = 1, 2, 3, ...)$ kaikki lukua 0.35 suuremmat jäsenet.

2. Kirjoita ohjelma, joka laskee ja tulostaa geometrisesta lukujonosta $a_n = 4 \cdot 3^{n-1}$ $(n = 1, 2, 3, ...)$ niin monta ensimmäistä termiä, kuin käyttäjä haluaa.

3. Kirjoita ohjelma, joka laskee ja tulostaa kaksinumeroisten parillisten positiivisten kokonaislukujen lukumäärän ja summan.

4. Kirjoita ohjelma, joka kysyy käyttäjältä kaksi lukua ja laskee niiden suurimman yhteisen tekijän Eukleideen algoritmilla. Testaa ohjelmaa ratkaisemalla seuraava tehtävä: Määritä Eukleideen algoritmilla lukujen 34068 ja 14630 suurin yhteinen tekijä syt(34068, 14630) (yo-tehtävä K2000 teht. 15, osa tehtävästä) [19].

5. Kirjoita ohjelma, joka tulostaa kaikki alkuluvut käyttäjän antamaan ylärajaan asti.

6. Kirjoita ohjelma, joka laskee ja tulostaa käyttäjän antaman luonnollisen luvun kaikki aidot jakajat sekä aitojen jakajien summan. Luku k on luvun n aito jakaja, jos $\frac{n}{k} \in N$ ja jos $k < n$. [44]

7. Kirjoita ohjelma, joka laskee ja tulostaa vektorien $\bar{a} = \bar{i} + 4\bar{j} - \bar{k}$ ja $\bar{b} = 11\bar{i} - 6\bar{j} + \bar{k}$ summan ja pistetulon sekä molempien vektorien pituudet.

8. Määritä ohjelmoimalla vektorien $\bar{a} = 2\bar{i} + 5\bar{j}$ ja $\bar{b} = \bar{i} - 2\bar{j}$ summavektori ja summavektorin suuntainen yksikkövektori. (yo-tehtävä K2009 teht. 3a, tähän lisätty sana ohjelmoimalla) [19]

9. Kirjoita ohjelma, joka laskee ja tulostaa luvuille $x = \left[0, \frac{1}{4}\pi, \frac{1}{3}\pi, \frac{1}{2}\pi\right]$ laskutoimitukset $\sin(x)$, $\cos(x)$, $\sin^2(x)$, $\cos^2(x)$ ja $\sin^2(x) + \cos^2(x)$. Voit käyttää Numerical Pythonin vektorointitoimintoa. Huomioi, että Pythonin math-moduulin funktiot eivät pysty käsittelemään listoja, mutta Numerical Pythonilla saat luvun sinin yksinkertaisesti käskyllä `sin(x)` ja luvun π käskyllä `pi`. Testaa ohjelmaa useilla listan x lukujen arvoilla. Mitä huomaat?

10. Kirjoita ohjelma, joka määrittelee funktion $f(x) = 4x^4 - 4x^3$, kutsuu funktiota samassa tiedostossa ja tulostaa funktion arvon käyttäjän antamalla muuttujan x arvolla.

11. Kirjoita ohjelma, joka tutkii funktioiden $f(x) = x$ ja $g(x) = \ln(x)$ kasvunopeuksia. Tulosta funktioiden arvot kullakin muuttujan x arvolla kahden alkion pituisena listana, kun muuttuja x kuuluu välille $[1, 10]$. Kumman funktion arvot kasvavat hitaammin?

12. Kirjoita ohjelma, joka havainnollistaa funktion $h(x) = \frac{1}{x}$ vähenemistä. Laske funktiolle arvoja siihen asti, kun $h(x) < 0.05$.

13. Kirjoita ohjelma, jossa toteutetaan ensin Eukleideen algoritmi funktiona. Sitten kirjoitetaan funktio, joka tulostaa kaikki alkuluvut käyttäjän antamaan ylärajaan asti ja käyttää alkulukujen tunnistuksessa hyödykseen Eukleideen algoritmi -funktiota. Luku on alkuluku, kun sen suurin yhteinen tekijä kaikkien itseään pienempien lukujen kanssa on yksi. (Katso tehtävät 4 ja 5.)

14. Toteuta ohjelmoimalla seuraava algoritmi: Käyttäjä antaa jonkin kolminumeroisen luvun, jonka ensimmäisen ja viimeisen numeron erotus on vähintään yksi. Tämä luku käännetään toisinpäin (esimerkiksi luvusta 531 tulee 135) ja vähennetään suuremmasta luvusta pienempi. Saatu erotus käännetään taas toisinpäin. Lopullinen tulos saadaan, kun lasketaan yhteen erotus ja siitä kääntämällä saatu luku. Voit tehdä algoritmiosiosta funktion, joka palauttaa laskun lopputuloksen ja kutsua tätä funktiota myöhemmin samassa tiedostossa käyttäjän antamalla luvulla. Testaa ohjelmaa useita kertoja eri syötteillä. Mitä huomaat?

15. Kirjoita seuraavassa kuvattu ohjelma: Käyttäjä antaa syötteenä, kuinka monta omenaa Klaudialla ja Natalialla on yhteensä ja kuinka monta omenaa Klaudialla on enemmän kuin Natalialla. Ohjelma laskee ja tulostaa Klaudian ja Natalian omenien määrät. Kirjoita omenien määrän ratkaiseva osa funktiona ja kutsu sitä myöhemmin samasta tiedostosta käyttäjän antamilla arvoilla. Huomioi, että omenien määrien on oltava luonnollisia lukuja. [43]

16. Piirrä funktioiden $f(x) = e^x$ ja $g(x) = \frac{1}{e^x}$ kuvaajat samaan kuvaan Matplotlibin avulla. Voit antaa muuttujalle x esimerkiksi arvot väliltä $[-2, 2]$. Mitä huomaat?

17. Piirrä funktion $f(x) = \sqrt{x\sqrt{2+\sqrt{x}} - x\sqrt[4]{x}} \cdot \sqrt{x\sqrt{2+\sqrt{x}} + x\sqrt[4]{x}}$ kuvaaja Matplotlibin avulla. Mitä huomaat? (yo-tehtävä S1997 teht.6a, osa tehtävästä) [11]

18. Toteuta ohjelmoimalla numeronarvauspeli. Ohjelma arpoo jonkin satunnaisluvun väliltä $[0, 100]$ ja käyttäjä yrittää arvata tämän luvun. Ohjelma kertoo jokaisen arvauksen jälkeen, onko luku pienempi vai suurempi kuin käyttäjän arvaama luku. Lisäksi ohjelma laskee, kuinka monta arvausta käyttäjä joutui käyttämään luvun arvaamiseen ja ilmoittaa tämän ohjelman lopussa. [4]

19. Ohjelmoi seuraava satunnaislukupeli: Tietokone arpoo sata kokonaislukua väliltä $[0, 20]$ ja lisää ne yksitellen listaan. Käyttäjä antaa jonkin kokonaisluvun väliltä $[0, 20]$ ja ohjelma ilmoittaa, kuinka monta kertaa tämä luku sisältyy arvottujen lukujen listaan. Tietokone arpoo yhden satunnaisen kokonaisluvun väliltä $[0, 20]$ ja ilmoittaa tämän luvun sekä kertoo, kuinka monta kertaa tämä luku sisältyy arvottujen lukujen listaan. Ohjelma ilmoittaa voittajaksi joko käyttäjän tai tietokoneen sen mukaan, kumman arvaama luku sisältyi useammin arvottujen lukujen listaan. Lopuksi ohjelma tulostaa arpomansa sadan luvun listan.

20. Toteuta ohjelmoimalla lottokone, joka arpoo seitsemän numeroa ja kolme lisänumeroa. Arvotut varsinaiset numerot kerätään yhteen listaan

ja lisänumerot toiseen listaan. Ota huomioon, että jokainen numero saa tulla arvotuksi korkeintaan yhden kerran.

21. Kirjoita ohjelma, joka simuloi nopanheittoa. Käyttäjä saa antaa heittojen lukumäärän, ja ohjelma ilmoittaa heittotulokset listana sekä laskee, kuinka monta kappaletta mitäkin numeroa saatiin.

22. Piirrä tehtävän 21 nopanheiton tuloksista piirakkakuvio Matplotlibin avulla.

23. Kirjoita rekursiivinen funktio, joka palauttaa n:n jäsenen aritmeettiselle lukujonolle, jossa $a_1 = 0$ ja $d = 2$. Aritmeettisen lukujonon rekursiivinen määritelmä on $a_n = a_{n-1} + d$. Kutsu funktiota samassa tiedostossa käyttäjän antamalle indeksille n ($0 \leq n \leq 999$).

24. Hanoin tornit on matemaattinen peli, joka koostuu kolmesta tangosta ja erikokoisista kiekoista. Pelin alussa kiekot ovat yhdessä tangossa suuruusjärjestyksessä suurin kiekko alimmaisena. Pelin tavoitteena on siirtää kiekot mahdollisimman vähällä siirtojen määrällä yhdestä tangosta toiseen tankoon samaan järjestykseen käyttäen kolmatta tankoa aputankona. Vain yhtä kiekkoa saa siirtää kerrallaan ja mikään kiekko ei saa koskaan olla itseään pienemmän kiekon päällä. Pienin mahdollinen siirtojen määrä, jolla kiekot saadaan siirrettyä edellä kuvatulla tavalla, voidaan esittää rekursioyhtälönä

$$\begin{cases} T_0 = 0 \\ T_n = 2T_{n-1} + 1 \quad \text{kun } n \geq 1. \end{cases}$$

Rekursioyhtälö perustuu seuraavaan algoritmiin: Siirretään ensin $n-1$ kiekkoa ensimmäisestä tangosta aputankoon. Tähän tarvitaan T_{n-1} siirtoa. Siirretään sitten suurin kiekko ensimmäisestä tangosta toiseen tankoon. Tämä vaatii yhden siirron. Siirretään lopuksi $n-1$ kiekkoa aputangosta toiseen tankoon. Tämä vaatii T_{n-1} siirtoa. [54] Kirjoita rekursiivinen funktio, joka palauttaa pienimmän tarvittavan siirtojen määrän n kiekolle. Kutsu funktiota samassa tiedostossa käyttäjän antamalle määrälle kiekkoja.

25. Kirjoita ohjelma, joka etsii funktion $f(x) = 2^x + x$ (ainoan) nollakohdan kolmen desimaalin tarkkuudella käyttäen välinpuolitusmenetelmää. Etsi sopiva aloitusväli piirtämällä funktion kuvaaja Matplotlibin avulla. Kertaa välinpuolitusmenetelmän teoria Numeerinen matematiikka -kurssin kirjasta. [10]

26. Kirjoita ohjelma, joka etsii yhtälön $2^{-x} - x = 0$ (ainoan) ratkaisun kolmen desimaalin tarkkuudella käyttäen kiintopisteiteraatiota. Etsi sopiva alkuarvaus piirtämällä funktion kuvaaja Matplotlibin avulla. Kertaa kiintopisteiteraation teoria Numeerinen matematiikka -kurssin kirjasta. [10]

27. Muokkaa luvussa 4.7.2 kirjoitetusta Newtonin menetelmän ohjelmakoodista tarvittavat kohdat ja ratkaise sen avulla seuraava tehtävä: Määritä funktion $f(x) = x\sin(x)$ pienin positiivinen ääriarvokohta ja vastaava ääriarvo ratkaisemalla derivaatan nollakohta Newtonin menetelmällä. Anna vastaukset viiden desimaalin tarkkuudella. Hahmottele funktion kuvaaja välillä $[0, 2\pi]$. (yo-tehtävä K2005 teht. 15) [19]

28. Ohjelmoi keskipistesääntö ja laske sen avulla puolikkaan yksikköympyrän pinta-ala käyttäen ensin kymmentä jakoväliä. Ohjelman on laskettava myös pinta-alan tarkka arvo sekä likiarvon suhteellinen virhe. Tee sitten sama uudestaan käyttäen tuhatta jakoväliä ja vertaile likiarvon tarkkuutta edelliseen verrattuna. Vinkki: $f(x) = \sqrt{1 - x^2}$.

29. Muokkaa luvussa 4.7.3 kirjoitetusta puolisuunnikassäännön ohjelmakoodista tarvittavat kohdat ja ratkaise sen avulla seuraava tehtävä: Funktion f kuvaajan kaarenpituus välillä $[a, b]$ on $\int_a^b \sqrt{1 + f'(x)^2}dx$. Laske funktion $\ln(x)$ kuvaajan kaarenpituus välillä $[1, 2]$ puolisuunnikassäännöllä jakamalla väli neljään osaväliin. Anna vastaus kolmen desimaalin tarkkuudella. (yo-tehtävä K2010 teht. 13) [19]

30. Muokkaa luvussa 4.7.4 kirjoitetusta Simpsonin säännön ohjelmakoodista tarvittavat kohdat ja ratkaise sen avulla seuraava tehtävä: Laske integraali $\int_0^4 x^2 dx$ numeerisesti Simpsonin säännöllä jakamalla integroi-

misväli neljään osaväliin. (yo-tehtävä S2006 teht. 15, osa tehtävästä) [19]

31. Kirjoita ohjelma, jossa määritellään luokka Kolmio ja luodaan kolmio-olio tästä luokasta. Luokka saa attribuutteina kolmion kannan ja korkeuden. Luokassa on oltava konstruktorin lisäksi metodit kolmion kannan pituuden, korkeuden ja pinta-alan palauttamista varten. Ohjelma kysyy käyttäjältä kannan ja korkeuden arvot ja kertoo kolmion pinta-alan.

32. Kirjoita ohjelma, jossa määritellään luokka Kissaelain ja luodaan kolme oliota (gepardiolio, leijonaolio ja kotikissaolio) tästä luokasta. Luokka saa attribuuttina kissaeläimen huippunopeuden maileina tunnissa. Luokassa on oltava konstruktorin lisäksi huippunopeuden palauttava metodi sekä metodi, joka laskee ja palauttaa kissaeläimen sadan metrin juoksuun kuluvan ajan sekunteina, jos eläin juoksisi koko matkan huippunopeudellaan. Ohjelman on tulostettava kunkin kissaeläimen sadan metrin juoksuun kuluva aika. Gepardin huippunopeus on 70 mph, leijonan 50 mph ja kotikissan 30 mph [8]. 1 mph $= 1,609$ km/h ja 1 km/h $= \frac{1}{3,6}$ m/s.

33. Kirjoita ohjelma, jossa määritellään luokka Pankkitili ja luodaan pankkitiliolio tästä luokasta. Luokka saa attribuuttina tilin saldon. Luokassa on oltava konstruktorin lisäksi metodit tilin saldon palauttamista, rahan nostoa ja nostetun summan palauttamista sekä rahan tallettamista ja talletetun summan palauttamista varten. Huomioi, että jos käyttäjä yrittää nostaa enemmän rahaa kuin tilillä on, hän saa nostettua vain sen verran rahaa kuin tilillä on. Ohjelma kysyy käyttäjältä, haluaako tämä nostaa vai tallettaa rahaa vai tarkistaa tilin saldon, ja toimii käyttäjän valinnan mukaan.

34. Toteuta olio-ohjelmointia käyttäen kahden pelaajan lautapeli, jossa pelaajat heittävät vuorotellen noppaa ja siirtyvät pelilaudalla jokaisen heiton jälkeen niin monta askelta kuin nopan silmäluku osoittaa. Pelaajat saavat määritellä pelin alussa rajan, jonka ensimmäisenä ylittä-

nyt pelaaja voittaa pelin.

Ohje: Voit määritellä esimerkiksi luokat Noppa, Pelaaja ja Peli. Luo noppaolio Noppa-luokasta, kaksi pelaajaoliota Pelaaja-luokasta ja peliolio Peli-luokasta. Noppa-luokassa on oltava kontruktorin lisäksi nopan heittotuloksen palauttava metodi ja Pelaaja-luokassa metodit nopan heittämistä, heittotuloksen palauttamista, pelaajan siirtymistä, sijainnin palauttamista ja pelaajan tunnuksen palauttamista varten. Peli-luokassa on peliä pyörittävä silmukka, joka pyytää kumpaakin pelaajaa vuorollaan heittämään noppaa sekä tulostaa pelaajan heittotuloksen ja uuden sijainnin pelilaudalla. Kun toinen pelaaja ylittää pelin alussa määritellyn rajan, tulostuu ruudulle ilmoitus voitosta.

35. Lisätehtävä: Ohjelmoi kertaluvun 4 Runge-Kutta -menetelmä ja ratkaise sitä käyttäen differentiaaliyhtälö $y' + 2\sin(x) = y, \quad y(0) = 1$ välillä $[0, 2]$ askelpituudella $h = 0,5$. Testaa samaa ohjelmaa uudestaan vaihtamalla askelpituudeksi $h = 0,1$. Vertaa pisteessä $2,0$ saatuja ratkaisufunktion arvoja Eulerin menetelmällä ja keskipistemenetelmällä saatuihin vastaaviin (katso lisämateriaali).

36. Lisätehtävä: Ratkaise lineaarinen yhtälöryhmä [13]

$$\begin{cases} x - y + z = 1 \\ x + y = 2 \\ 2x - y - z = 0 \end{cases}$$

joko testaamalla luvussa 4.7.6 esitettyä Gaussin eliminoinnin ohjelmakoodia tai käyttämällä Numerical Pythonin `linalg.solve(a,b)`-käskyä.

5 Lisämateriaali: Pelien ohjelmointia ja muita sovelluksia

Tässä luvussa luodaan pienimuotoinen katsaus pelien ohjelmointiin. Pelien ohjelmointi on oma pitkälle kehittynyt alansa, josta voisi järjestää aivan oman kurssinsa tai jopa useita. Tällä kurssilla opituilla tiedoilla ja taidoilla on kuitenkin mahdollista päästä hyvään alkuun. Luvun tarkoituksena on laajentaa ymmärrystä siitä, mitä kaikkea ohjelmoinnin avulla voidaan tehdä sekä tarjota pohja niin pelien kuin muunkin ohjelmoinnin taitojen kehittämiseen tulevaisuudessa.

5.1 Tekstipohjaisia pelejä

Yksinkertaisimmillaan pelejä voidaan toteuttaa tekstipohjaisina ohjelmina samalla tavoin kuin muutkin ohjelmat aiemmin tällä kurssilla on tehty. Pelin kulkua voidaan tällöin seurata normaaliin tapaan joko IDLE:ssä tai komentorivillä. Peleistä ovat näkyvillä vain testausesimerkit. Pelien lähdekoodeja ei niiden pituuden vuoksi tutkielmassa esitellä, mutta ne on tarkoitus laittaa kurssin Internet-sivuille.

Hirsipuu
Hirsipuu on vanha arvauspeli kahdelle (tai useammalle) pelaajalle. Pelaajista toinen toimii arvuuttajana ja keksii jonkin sanan, jota toinen pelaaja yrittää kirjain kerrallaan arvata. Aluksi arvuuttaja piirtää niin monta viivaa kuin kyseisessä sanassa on kirjaimia. Toinen pelaaja arvaa esimerkiksi, sisältyykö sanaan kirjain A. Jos sisältyy, arvuuttaja kirjoittaa kirjaimen A tyhjille viivoille kaikkiin niihin kohtiin, joissa se sanassa on. Jos kirjain A ei sisälly sanaan, arvuuttaja täydentää hirsipuuta esittävään kuvaan ensimmäisen ruumiinosan eli pään. Kaikki arvaajan käyttämät kirjaimet kirjoitetaan muistiin samojen arvausten välttämiseksi. Arvaaja yrittää tietysti koko ajan miettiä, mikä sana voisi olla. Keksittyään esimerkiksi, että sana saattaisi olla hirsipuu, hän yrittää tarjota siihen sisältyviä kirjaimia. Jos arvaaja saa sanan kaikki kirjaimet näkyviin ennen hirsipuun täyttymistä, hän voittaa. Jos taas

hirsipuu tulee täyteen ennen sanan arvaamista, arvuuttaja voittaa pelin. [48]

Alla olevassa esimerkissä hirsipuu on toteutettu ohjelmoimalla siten, että käyttäjä toimii arvaajana ja arvuuttaja on korvattu tietokoneella. Pelin helpottamiseksi käyttäjä saa pelin alussa valita aihealueen, johon arvattava sana kuuluu.

Esimerkki 32. *Hirsipuu (testausesimerkki)*

```
 ------
 |    |
 |
 |
 |
 |
 |
 |
----------

Valitse hirsipuun aihealue:
         1 = Matematiikka
         2 = Tietokoneet
         3 = Urheilu
         4 = Viihde
         5 = Maantieto
         6 = Autot

Valinta on : 1
Peli alkaa!
Sanat sisaltavat vain aakkosten kirjaimia a-z.

 ------
 |    |
 |
 |
 |
 |
 |
 |
----------
```

```
Arvatut kirjaimet:
[]

Sana tahan mennessa:
------

Anna arvauksesi: a
a ei kuulu sanaan.

 ------
 |    |
 |    O
 |
 |
 |
 |
 |
----------

Arvatut kirjaimet:
['a']

Sana tahan mennessa:
------

Anna arvauksesi: i
i kuuluu sanaan.

 ------
 |    |
 |    O
 |
 |
 |
 |
 |
----------
```

```
Arvatut kirjaimet:
['a', 'i']

Sana tahan mennessa:
----i-

Anna arvauksesi: e
e ei kuulu sanaan.

 ------
 |    |
 |    O
 |    |
 |
 |
 |
 |
----------

Arvatut kirjaimet:
['a', 'i', 'e']

Sana tahan mennessa:
----i-

Anna arvauksesi: k
k kuuluu sanaan.

 ------
 |    |
 |    O
 |    |
 |
 |
 |
 |
----------

Arvatut kirjaimet:
```

```
['a', 'i', 'e', 'k']

Sana tahan mennessa:
k---i-

Anna arvauksesi: u
u ei kuulu sanaan.

 ------
 |    |
 |    O
 |  --|
 |
 |
 |
 |
----------

Arvatut kirjaimet:
['a', 'i', 'e', 'k', 'u']

Sana tahan mennessa:
k---i-

Anna arvauksesi: l
l kuuluu sanaan.

 ------
 |    |
 |    O
 |  --|
 |
 |
 |
 |
----------

Arvatut kirjaimet:
['a', 'i', 'e', 'k', 'u', 'l']
```

```
Sana tahan mennessa:
k-l-i-

Anna arvauksesi: t
t ei kuulu sanaan.

 ------
 |    |
 |    O
 |  --|--
 |
 |
 |
 |
----------

Arvatut kirjaimet:
['a', 'i', 'e', 'k', 'u', 'l', 't']

Sana tahan mennessa:
k-l-i-

Anna arvauksesi: o
o kuuluu sanaan.

 ------
 |    |
 |    O
 |  --|--
 |
 |
 |
 |
----------

Arvatut kirjaimet:
['a', 'i', 'e', 'k', 'u', 'l', 't', 'o']
```

```
Sana tahan mennessa:
kol-io

Anna arvauksesi: m
m kuuluu sanaan.

Voitit!

Sana on kolmio
```

Ristinolla

Ristinolla on kahden pelaajan peli, jossa pelaajat piirtävät vuorotellen oman merkkinsä (joko X tai O) 3×3 -ruudukkoon ja yrittävät saada kolme omaa merkkiään peräkkäin joko vaaka-, pysty- tai vinoriville. Pelistä on olemassa useita variaatioita. [53]

Esimerkissä 33 on toteutettu ohjelmoimalla perinteinen versio ristinollasta siten, että käyttäjä pelaa tietokonetta vastaan. Tietokone ei toimi pelissä sattumanvaraisesti vaan yrittää mukailla järkevästi pelaavaa ihmistä.

Esimerkki 33. *Ristinolla (testausesimerkki)*

```
Ristinolla: Anna vuorollasi sen ruudun numero, johon haluat
tehda seuraavan siirron. Siirtosi merkitaan ruudukkoon x:lla
ja tietokonevastustajasi siirrot o:lla.

             0 | 1 | 2
             ---------
             3 | 4 | 5
             ---------
             6 | 7 | 8

  |   |
---------
  |   |
---------
  |   |
```

```
Anna siirtosi valilta 0-8: 4

  |   |
---------
  | x |
---------
  |   |

Vastustajan siirto:  0

o |   |
---------
  | x |
---------
  |   |

Anna siirtosi valilta 0-8: 6

o |   |
---------
  | x |
---------
x |   |

Vastustajan siirto:  2

o |   | o
---------
  | x |
---------
x |   |


Anna siirtosi valilta 0-8: 1
```

```
o | x | o
---------
  | x |
---------
x |   |

Vastustajan siirto:  7

o | x | o
---------
  | x |
---------
x | o |

Anna siirtosi valilta 0-8: 2

Ruutu varattu, valitse toinen ruutu.

Anna siirtosi valilta 0-8: 5

o | x | o
---------
  | x | x
---------
x | o |

Vastustajan siirto:  3

o | x | o
---------
o | x | x
---------
x | o |

Anna siirtosi valilta 0-8: 8
```

```
o | x | o
---------
o | x | x
---------
x | o | x

Tasapeli!

Paina Enter lopettaaksesi ohjelman.
```

5.2 Pelien grafiikkaominaisuudet - Pygame

Rajoittuminen pelkkään tekstipohjaisuuteen ei kuitenkaan ole pelien ohjelmoinnissa kovin tarkoituksenmukaista, sillä nykyään on olemassa paljon erilaisia työkaluja muun muassa grafiikka- ja ääniominaisuuksien lisäämiseksi peleihin.

Tässä luvussa tutustutaan hieman Pythonin Pygame-nimiseen multimediakirjastoon. Työssä käytetään Pygamesta versiota pygame-1.9.2pre.win-amd64-py2.6. Kirjasto ei sisälly valmiiksi Pythoniin vaan se on ladattava Internet-osoitteesta

- www.pygame.org/download.shtml tai
- www.lfd.uci.edu/~gohlke/pythonlibs (64-bittisille käyttöjärjestelmille).

Pygame on peliohjelmointiin tarkoitettu kirjasto, joka sisältää useita valmiita funktioita grafiikka- ja ääniominaisuuksien luomista varten. Sen avulla on mahdollista luoda Python-koodilla peliä varten graafinen ikkuna, ladata peliin kuvia ja piirtää itse eri muotoja, luoda animaatioita ja törmäyksiä, määritellä kappaleiden nopeuksia, luoda ääniä ja niin edelleen. Vaikka Pygame on kohtalaisen yksinkertainen ja nopea työkalu pelien ohjelmointiin, se voi tuntua aloittelijasta monimutkaiselta. [4] Siksi tässä tyydytäänkin esittämään vain esimerkkejä peleistä, joita Pygamen avulla on mahdollista ohjelmoida. Pelien lähdekoodeja ei niiden pituuden vuoksi tutkielmassa esitellä, mutta ne on tarkoitus laittaa kurssin Internet-sivuille. Lisätietoa pelien ohjelmoinnista Pygamen avulla löytyy muun muassa kirjasta Will McGugan:

Beginning Game Development with Python and Pygame : From Novice to Professional.

Pygamen avulla on mahdollista luoda uusia pelejä tai vaihtoehtoisesti toteuttaa itse ohjelmoiden vanhoja klassikkoja. Tässä esitellään kaksi tietokonepeliä (Matopeli ja Pong) Pygamella toteutettuna. Pelit on toteutettu seuraavista Internet-osoitteista löytyvien avointen esimerkkikoodien pohjalta muokkauksia tekemällä.

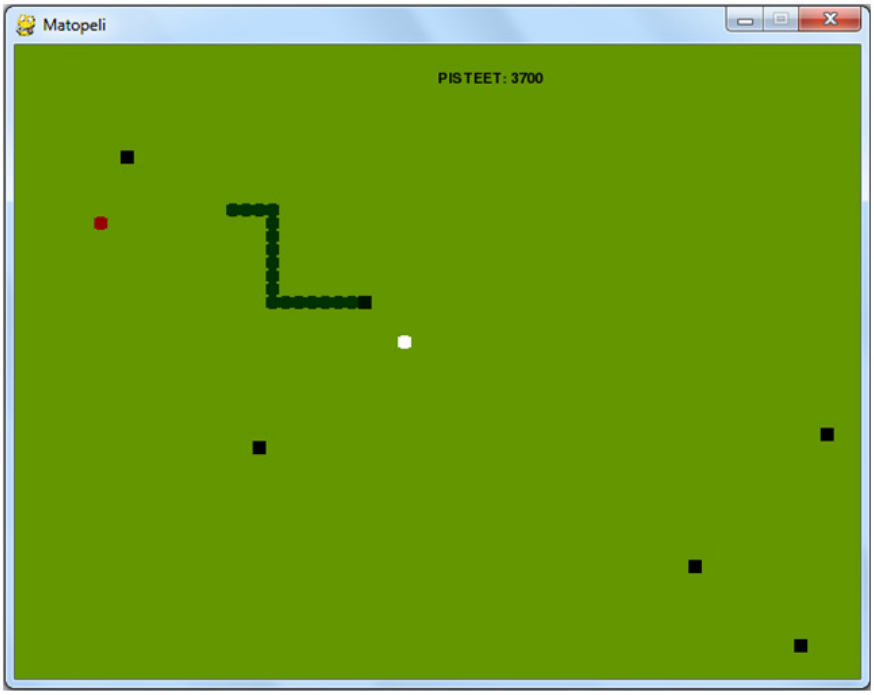
- www.pygame.org/project-Minisnake-1020-.html (Matopeli)
- www.pygame.org/project-PyPong-678-.html (Pong)

Matopeli

Matopelissä on tarkoituksena ohjata ruudulla liikkuvaa matoa tietokoneen nuolinäppäinten avulla kohti alueella kulloinkin eri paikoissa sijaitsevaa ruokapalaa. Samalla tulisi estää madon törmääminen alueen reunoihin sekä omaan häntäänsä. Pelaajan pistesaldo kasvaa aina ruokapalan pyydystämisen jälkeen. Pistesaldon kasvaessa peli vaikeutuu eli mato pitenee ja sen nopeus kasvaa. [52]

Alla olevassa Matopelissä on pelin perinteiseen versioon verrattuna kaksi lisäominaisuutta. Tavallisten punaisten ruokapalojen lisäksi ruudulle ilmestyy välillä valkoinen superruokapala, jonka keräämisestä pelaaja saa tuhat pistettä normaalin sadan pisteen sijaan. Superruokapala ilmestyy ruudulle aina, kun tietokoneen kellonaika jaettuna luvulla 19 tuottaa jakojäännöksen, joka on pienempi kuin kolme. Tämä tapahtuu tietenkin sillä ehdolla, että edellinen superruokapala on kerätty. Lisäksi ruudulle ilmestyy esteitä (mustia neliöitä) sitä mukaa, kun mato saalistaa tavallisia ruokapaloja. Jos mato törmää esteeseen, peli päättyy.

Kuva 11: Matopeli

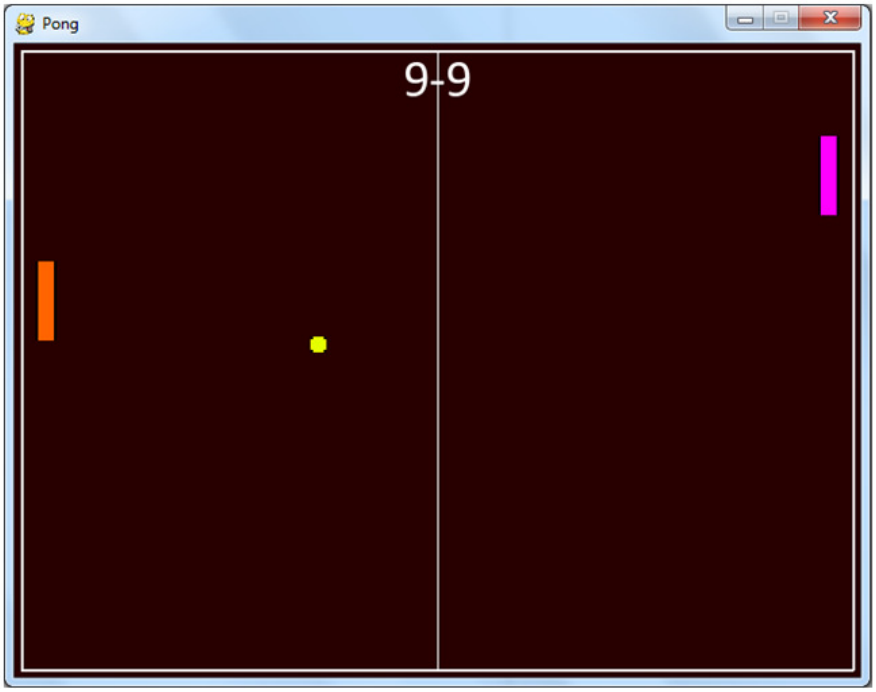


Pong

Pong on pöytätennistä muistuttava kahden pelaajan peli ja ensimmäinen kaupallisesti menestynyt videopeli. Toisella pelaajalla on maila kentän (ruudun) vasemmassa reunassa ja toisella oikeassa, ja kentällä liikkuu fysiikan lakien mukaisesti kentän ylä- ja alareunoihin sekä mailoihin törmäilevä pallo. Pelaajat liikuttelevat mailojaan ylös ja alas (tässä toinen pelaaja nuolinäppäimiä ja toinen näppäimiä w ja z käyttämällä). Tavoitteena on saada pallo vastustajan mailan taakse eli vastakkaiseen reunaan. Kun pelaaja saa pallon vastustajan mailan taakse, hän saa pisteen. Pelin tavoitteena on kerätä tietty määrä pisteitä ennen vastustajaa. Alla olevassa pelissä se pelaaja voittaa, joka saa ensimmäisenä kymmenen pistettä. [51]

Tässä peli on toteutettu kahden luonnollisen pelaajan pelinä. Pelistä olisi koodin pienillä muokkauksilla mahdollista tehdä myös yhden luonnollisen pelaajan ja tietokoneen välinen peli.

Kuva 12: Pong



5.3 Pelimoottorit

Pyrittäessä edellisiäkin esimerkkejä edistyneempään pelinkehitykseen on mahdollista käyttää apuna pelimoottoreita. Ne ovat pelinkehitysalustoja, jotka sisältävät valmiita kirjastoja ja toteutuksia pelin eri osa-alueita kuten grafiikkaa, fysiikkaa ja pelidynamiikkaa varten. Ne eivät tee peliä pelintekijän puolesta vaan niiden tarkoitus on helpottaa pelintekijöitä, jotta nämä voivat keskittyä enemmän itse pelilogiikan tekemiseen. [36]

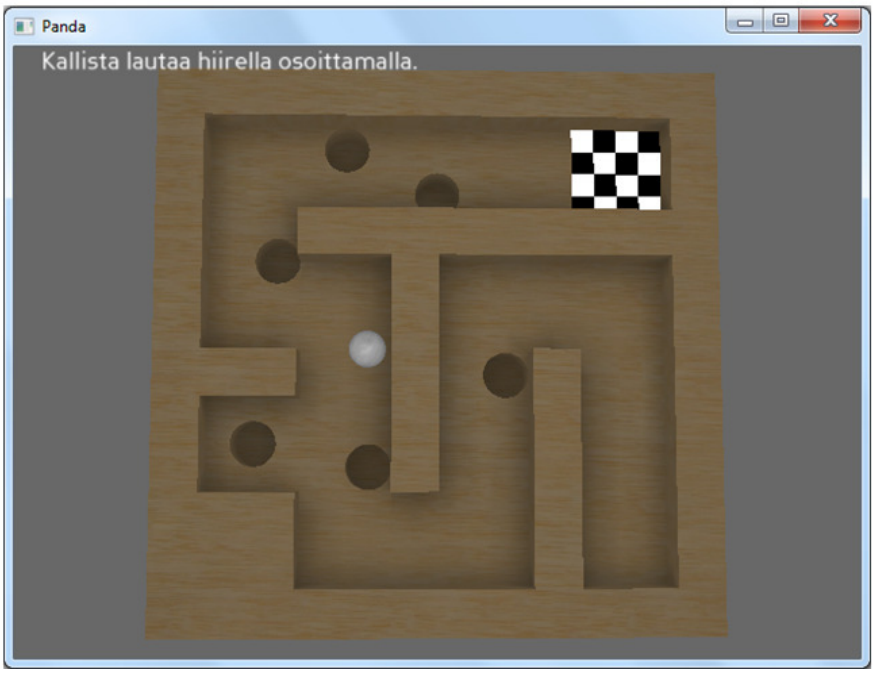
Suurin osa pelimoottoreista on tehty C++ -ohjelmointikielellä, mutta useat niistä tukevat muitakin ohjelmointikieliä. Muutamat pelimoottorit tukevat myös Python-kieltä eli toisin sanoen niissä on Python-rajapinta. [49]

Esimerkki Python-kieltä tukevasta pelimoottorista on Panda3D. Se kehitettiin aikoinaan The Walt Disney Companyn tarkoituksia varten, mutta

nykyisin se on ilmainen avoimen lähdekoodin ohjelma, joka on ladattavissa Internet-osoitteesta www.panda3d.org/download.php. Latauksen yhteydessä saa joukon esimerkkiohjelmia koodeineen. Alla olevat peliesimerkit Ball in Maze ja Asteroids on saatu näistä Panda3D:n esimerkkikoodeista pieniä muokkauksia tekemällä. Pelien lähdekoodeja ei niiden pituuden vuoksi tutkielmassa esitellä, mutta ne on tarkoitus laittaa kurssin Internet-sivuille. [26]

Pelissä Ball in Maze on tavoitteena ohjata pallo labyrintin läpi kallistelemalla labyrinttia hiiren osoittimen avulla. Samalla on varottava, ettei pallo putoa reitillä oleviin koloihin. Jos pallo kuitenkin putoaa koloon, se siirtyy automaattisesti takaisin lähtöön ja peli alkaa alusta.

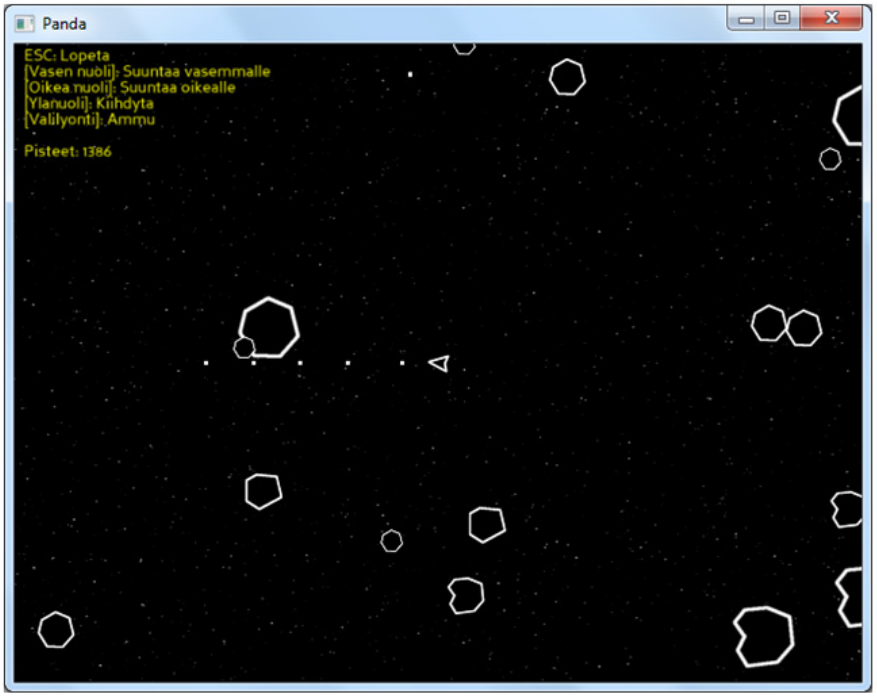
Kuva 13: Ball in Maze



Pelissä Asteroids on tavoitteena kerätä mahdollisimman paljon pisteitä ampumalla asteroideja avaruusaluksesta. Asteroidit puolittuvat jokaisesta osumasta, ja jokainen osuma tuottaa pelaajalle 99 pistettä. Jos asteroidi osuu avaruusalukseen, pelaajan pisteet nollautuvat ja peli alkaa alusta. Avaruusa-

lusta liikutetaan nuolinäppäinten avulla: vasen nuoli suuntaa alusta vasemmalle, oikea nuoli oikealle ja ylänuoli lisää aluksen nopeutta. Ampuminen suoritetaan välilyöntinäppäimellä.

Kuva 14: Asteroids



5.4 Ohjelmoinnin muita soveltamiskohteita

Kurssin alussa mainittiin, että ohjelmoinnin avulla voi tehdä lähes kaikkea. Tämän kurssin puitteissa on mahdollista tarjota vain hyvin pieni johdatus ohjelmoinnin laajaan maailmaan, mutta esitellään tässä luvussa lyhyesti vielä kaksi ohjelmoinnin sovelluskohdetta, joiden toteutus tosin vaatii vahvaa matematiikan ja ohjelmoinnin ymmärrystä.

Python Sudoku

Python Sudoku on ohjelma, joka pystyy sekä luomaan, ratkaisemaan

että tulostamaan vaikeustasoltaan erilaisia sudokuristikkoja. Ohjelmaan ei tässä yhteydessä perehdytä tarkasti, mutta se on mahdollista ladata omalle tietokoneelle testauskäyttöön tai muokattavaksi Internet-osoitteesta http://sourceforge.net/projects/pythonsudoku/files. [34]

PageRank-algoritmi

Hakukone Googlen PageRank-algoritmi asettaa tärkeysjärjestykseen kaikki ne Internet-sivut, joilla esiintyy se hakusana tai sanayhdistelmä, jota käyttäjä haluaa etsiä. Hakutulokset eivät siis tulostu näytölle satunnaisessa järjestyksessä haun jälkeen. PageRank-algoritmi määrittää kunkin Internet-sivun tärkeysasteen sillä perusteella, kuinka monelta muulta sivulta löytyy linkki kyseiselle sivulle ja kuinka tärkeitä nämä sivut vastaavasti ovat. Eli sivu, jolle on olemassa useita linkkejä muilta samalla perusteella tärkeiltä sivuilta, saavuttaa korkeamman tärkeysasteen kuin sivu, jolle on olemassa vain vähän linkkejä sellaisilta sivuilta, joihin ei kenties ole osoitettu linkkejä mistään. [1]

Sivun PageRank-arvo kuvaa kyseisen sivun tärkeyttä suhteessa muihin sivuihin. Jokaisen sivun tärkeyden määrittäminen vaatii tiedon tähän sivuun linkitettyjen sivujen tärkeydestä. Näiden sivujen tärkeyden määrittäminen taas vaatii tiedon niihin linkitettyjen sivujen tärkeydestä ja niin edelleen. Pohjattoman rekursion välttämiseksi sivujen tärkeyden määrittämiseen käytetään apuna sivujen linkityksien perusteella muodostettua hyperlinkkimatriisia ja sen ominaisvektoria. [1]

Jotta Google pystyy käsittelemään satoja miljoonia Internet-sivuja nopeasti, sillä täytyy olla erittäin tehokas sivujen läpikäyntisysteemi. Tämä systeemi toimii niin, että yksittäinen URL-palvelin jakaa listan eri sivujen URL-tunnisteita eli käytännössä sivujen Internet-osoitteita useammalle sivujen läpikävijälle. Sekä URL-palvelin että edellä mainitut sivujen läpikävijät on toteutettu Pythonilla. [2]

Lähteet

[1] D. Austin: *How Google Finds Your Needle in the Web's Haystack*, Feature Column of American Mathematical Society, Grand Valley State University. Viitattu 13.11.2010
http://www.ams.org/samplings/feature-column/fcarc-pagerank

[2] S. Brin, L. Page: *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, Stanford University. Viitattu 13.11.2010
http://infolab.stanford.edu/~backrub/google.html

[3] Curiousmath, Fun math tricks. Viitattu 2.12.2010
http://www.curiousmath.com/index.php?name=News&file=article&sid=31

[4] M. Dawson: *Python Programming for the Absolute Beginner*, Second Edition, Canada, 2008.

[5] Formatoitu tulostus. Viitattu 11.12.2010
http://www.mit.jyu.fi/opetus/Ciao/ciao040.htm

[6] Fast fractals with Python and Numpy. Viitattu 28.8.2010
http://thesamovar.wordpress.com/2009/03/22/fast-fractals-with-python-and-numpy

[7] J. Hunter, D. Dale: *The Matplotlib User's Guide*. Viitattu 3.8.2010
http://www.phy.olemiss.edu/~jgladden/comp_phys/resources/pylab_users_guide_0.91.2svn.pdf

[8] Infoplease, Speed of Animals. Viitattu 3.12.2010
http://www.infoplease.com/ipa/A0004737.html

[9] J. Järvinen: *Johdatus informaatioteknologiaan*, Turun yliopisto, 2007.

[10] J. Kangasaho, J. Mäkinen, J. Oikkonen, J. Paasonen, M. Salmela: *Numeerinen matematiikka*, 1.-2. painos, WSOY, Porvoo, 1998.

[11] J. Kangasaho, P. Piri, H. Taavitsainen: *Pitkän matematiikan ylioppilaskokeet 1994-2004*, 7. painos, WSOY, Juva, 2004.

[12] P. Kontkanen, J. Lehtonen, K. Luosto, J. Nurmi, R. Nurmiainen: *Pyramidi Matematiikan tietokirja 4*, 1. painos, Tammi, Helsinki, 2000.

[13] P. Kontkanen, R. Liira, K. Luosto, J. Nurmi, R. Nurmiainen, A. Ronkainen, S. Savolainen: *Pyramidi Matematiikan tietokirja 2*, 1.-4. painos, Tammi, Helsinki, 1998.

[14] P. Kosonen, J. Peltomäki, S. Silander: *Java 2, Ohjelmoinnin peruskirja*, 4. painos, Vantaa, 2007.

[15] J. Kytömäki, etälukio, kurssi MAA12. Viitattu 20.11.2010 http://www02.oph.fi/etalukio/pitka_matematiikka/kurssi12/muutos/deri_kesk.html

[16] A.-J. Lakanen, T. Jäntti, V. Lappalainen, J. Tammela: *Nuorten peliohjelmointi*, Jyväskylän yliopisto, 2010. Viitattu 28.8.2010 http://kurssit.it.jyu.fi/npo/2010/oppimateriaali/oppimateriaali_v1.2.pdf

[17] F. Lundh: *Importing Python Modules*, 2001. Viitattu 10.12.2010 http://effbot.org/zone/import-confusion.htm

[18] L. Malmi: *Ohjelmointikielet ja -paradigmat*, luento 27.2.2007, Teknillinen korkeakoulu. Viitattu 29.8.2010 http://www.cs.hut.fi/Opinnot/T-106.3100/K2007/Luennot/Luentokielet-27022007.pdf

[19] Matematiikan ylioppilastehtävät. Viitattu 8.8.2010-20.8.2010 http://intmath.org/other/yoteht

[20] Math-moduulin funktiot. Viitattu 18.11.2010 http://docs.python.org/library/math.html

[21] Matplotlib User's Guide - Matplotlib pyplot. Viitattu 3.8.2010, 6.8.2010 http://matplotlib.sourceforge.net/api/pyplot_api.html

[22] Matplotlib, x- ja y-koordinaattien fonttikoko. Viitattu 10.12.2010
http://www.mail-archive.com/matplotlib-
users@lists.sourceforge.net/msg05330.html

[23] U. Nikula: *Ohjelmoinnin perusteet*, luento 9.9.2008, Lappeenrannan teknillinen yliopisto. Viitattu 29.8.2010
http://www2.it.lut.fi/kurssit/08-09/CT20A0200/Luento02.pdf

[24] Numeerinen laskenta ja virheet. Viitattu 16.12.2009
http://www.helsinki.fi/~fyl_tlpk/luento/Tlp-num1.pdf

[25] Numerical Pythonin funktioita. Viitattu 18.11.2010
http://docs.scipy.org/doc/numpy/reference/routines.linalg.html

[26] Panda3D. Viitattu 6.11.2010
http://www.panda3d.org/manual/index.php/Introduction_to_
Panda3D

[27] K. Parvinen: *Numeerinen analyysi*, Turun yliopisto, 2009.

[28] Python In Schools. Viitattu 28.8.2010
http://www.seapig.org/PythonInSchools

[29] Pythonin moduulit. Viitattu 27.11.2010, 10.12.2010
http://docs.python.org/tutorial/modules.html

[30] Pythonin sisäänrakennetut funktiot. Viitattu 18.11.2010, 10.12.2010
http://docs.python.org/library/functions.html

[31] Pythonin version 2.6.5 syntaksiopas. Viitattu 18.11.2010
http://docs.python.org/release/2.6.5/reference/expressions.html

[32] Python-opas. Viitattu 11.9.2010
http://wiki.mureakuha.com/wiki/Python

[33] Python Scripts and Modules, Applied Mathematics University of Washington. Viitattu 27.11.2010
http://kingkong.amath.washington.edu/uwamath583/
sphinx/notes/html/python_scripts_modules.html

[34] Python Sudoku. Viitattu 13.11.2010
http://pythonsudoku.sourceforge.net

[35] S. Pyöttiälä, V. Leppänen, J. Boberg: *Algoritmien ja ohjelmoinnin peruskurssi*, 2. versio, Turun yliopisto, 2007.

[36] Radiant Raccoon: *Pelimoottorit*, luento, Jyväskylän yliopisto.
Viitattu 6.11.2010
http://users.jyu.fi/∼vesal/kurssit/winohj06/materiaali/Pelimoottorit.ppt

[37] Random-moduulin funktiot. Viitattu 18.11.2010, 29.11.2010
http://docs.python.org/library/random.html

[38] A. Rasila: *Ohjelmoinnin alkeita Python-kielellä*, Matematiikkalehti Solmu, 1/2004. Viitattu 20.11.2010
http://solmu.math.helsinki.fi/2004/1/python.pdf

[39] A. Rasila: *Numeerista lineaarialgebraa Python-kielellä*, Matematiikkalehti Solmu, 2/2005. Viitattu 23.10.2009, 21.8.2010
http://solmu.math.helsinki.fi/2005/2/python3.pdf

[40] Scientific Pythonin funktioita. Viitattu 18.11.2010
http://www.scipy.org/doc/api_docs/SciPy.misc.common.html

[41] J. Smed, H. Hakonen: *Algorithms and Networking for Computer Games*, Wiley, Eastbourne

[42] J. Smed, H. Hakonen, T. Raita: *Sopimuspohjainen olio-ohjelmointi Java-kielellä*, 2007. Viitattu 29.11.2010
http://www.cs.utu.fi/staff/jouni.smed/SHR07-SPOO.pdf

[43] Sphere Online Judge, Programming problems. Viitattu 1.12.2010
http://www.spoj.pl/problems/JULKA/

[44] Sphere Online Judge, Programming problems. Viitattu 1.12.2010
http://www.spoj.pl/problems/DIVSUM/

[45] J. Wallasvaara: *XNA...at a glance*, luento 30.10.2009, Turun yliopisto. Viitattu 28.8.2010
https://moodle.utu.fi/file.php/2146/XNA_Game_Studio_overview_30.10.2009_Turku.pdf

[46] Wikipedia, Ada Lovelace. Viitattu 29.8.2010
http://en.wikipedia.org/wiki/Ada_Lovelace

[47] Wikipedia, Deterministic algorithm. Viitattu 29.11.2010
http://en.wikipedia.org/wiki/Deterministic_algorithm

[48] Wikipedia, Hangman game. Viitattu 23.10.2010
http://en.wikipedia.org/wiki/Hangman_(game)

[49] Wikipedia, List of game engines. Viitattu 6.11.2010
http://en.wikipedia.org/wiki/List_of _game_engines

[50] Wikipedia, Matriisi. Viitattu 21.8.2010
http://fi.wikipedia.org/wiki/Matriisi

[51] Wikipedia, Pong. Viitattu 30.10.2010
http://en.wikipedia.org/wiki/Pong

[52] Wikipedia, Snake video game. Viitattu 30.10.2010
http://en.wikipedia.org/wiki/Snake_(video_game)

[53] Wikipedia, Tic-tac-toe. Viitattu 23.10.2010
http://en.wikipedia.org/wiki/Tic-tac-toe

[54] Wikipedia, Tower of Hanoi. Viitattu 1.12.2010
http://en.wikipedia.org/wiki/Tower_of_Hanoi

Internet-osoitteet, joita ei varsinaisesti ole käytetty työn lähteinä mutta joista on ladattu työssä tarvittavia ohjelmia: (suluissa mainitaan päivämäärä, jolloin viimeisin työssä käytetty versio kyseisestä ohjelmasta on ladattu)

- Python: www.python.org/download (14.7.2010)
- Numerical Python: http://sourceforge.net/projects/numpy/files
- Scientific Python: http://sourceforge.net/projects/scipy/files
- Matplotlib: http://matplotlib.sourceforge.net
- Numerical Python, Scientific Python, Matplotlib ja Pygame 64-bittisille käyttöjärjestelmille: www.lfd.uci.edu/∼gohlke/pythonlibs (14.7.2010, 17.7.2010, 22.10.2010)
- Pygame: www.pygame.org/download.shtml
- Matopelin alkuperäinen koodi: www.pygame.org/project-Minisnake-1020-.html (27.10.2010)
- Pongin alkuperäinen koodi: www.pygame.org/project-PyPong-678-.html (29.10.2010)
- Panda3D: www.panda3d.org/download.php (21.10.2010)

A Liite 1: Harjoitustehtävien ratkaisut

Luvun 3 harjoitustehtävät:

1. ```
   print "a)", 2*25+50
   print "b)", 10.0/3
   print "c)", 3**6
   print "d)", (38909**2+(766*6))/5
   print "e)", abs(5*40-4*60)
   print "f)", (1.0/2+1.0/4+1.0/6)/(1.0/2+5.0/12)
   ```

2. ```
   print ((324*80+1)*250+9187+9187-250)/2
   ```

   Lopputuloksen pitäisi olla sama kuin puhelinnumerosi.

3. ```
   puh_numero = raw_input("Anna puhelinnumerosi: ")
   print "Puhelinnumerosi on ",puh_numero,"."
   ```

4. ```
   a = input("Anna 1. luku: ")
   b = input("Anna 2. luku: ")
   c = input("Anna 3. luku: ")
   d = input("Anna 4. luku: ")
   e = input("Anna 5. luku: ")
   print "Lukujen keskiarvo on ",(a+b+c+d+e)/5,"."
   ```

5. ```
   s = input("Anna matka (km): ")
   t = input("Anna aika (h): ")
   print "Keskinopeus: ",float(s)/t," km/h."
   ```

6. ```
   lista = []
   lista.append(1)
   lista.append(2)
   lista.append(3)
   lista.append(4)
   ```

```
lista.append(5)
print "lista: ",lista
print "listan pituus: ",len(lista)
print "kolmas alkio: ",lista[2]
```

7.
```
a = input("Anna ensimmainen luku: ")
suurin = a
b = input("Anna toinen luku: ")
if b>suurin:
    suurin = b
c = input("Anna kolmas luku: ")
if c>suurin:
    suurin = c
print "Suurin luvuista on ",suurin,"."
```

8.
```
a = input("Anna kokonaisluku: ")
b = input("Anna toinen kokonaisluku: ")
while a%1 != 0 or b%1 != 0 or b == 0:
    a = input("Anna kokonaisluku: ")
    b = input("Anna toinen kokonaisluku: ")
if a%b == 0:
    print "Luku" ,a, "on jaollinen luvulla" ,b,"."
else:
    print "Luku" ,a, "ei ole jaollinen luvulla" ,b,"."
    print "Jakojaannos on" ,a%b,"."
```

9.
```
a = input("Anna luku: ")
#While-lause.
i=0 #i on kertoja
while i<=10:
    print i,"*",a," = ",i*a
    i+=1
print "--------------------"
```

```
#For-lause.
for i in range(11):
    print i,"*",a," = ",i*a
```

10.
```
#kertotaulu
for j in range(1,11):
    for i in range(10):
        print (i+1)," * ", (j), " = " , (i+1)*j
    print"\n=====================\n"
```

11.
```
import math
print "a)", math.log(83)
print "b)", math.log(5,6)
print "c)", math.log10(1021021)
print "d)", math.exp(math.e)
print "e)", math.factorial(18)
print "f)", math.tan(math.pi/4)
```

12.
```
import math
print math.degrees(math.pi/6)
print math.radians(135)
```

13.
```
import math
r = input("Anna ympyran sade (cm): ")
while r < 0:
    r = input("Anna ympyran sade (cm): ")
print "Ympyran ala (cm^2): ",math.pi*r**2
```

14.
```
import math
a = input("Anna ensimmainen kateetti: ")
b = input("Anna toinen kateetti: ")
while a <= 0 or b <= 0:
```

```
    a = input("Anna ensimmainen kateetti: ")
    b = input("Anna toinen kateetti: ")
hypotenuusa = math.sqrt(a**2 + b**2)
print "Hypotenuusa =",hypotenuusa
```

Luvun 4 harjoitustehtävät:

1. ```
   import math
   n = 1
   while math.sqrt(n)/n  > 0.35:
       print math.sqrt(n)/n
       n += 1
   ```

2. ```
   n = input("Kuinka monta termia haluat? ")
   while n%1 != 0 or n < 0:
       n = input("Kuinka monta termia haluat? ")
   for i in range(1,n+1):
       print 4*3**(i-1)
   ```

3. ```
   ensimmainen = 2
   viimeinen = 98
   d = 2
   termien_lkm = (viimeinen - ensimmainen + d)/d
   summa = termien_lkm * (ensimmainen + viimeinen)/2.0
   print "Lukuja on ",termien_lkm," kpl."
   print "Lukujen summa on ",summa
   ```

4. ```
   a = input("Anna suurempi kok.luku: ")
   b = input("Anna pienempi kok.luku: ")
   while a%1 != 0 or b%1 != 0 or a == 0 or b == 0:
       a = input("Anna suurempi kok.luku: ")
       b = input("Anna pienempi kok.luku: ")
   while b != 0:
   ```

```
    apu1 = a
    a = b
    apu2 = b
    b = apu1 % b
    if b == 0:
        print "syt: ",abs(apu2)
```

Testaustulos: $\text{syt}(34086, 14630) = 38$

5.
```
raja = int(input("Anna ylaraja: "))
if raja <= 1:
    print "Lukua ", raja, "pienempia alkulukuja ei ole."

laskuri = 0
for jaettava in range(3, raja+2):
    if laskuri == jaettava-3:
        print jaettava - 1
    laskuri = 0
    for jakaja in range(2, jaettava):
        if jaettava % jakaja != 0:
            laskuri+=1
```

6.
```
print "Laske haluamasi luvun aitojen jakajien summa."
n = input("Anna luonnollinen luku: ")
while n%1 != 0 or n<0:
    n = input("Anna luonnollinen luku: ")
aj = []
for i in range(1,n):
    if n%i == 0:
        aj.append(i)
print "Aidot jakajat: ",aj
summa = 0
for i in range(0,len(aj)):
    summa += aj[i]
print "Aitojen jakajien summa: ",summa
```

7.
```
from numpy import *
#a = i + 4j - k
a = array([1,4,-1])
print "a = ",a
#b = 11i -6j + k
b = array([11,-6,1])
print "b = ",b
#summa
print "a + b = ",a+b
#pistetulo
print "pistetulo = ",dot(a,b)
#pituudet
a_pit = math.sqrt(a[0]**2+a[1]**2+a[2]**2)
b_pit = math.sqrt(b[0]**2+b[1]**2+b[2]**2)
print "a:n pituus = ",a_pit
print "b:n pituus = ",b_pit
```

8.
```
from numpy import *
#a = 2i + 5j
a = array([2,5])
#b = i -2j
b = array([1,-2])
#summa
summa = a+b
print "a + b = ",summa[0],"i+",summa[1],"j"
#yksikkovektori
c = summa[0]/math.sqrt(summa[0]**2+summa[1]**2)
d = summa[1]/math.sqrt(summa[0]**2+summa[1]**2)
print "yksikkovektori: ",c,"i+",d,"j"
```

9.
```
from numpy import *
x = array([0, (1.0/4)*pi, (1.0/3)*pi, (1.0/2)*pi])
```

```
print "sin(x): ",sin(x)
print "cos(x): ",cos(x)
print "(sin(x))^2: ",(sin(x))**2
print "(cos(x))^2: ",(cos(x))**2
print "(sin(x))^2 + (cos(x))^2: ",(sin(x))**2 + (cos(x))**2
```

Huomataan, että kaikilla testatuilla luvuilla $\sin^2(x) + \cos^2(x) = 1$.

10.
```
def f(x):
    return 4*x**4-4*x**3

#main
x = input("Anna muuttujan x arvo: ")
print f(x)
```

11.
```
import math

def f(x):
    return x

#AE: x > 0
def g(x):
    return math.log(x)

def main():
    for i in range(1,11):
        print [f(i),g(i)]

main()
```

Funktion $g(x) = \ln(x)$ arvot kasvavat hitaammin.

12.
```
#AE: x != 0
def h(x):
    return 1.0/x
```

```
def main():
    i = 1
    while h(i) >= 0.05:
        print h(i)
        i+=1

main()
```

13.
```
#Palauttaa lukujen a ja b suurimman yhteisen tekijan.
#AE: a > b, a ja b pos.kokonaislukuja
def eukleides(a,b):
    while b != 0:
        apu = a
        a = b
        b = apu % b
    return a

#Tulostaa enintaan luvun raja suuruiset alkuluvut.
#AE: rajan oltava pos.kokonaisluku
def alkuluvut(raja):
    for i in range(raja,1,-1):
        laskuri = 0
        for j in range(i-1,1,-1):
            syt = eukleides(i,j)
            if syt != 1:
                laskuri += 1
        if laskuri == 0:
            print i

#Tulostaa enintaan kayttajan antaman luvun suuruiset alkuluvut.
#AE: true
def main():
    ylaraja = int(input("Anna ylaraja: "))
```

```
    alkuluvut(ylaraja)

main()
```

14.
```
#Palauttaa algoritmin laskeman luvun.
#AE: abs(a-c) > 0 & a,b,c luonnollisia lukuja
def taikatemppu(a,b,c):
    luku1 = a*100 + b*10 + c
    luku2 = c*100 + b*10 + a
    if luku1 < luku2:
        uusi1 = luku2 - luku1
    else:
        uusi1 = luku1 - luku2
    s = uusi1/100
    k = 1.0/10 * (uusi1%100.0 - (uusi1%100.0)%10)
    y = (uusi1%100.0)%10
    uusi2 = y*100 + k*10 + s
    return int(uusi1 + uusi2)

a,b,c = input("Anna luku muodossa a,b,c: ")
print taikatemppu(a,b,c)
```

Testaustuloksia:

```
>>>
Anna luku muodossa a,b,c: 9,8,6
1089
>>>
Anna luku muodossa a,b,c: 6,3,2
1089
>>>
Anna luku muodossa a,b,c: 2,1,0
1089
>>>
Anna luku muodossa a,b,c: 1,2,3
```

```
1089
>>>
Anna luku muodossa a,b,c: 3,6,4
1089
>>>
Anna luku muodossa a,b,c: 6,4,7
1089
```

15.
```
#Tulostaa Klaudian ja Natalian omenien maarat, jos
#ne ovat luonn.lukuja. Muuten virheilmoitus.
#AE: n,k luonn.lukuja
def ratkaisu(n,k):
    #Klaudian omenat = x
    #x = (n - x) + k
    x = (n + k)/2.0
    if x%1 != 0:
        print "Jako ei mene tasan."
    else:
        print "Klaudian omenat: ",int(x)
        print "Natalian omenat: ",int(n-x)
    return " "

#main
n = input("Kuinka monta omenaa on yhteensa? ")
k = input("Kuinka monta omenaa Klaudialla on enemman? ")
while n < 0 or k < 0 or n%1 != 0 or k%1 != 0:
    print "Omenien maara oltava luonn.luku."
    n = input("Kuinka monta omenaa on yhteensa? ")
    k = input("Kuinka monta omenaa Klaudialla on enemman? ")
ratkaisu(n,k)
```

16.
```
from pylab import *
def f(x):
    return exp(x)
```

```
def g(x):
    return 1/(exp(x))
x = arange(-2.0, 2.0, 0.1)
ex1, ex2 = plot(x, f(x), x, g(x))
setp(ex1, linestyle='-', linewidth=3.0, color='c')
setp(ex2, linestyle='--', linewidth=2.0, color='k')
grid()
show()
```

Huomataan, että funktioiden kuvaajat ovat symmetriset y-akselin suhteen.

17.
```
from pylab import *
def f(x):
    a = sqrt(x*sqrt(2+sqrt(x))-x*sqrt(sqrt(x)))
    b = sqrt(x*sqrt(2+sqrt(x))+x*sqrt(sqrt(x)))
    return a*b
x = arange(0.0, 5.0, 0.1)
plot(x, f(x))
grid()
show()
```

Huomataan, että kuvaaja on lineaarinen.

18.
```
import random
numero = random.randrange(101)
arvaus = input("Arvaukseni on: ")
while arvaus<0 or arvaus>100 or arvaus%1!=0:
    arvaus = input("Arvaukseni on: ")
yritykset = 1
while (arvaus != numero):
    if (arvaus > numero):
        print "Pienempi"
    else:
        print "Suurempi"
    arvaus = input("Uusi arvaukseni on: ")
```

```
    while arvaus<0 or arvaus>100 or arvaus%1!=0:
        arvaus = input("Uusi arvaukseni on: ")
    yritykset += 1
print "Oikein, se on ",numero, "! :)"
print "Kaytit ",yritykset, "arvausta."
```

19.
```
import random
luvut = []
for i in range(101):
    luvut.append(random.randrange(21))
a = input("Anna kok.luku valilta 0-20: ")
while a < 0 or a > 20 or a%1 != 0:
    a = input("Anna kok.luku valilta 0-20: ")
osumat = 0
for i in range(len(luvut)):
    if a == luvut[i]:
        osumat += 1
print "Lukusi esiintyi ",osumat," kertaa."
b = random.randrange(21)
print "Vastustajan luku: ",b
osumat2 = 0
for i in range(len(luvut)):
    if b == luvut[i]:
        osumat2 += 1
print "Vastustajan luku esiintyi ",osumat2," kertaa."
if osumat > osumat2:
    print "Voitit!"
elif osumat == osumat2:
    print "Tasapeli!"
else:
    print "Havisit tietokoneelle!"
print "Luvut olivat: ",luvut
```

20.
```
import random
#Palauttaa listat numeroista ja lisanumeroista.
#AE: true
def lottokone():
    numerot = []
    while len(numerot) < 7:
        uusi = random.randrange(1,40)
        if uusi not in numerot:
            numerot.append(uusi)
    lisanumerot = []
    while len(lisanumerot) < 3:
        uusi = random.randrange(1,40)
        if uusi not in lisanumerot and uusi not in numerot:
            lisanumerot.append(uusi)
    return numerot,lisanumerot
#main
print lottokone()
```

Testaus:

```
>>>
([4, 37, 29, 28, 8, 3, 12], [32, 22, 25])
```

21.
```
import random

#Palauttaa heittotulokset sisaltavan listan.
#AE: heittojen_maara > 0 ja heittojen_maara on kokonaisluku
def nopanheitto(heittojen_maara):
    heitot = []                             #lista
    for i in range(heittojen_maara):
        heittotulos = random.randrange(1,7) #sat.luku 1-6
        heitot.append(heittotulos)          #alkio listaan
    return heitot
```

```
#Palauttaa listan, jonka alkiot kunkin heittotuloksen lkm.
#AE: heitto_lista != none
def laskuri(heitto_lista):
    yk_lkm = 0
    ka_lkm = 0
    ko_lkm = 0
    ne_lkm = 0
    vi_lkm = 0
    ku_lkm = 0
    for i in range(len(heitto_lista)):
        if heitto_lista[i] == 1:
            yk_lkm += 1
        elif heitto_lista[i] == 2:
            ka_lkm += 1
        elif heitto_lista[i] == 3:
            ko_lkm += 1
        elif heitto_lista[i] == 4:
            ne_lkm += 1
        elif heitto_lista[i] == 5:
            vi_lkm += 1
        else:
            ku_lkm += 1
    return [yk_lkm, ka_lkm, ko_lkm, ne_lkm, vi_lkm, ku_lkm]

#Tulostaa heittotulokset sisaltavan listan ja kunkin
#heittotuloksen maaran.
#AE: true
def main():
    heittojen_lkm = input("Anna heittojen lukumaara: ")
    heitot = nopanheitto(heittojen_lkm)
    print "Heittotulokset:", heitot
    print "Tulosten lukumaarat:", laskuri(heitot)
main()
```

Testaus:

```
>>>
Anna heittojen lukumaara: 100
Heittotulokset: [2, 4, 4, 3, 4, 3, 1, 6, 2, 2, 3, 4, 5, 5, 2,
6, 5, 6, 3, 3, 6, 5, 1, 2, 1, 1, 6, 1, 3, 4, 3, 5, 3, 5, 3, 1,
1, 4, 4, 6, 2, 4, 5, 6, 2, 4, 6, 5, 6, 5, 1, 2, 4, 4, 1, 3, 4,
2, 6, 2, 5, 4, 6, 2, 3, 6, 1, 1, 3, 3, 5, 5, 5, 1, 5, 2, 4, 1,
1, 3, 5, 4, 1, 1, 1, 5, 4, 2, 1, 4, 6, 5, 1, 5, 6, 2, 6, 2, 4,
5]
Tulosten lukumaarat: [19, 15, 14, 18, 19, 15]
```

22.
```
from pylab import *
x = [19, 15, 14, 18, 19, 15]
pie(x, colors=('y', 'w', 'b', 'c', 'm', 'r'), explode=None,
    labels=('1', '2', '3', '4', '5', '6'),
    autopct='%1.f%%', shadow=True)
title('Tulosjakauma')
show()
```

23.
```
#Palauttaa aritm.jonon n:n termin.
#AE: n kok.luku ja 0 >= n >= 1000
def aritmeettinen(n):
    d = 2
    if n == 1:
        return 0
    else:
        return aritmeettinen(n-1) + d
#main
n = input("Valitse termi (indeksi valilla 1-999): ")
while n <= 0 or n >= 1000 or n%1 != 0:
    n = input("Valitse termi (indeksi valilla 1-999): ")
print aritmeettinen(n)
```

24.
```
#Palauttaa pienimman tarvittavan siirtojen lkm.
#AE: n kuuluu luonn. lukuihin.
def Hanoi(n):
    if n == 0:
        return n
    else:
        return 2*Hanoi(n-1)+1
#main
n = input("Anna kiekkojen maara: ")
while n < 0 or n%1 != 0:
    n = input("Anna kiekkojen maara: ")
print Hanoi(n)
```

25. Kuvaajan lähdekoodi:

```
from pylab import *
def f(x):
    return 2**x + x
x = arange(-3.0, 4.0, 0.1)
plot(x, f(x))
grid()
show()
```

Välinpuolitusmenetelmä:

```
#Palauttaa funktion f(x)=2**x + x arvon.
#AE: x kuuluu reaalilukuihin
def f(x):
    return 2**x + x

#Tulostaa likiarvoja yhtalon 2**x + x = 0 ratkaisuiksi.
#Saa parametreina valin alku- ja loppupisteen.
#AE: a ja b kuuluvat reaalilukuihin
def puolitus(a, b):
```

```
    if f(a)*f(b)<0:               #f(a) ja f(b) erimerkkiset
        kp=(a+b)/2.0              #c on valin [a,b] kp
        h=0.0005                  #haluttu tarkkuus (3 des.)
        print " a    kp          f(kp)     f(kp-h)*f(kp+h)"
        taul = (0,kp,f(kp),f(kp-h)*f(kp+h))
        print "%2d%12.6f%12.6f%12.4e"%taul
        d=1                       #askel
        while f(kp-h)*f(kp+h)>0:
            kp=(a+b)/2.0
            if f(a)*f(kp)<0:      #f(a) ja f(c) erimerkkiset
                b=kp              #ratkaisu on valilla [a,c]
            else:                 #f(a) ja f(c) samanmerkkiset
                a=kp              #ratkaisu on valilla [c,b]
                taul = (d,kp,f(kp),f(kp-h)*f(kp+h))
            print "%2d%12.6f%12.6f%12.4e"%taul
            d+=1
    else:
        print "Virhe! Arvot f(a) ja f(b) ovat samanmerkkiset."
    return " "

#main
puolitus(-2.0, 1.0)

>>>
as    kp          f(kp)     f(kp-h)*f(kp+h)
 0   -0.500000    0.207107  4.2893e-02
 1   -0.500000    0.207107  4.2893e-02
 2   -1.250000   -0.829552  6.8816e-01
 3   -0.875000   -0.329746  1.0873e-01
 4   -0.687500   -0.066571  4.4312e-03
 5   -0.593750    0.068868  4.7423e-03
 6   -0.640625    0.000810  1.3445e-07
 7   -0.664063   -0.032964  1.0861e-03
 8   -0.652344   -0.016098  2.5862e-04
 9   -0.646484   -0.007649  5.7989e-05
```

```
10   -0.643555   -0.003421  1.1181e-05
11   -0.642090   -0.001306  1.1836e-06
12   -0.641357   -0.000248 -4.6009e-07
```

$f(x) = 2^x + x \approx 0$, kun $x \approx -0,641$.

26. Kuvaajan lähdekoodi:

```
from pylab import *
def f(x):
    return 2**(-x) - x
x = arange(-3.0, 4.0, 0.1)
plot(x, f(x))
grid()
show()
```

Kiintopisteiteraatio: Muutetaan yhtälö muotoon $x = 2^{-x}$ ja ratkaistaan tämä yhtälö iteroimalla.

```
#Palauttaa funktion f(x)=2**(-x) arvon.
#AE: x kuuluu reaalilukuihin
def f(x):
    return 2**(-x)

#Tulostaa likiarvoja yhtalon x=2**(-x) ratkaisulle.
#Saa parametrina alkuarvauksen.
#AE: a kuuluu reaalilukuihin
def iteraatio(a):
    h=0.0005
    print "as a         f(a)       (a-f(a)-h)*(a-f(a)+h)"
    d=0          #askel
    while (a-f(a)-h)*(a-f(a)+h)>0:
        taul = (d,a,f(a),(a-f(a)-h)*(a-f(a)+h))
        print "%1d %9.6f %9.6f %12.4e"%taul
        a=f(a)
```

```
        d+=1
    taul = (d,a,f(a),(a-f(a)-h)*(a-f(a)+h))
    print "%1d %9.6f %9.6f %12.4e"%taul
    return " "
#main
iteraatio(0.6)

>>>
as a         f(a)       (a-f(a)-h)*(a-f(a)+h)
0  0.600000  0.659754   3.5703e-03
1  0.659754  0.632986   7.1626e-04
2  0.632986  0.644840   1.4027e-04
3  0.644840  0.639564   2.7593e-05
4  0.639564  0.641907   5.2420e-06
5  0.641907  0.640865   8.3547e-07
6  0.640865  0.641328  -3.5653e-08
```

Yhtälön $x = 2^{-x}$ ratkaisu on $x \approx 0,641$, joka on samalla myös kysytyn yhtälön $2^{-x} - x = 0$ ratkaisu.

27. Kuvaajan lähdekoodi:

```
from pylab import *
def f(x):
    return x*sin(x)
x = arange(0.0, 2*pi, 0.1)
plot(x, f(x))
grid()
show()
```

Newtonin menetelmä:

```
import math
#Palauttaa funktion f(x)=x*sin(x) arvon.
```

```
#AE: x kuuluu reaalilukuihin
def f(x):
    return x*math.sin(x)

#Palauttaa derivaattafunktion f'(x)=sin(x)+x*cos(x) arvon.
#AE: x kuuluu reaalilukuihin
def df(x):
    return math.sin(x)+x*math.cos(x)

#Palauttaa derivaattafunktion f''(x)=2*cos(x)-x*sin(x)arvon.
#AE: x kuuluu reaalilukuihin
def ddf(x):
    return 2*math.cos(x)-x*math.sin(x)

#Tulostaa likiarvoja funktion df nollakohdaksi.
#AE: x_i kuuluu reaalilukuihin.
def newton():
    i = 0
    x_i = 2.0                           #alkuarvaus
    h = 0.000005                        #haluttu tarkkuus
    print "i  x_i        df(x_i)    df(x_i-h)*df(x_i+h)"
    taul = (i,x_i,df(x_i),df(x_i-h)*df(x_i+h))
    print "%1d %9.6f %10.6f %12.4e"%taul
    while df(x_i - h)*df(x_i + h) >= 0:
        x_i = x_i-(df(x_i)/ddf(x_i))    #uusi tarkastelukohta
        i += 1
        taul = (i,x_i,df(x_i),df(x_i-h)*df(x_i+h))
        print "%1d %9.6f %10.6f %12.4e"%taul
    #aariarvokohta
    print "\nAariarvokohta: ","%.6f"%taul[1]
    #funktion aariarvo
    print "Aariarvo: ","%.6f"%f(taul[1])
    return " "
#main
newton()
```

```
>>>
i  x_i        df(x_i)    df(x_i-h)*df(x_i+h)
0  2.000000   0.077004   5.9296e-03
1  2.029048  -0.000785   6.1670e-07
2  2.028758  -0.000000  -1.8278e-10

Aariarvokohta:  2.028758
Aariarvo:  1.819706
```

28.
```
import math

#Palauttaa kysytyn pinta-alan tarkan arvon.
#AE: true
def tarkka():
    return math.pi/2.0

#Palauttaa funktion f(x)= math.sqrt(1-x**2) arvon.
#AE: -1 <= x <= 1
def f(x):
    return math.sqrt(1-x**2)

#Palauttaa suorakulmioiden pinta-alojen summan.
#Saa parametreina alkupisteen,loppupisteen ja jakovalien lkm.
#AE: x0<xn and n>0
def suorakulmio(x0, xn, n):
    d=(xn-x0)/(float)(n)        #jakovalin pituus
    ala=0.0                     #suorakulm. alojen summa
    x1=x0+d/2.0                 #ensimmaisen jakovalin kp
    for j in range((int)(n)):
        xi=j*d
        ala = ala + d*f(x1+xi)
    return ala

#Palauttaa alan likiarvon suhteellisen virheen.
```

```
#Saa parametreina alkupisteen,loppupisteen ja jakovalien lkm.
#AE: x0<xn and n>0
def suhtvirhe(x0, xn, n):
    return 100*(suorakulmio(x0, xn, n)-tarkka())/tarkka()

#Tulostaa alan tarkan arvon, likiarvon ja suht.virheen.
#AE: true
def main():
    print "Tarkka arvo =","%.6f"%tarkka()
    print "Arvio alalle =","%.6f"%suorakulmio(-1.0,1.0,10.0)
    print "Suht. virhe =","%.2f"%suhtvirhe(-1.0,1.0,10.0)
main()
```

Tulokset kymmenellä jakovälillä:

```
>>>
Tarkka arvo = 1.570796
Arvio alalle = 1.585994
Suht. virhe = 0.97
```

Tulokset tuhannella jakovälillä (sama ohjelma kuin edellä, paitsi `suorakulmio` -funktiolle annetaan nyt parametreiksi `suorakulmio(-1.0, 1.0, 1000.0)`, samoin `suhtvirhe` -funktiolle):

```
>>>
Tarkka arvo = 1.570796
Arvio alalle = 1.570812
Suht. virhe = 9.80e-04
```

29.
```
import math

#Palauttaa funktion f(x)=math.sqrt(1+1/x**2) arvon.
#AE: x!=0
def f(x):
```

```
    return math.sqrt(1+1/x**2)

#Palauttaa puolisuunnikkaiden pinta-alojen summan.
#Saa parametreina alkupisteen,loppupisteen ja jakovalien lkm.
#AE: x0<xn and n>0
def puolisuunnikas(x0, xn, n):
    d=(xn-x0)/(float)(n)        #jakovalin pituus
    ala=0.0                     #puolisuunn. alojen summa
    for j in range((int)(n)):
        xi=j*d
        xiseur=(j+1)*d
        ala = ala + (f(x0+xi)+f(x0+xiseur))*d/2.0
    return ala

#Tulostaa pinta-alan likiarvon.
#AE: true
def main():
    print "Arvio alalle =","%.6f"%puolisuunnikas(1.0,2.0,4.0)

main()
```

30.
```
import math

#Palauttaa funktion f(x)=x**2 arvon.
#AE: x kuuluu reaalilukuihin
def f(x):
    return x**2

#Palauttaa likiarvon funktion f(x) maaratylle integraalille.
#Saa parametreiksi integrointivalin [a,b] ja jakovalien maaran.
#AE: a ja b kuuluvat reaalilukuihin ja n on parillinen
def simpson(a, b, n):
    d = (b-a)/(float)(n)
    arvio = 0.0
    for i in range(1, (int)(n), 2):
```

```
            xi = a + d*i
            arvio = arvio + 4*f(xi)
        for i in range(2, (int)(n), 2):
            xi = a + d*i
            arvio = arvio + 2*f(xi)
        arvio = (arvio + f(a) + f(b))*d/3.0
        return arvio

    #Tulostaa integraalin likiarvon.
    #AE: true
    def main():
        print "%.6f"%simpson(0.0, 4.0, 4.0)

    main()
```

31.
```
class Kolmio(object):
    """Kolmio-olion konstruktori"""

    #Kolmion alustaja.
    #AE: kanta >= 0 ja korkeus >= 0
    def __init__(self, kanta, korkeus):
        self.__kanta = kanta
        self.__korkeus = korkeus

    #Palauttaa kolmion kannan pituuden.
    def anna_kanta(self):
        return self.__kanta

    #Palauttaa kolmion korkeuden.
    def anna_korkeus(self):
        return self.__korkeus

    #Palauttaa kolmion pinta-alan.
    def pinta_ala(self):
```

```
        return self.__kanta*self.__korkeus/2

#main
print "Kolmio:"
kanta = float(raw_input("Anna kanta: "))
korkeus = float(raw_input("Anna korkeus: "))
kolmioolio = Kolmio(kanta, korkeus)
print "Kolmion pinta-ala on ", kolmioolio.pinta_ala(),"."
```

32.
```
class Kissaelain(object):
    """Kissaelainolion konstruktori"""

    #Kissaelainolion alustaja.
    #AE: huippunopeus >= 0
    def __init__(self, huippunopeus):
        self.__huippunopeus = huippunopeus

    #Palauttaa huippunopeuden (mph).
    def anna_huippunopeus(self):
        return self.__huippunopeus

    #Palauttaa sadan metrin juoksun ajan sekunteina.
    def aika(self):
        nopeus_ms = (1.609*self.__huippunopeus)/3.6
        return 100.0/nopeus_ms

#main
print """
Kissaelainten nopeudet 100m juoksussa, jos elaimet
juoksisivat koko matkan huippunopeudellaan."""
gepardiolio = Kissaelain(70)
print "Gepardi: ",gepardiolio.aika(),"s"
leijonaolio = Kissaelain(50)
print "Leijona: ",leijonaolio.aika(),"s"
```

```
kotikissaolio = Kissaelain(30)
print "Kotikissa: ",kotikissaolio.aika(),"s"
```

Testaustulos:

```
>>>
Kissaelainten nopeudet 100m juoksussa, jos elaimet
juoksisivat koko matkan huippunopeudellaan.
Gepardi:  3.19630649028 s
Leijona:  4.47482908639 s
Kotikissa:  7.45804847732 s
```

33.
```
import math

class Pankkitili(object):
    """Pankkitiliolioiden konstruktori"""

    #Pankkitiliolioiden alustaja.
    #AE: saldo >= 0
    def __init__(self, saldo):
        self.__saldo = saldo

    #Palauttaa pankkitilin saldon.
    def anna_saldo(self):
        return self.__saldo

    #Palauttaa nostetun rahasumman.
    #AE: summa >= 0
    def nosta_rahaa(self, summa):

        if(summa > self.__saldo):
            nosto = self.__saldo
            self.__saldo = 0.0
            return nosto
```

```
        else:
            self.__saldo -= summa
            return summa

    #Palauttaa talletetun rahasumman.
    #AE: summa >= 0
    def talleta_rahaa(self, summa):
        self.__saldo += summa
        return summa

#main
saldo = input("Anna tilin saldo euroina: ")
tiliolio = Pankkitili(saldo)
print """
1) Nosta rahaa.
2) Talleta rahaa.
3) Tarkista tilin saldo."""
valinta = input("Valitse vaihtoehto 1, 2 tai 3: ")
while valinta == 1 or valinta == 2 or valinta == 3:
    if valinta == 1:
        summa1 = input("Nostettava summa euroina: ")
        nosto = tiliolio.nosta_rahaa(summa1)
        print "Tililta nostettiin ",nosto," euroa."
    elif valinta == 2:
        summa2 = input("Talletettava summa euroina: ")
        talletus = tiliolio.talleta_rahaa(summa2)
        print "Tilille talletettiin ",talletus," euroa."
    else:
        print "Saldo euroina: ",tiliolio.anna_saldo()
    print """
    1) Nosta rahaa.
    2) Talleta rahaa.
    3) Tarkista tilin saldo."""
    valinta = input("Valitse vaihtoehto 1, 2 tai 3: ")
```

```
34. import random

    class Noppa(object):
        """Noppaolion konstruktori"""

        #Nopan alustaja.
        def __init__(self):
            self.__noppa = random

        #Palauttaa nopan silmaluvun.
        def anna_numero(self):
            return self.__noppa.randrange(6)+1

    class Pelaaja(object):
        """Pelaajaolion konstruktori"""

        #Pelaajan alustaja.
        #AE: noppa != None ja tunnus != None
        def __init__(self, tunnus, noppa):
            self.__tunnus = tunnus
            self.__noppa = noppa
            self.__sij = 0
            self.__heitto = 0

        #Heittaa noppaa.
        #Palauttaa viimeisen heiton silmaluvun.
        def heita_noppaa(self):
            self.__heitto = self.__noppa.anna_numero()
            return self.__heitto

        #Palauttaa viimeisen heiton silmaluvun.
        def anna_heitto(self):
            return self.__heitto

        #Siirtaa pelaajaa heittotuloksen verran.
```

```
    #Palauttaa pelaajan sijainnin.
    def siirto(self):
        self.__sij += self.__heitto
        return self.__sij

    #Palauttaa pelaajan sijainnin.
    def anna_sijainti(self):
        return self.__sij

    #Palauttaa pelaajan tunnuksen.
    def anna_tunnus(self):
        return self.__tunnus

class Peli(object):
    """Peliolion konstruktori"""

    #Pelin alustaja.
    #AE: raja pos.kok.luku
    def __init__(self, raja):
        self.__noppa = Noppa()
        pelaaja1 = Pelaaja("#1",self.__noppa)
        pelaaja2 = Pelaaja("#2",self.__noppa)
        self.__pel_lista = [pelaaja1, pelaaja2]
        self.__raja = raja

    #Palauttaa arvon True, jos toinen pelaaja ylittaa rajan.
    #AE: tunnus != None
    def voitto(self, tunnus):
        sij_kohta = self.__pel_lista[tunnus].anna_sijainti()
        return sij_kohta >= self.__raja

    #Pyorittaa pelia, kunnes toinen pelaaja voittaa.
    def silmukka(self):
        joku_voittanut = False
        while(not joku_voittanut):
```

```
            for i in range(len(self.__pel_lista)):
                tunnus = self.__pel_lista[i].anna_tunnus()
                print "Pelaaja ",tunnus,":"
                raw_input("Heita noppaa painamalla Enter.")
                heitto = self.__pel_lista[i].heita_noppaa()
                tulos = self.__pel_lista[i].anna_heitto()
                print "Pelaaja ",tunnus," heitti ",tulos
                sij = self.__pel_lista[i].siirto()
                print "Pelaaja ",tunnus," siirtyi kohtaan ",sij

                if self.voitto(i):
                    print "Pelaaja ",tunnus," voitti!"
                    joku_voittanut = True
                    break

    #Aloittaa pelin.
    def aloitus(self):
        self.silmukka()

#main
raja = input("Anna raja: ")
peli = Peli(raja)
peli.aloitus()
```

35. Lisätehtävä:

```
import math

#Palauttaa funktion f(ex,yy)=yy-2*math.sin(ex) arvon.
#AE: ex ja yy kuuluvat reaalilukuihin
def f(ex,yy):
    return yy-2*math.sin(ex)

#Tulostaa diff.yhtalon ratkaisulle tarkat arvot ja likiarvot.
#Saa parametreina alkuarvot x0, y0, askelpituuden ja valin
```

```
#loppupisteen.
#AE: x,y,b kuuluvat reaalilukuihin ja h>0
def rungekutta(x, y, h, b):
    print "xi   yi_tarkka yi_likiarvo"
    tarkka = math.cos(x)+math.sin(x)
    print "%.1f %9.6f %9.6f"%(x, tarkka, y)
    n = b/(float)(h) #askelten maara
    for i in range((int)(n)):
        k1 = f(x,y)
        k2 = f(x+h/2.0 , y+k1*h/2.0)
        k3 = f(x+h/2.0 , y+k2*h/2.0)
        k4 = f(x+h , y+k3*h)
        y = y + (k1+2*k2+2*k3+k4)*h/6.0
        x = x + h
        tarkka = math.cos(x)+math.sin(x)
        print "%.1f %9.6f %9.6f"%(x, tarkka, y)
    return " "
#main
rungekutta(0.0, 1.0, 0.5, 2.0)
```

Tulokset arvolla $h = 0,5$:

```
>>>
xi   yi_tarkka yi_likiarvo
0.0  1.000000  1.000000
0.5  1.357008  1.357209
1.0  1.381773  1.382455
1.5  1.068232  1.069769
2.0  0.493151  0.496059
```

Kun $h = 0,1$, kohdassa $2,0$ saadaan tulos
`2.0  0.493151  0.493157`. Huomataan, että jopa askelpituudella $h = 0,5$ saadut tulokset ovat merkittävästi parempia kuin Eulerin menetelmällä ja keskipistemenetelmällä saadut. Askelpituutta $h = 0,1$ käy-

tettäessä kohdassa $2,0$ saatu tulos antaa jopa viisi oikeaa desimaalia.

36. Lisätehtävä:

```
from numpy import *
a = array([[1.0,-1.0,1.0], [1.0,1.0,0.0], [2.0,-1.0,-1.0]])
b = array([1.0,2.0,0.0])
x = linalg.solve(a,b)
print x
```

B Liite 2: Tuntijakosuunnitelma

Kurssin ohjeellinen tuntijakosuunnitelma on laadittu olettaen, että lukiossa on yhden kurssin aikana 20 oppituntia, joiden pituus on 75 minuuttia. Suunnitelma kertoo alustavasti, mitä asioita milläkin tunnilla on tarkoitus käsitellä.

1.tunti
Yleistä tietoa kurssista
Luku 2 Ohjelmointi
Luku 3.2 Peruslaskutoimituksia

2.tunti
Luku 3.3 Ensimmäinen ohjelma
Luku 3.4 Skriptien käyttö
Luku 3.5 Ohjelmien testaaminen

3.tunti
Luku 3.6 Tyypit ja muuttujat
Luku 3.7 Taulukot ja listat

4.tunti
Luku 3.8 Ehto- ja toistorakenteet

5.tunti
Luku 3.9 Valmiit moduulit
Luvun 3 kertausta

6.tunti
Luku 4.1 Lukujonot - toistorakenteiden kertausta

7.tunti
Luku 4.2 Vektorilaskentaa - Numerical Python

8.tunti
Luku 4.3.1 Toisen asteen yhtälön ratkaisukaava
Luku 4.3.2 Johdatus funktioiden käyttöön

9.tunti
Luku 4.3.3 Toisen asteen yhtälön ratkaisukaava funktiona

10.tunti
Luku 4.4 Kuvaajien piirtäminen - Matplotlib

11.tunti
Luku 4.5.1 Moduuli random
Luku 4.5.2 Kolikonheittosimulaattori alkuun

12.tunti
Luku 4.5.2 Kolikonheittosimulaattori loppuun
Luku 4.5.3 Piirakkakuvion piirtäminen

13.tunti
Luku 4.6 Kertoman laskeminen - Rekursio
Pohjustusta lukuun 4.7 Numerisia menetelmiä

14.tunti
Luku 4.7.1 Numeerinen derivaatta
Luku 4.7.2 Yhtälön nollakohdat - Newtonin menetelmä alkuun

15.tunti
Luku 4.7.2 Yhtälön nollakohdat - Newtonin menetelmä loppuun
Luku 4.7.3 Numeerinen integrointi - Keskipistesääntö ja puolisuunnikassääntö

16.tunti
Luku 4.7.4 Numeerinen integrointi - Simpsonin sääntö

17.tunti
Luku 4.8.1 Mitä olio-ohjelmointi on?
Luku 4.8.2 Neliöolio, ympyräolio ja pallo-olio alkuun

18.tunti
Luku 4.8.2 Neliöolio, ympyräolio ja pallo-olio loppuun

19.tunti

Luku 4.8.3 Platonin kappaleet olio-ohjelmoinnilla

20.tunti

Kertausta

C Liite 3: Kuvaajien lähdekoodeja

1. Mandelbrotin joukko (Kuva 1):

```
#tiedostossa fraktaali.py
from pylab import *
from fonttikoko import *
ax = fonttikokomuutos()

#Palauttaa ruudukon, jonka jokainen piste on varitetty sen mukaan,
#pysyyko sen etaisyys origosta joka iteraatiolla pienempana kuin 2
#vai ei. Jos ei, niin variin vaikuttaa se, kuinka monennella
#iteraatiolla raja 2 ylittyy.
#Saa parametreina kuvan koon koordinaatistossa (n on vaakamitta ja
#m pystymitta), iterointien maksimimaaran ja laskenta-alueen rajat.
#AE: n,m,itermax > 0 ja xmin, xmax, ymin, ymax reaalilukuja
def mandelbrot(n, m, itermax, xmin, xmax, ymin, ymax):
        #ix: 00...0 ja iy: 01...m-1
        #    11...1        01...m-1
        #     ....          ....
        #    n-1..n-1      01...m-1
        ix, iy = mgrid[0:n, 0:m]
        #x ja y ovat x- ja y-arvot listan joka pisteessa.
        #linspace: n lukua tasaisin valein valilta xmin,xmax.
        x = linspace(xmin, xmax, n)[ix]
        y = linspace(ymin, ymax, m)[iy]
        #complex(real, imag), c = i
        c = x+complex(0,1)*y
        #Luodaan ruudukolle pohja.
        kuvapisteet = zeros(c.shape, int)
        #Tasoitetaan listat samankokoisiksi.
        ix.shape = n*m
        iy.shape = n*m
        c.shape = n*m
        z = copy(c)
        #Iteroidaan z = z**2 + c.
```

```
        for i in range(itermax):
            #listojen kertolasku alkioittain
            multiply(z, z, z)
            #listojen yhteenlasku alkioittain
            add(z, c, z)
            #pisteet, joiden etaisyys origosta kasvaa rajatta
            pisteet = abs(z)>2.0
            #Nama pisteet varitetaan jokainen omalla
            #iteraationumerollaan i.
            #Lisataan i+1, koska i kulkee aina yhden numeron
            #jaljessa.
            kuvapisteet[ix[pisteet], iy[pisteet]] = i+1
            #pisteet, joiden etaisyys origosta pysyy pienempana
            #tai yhtasuurena kuin 2
            pisteet = abs(z)<=2.0
            #Valitaan listoista z, ix, iy ja c pisteet, joita
            #on viela iteroitava.
            z = z[pisteet]
            ix = ix[pisteet]
            iy = iy[pisteet]
            c = c[pisteet]
        return kuvapisteet

#Piirtaa fraktaalin kuvan.
#AE: true
def main():
        I = mandelbrot(400, 400, 100, -2, .5, -1.25, 1.25)
        #vari
        I[I==0] = 95
        #T = transpoosi
        kuvapisteet = ax.imshow(I.T, origin='lower left')
        show()
main()
```

2. Mandelbrotin joukko (Kuva 2) piirretään muuten samalla koodilla, mutta funktiota kutsutaan eri parametrein:
   `I = mandelbrot(400, 400, 100, -1.6, -1.2, -0.15, 0.15)`.

3. Esimerkki 21 (Kuva 3):

```
#tiedostossa mplfunktioidenkasvu.py
from pylab import *
from fonttikoko import *
ax = fonttikokomuutos()
def f(x):
    return x**2
def g(x):
    return 2**x
x = arange(0.0, 10.0, 0.1)
fx, gx = ax.plot(x, f(x), x, g(x))
setp(fx, linestyle='-', linewidth=3, color='r')
setp(gx, linestyle='--', linewidth=1, color='k')
grid()
show()
```

4. Alla oleva funktio muuttaa Matplotlibin avulla piirrettyjen kuvaajien x- ja y-koordinaattien fonttikokoa. Funktio on määritelty käyttämällä apuna lähteessä [22] esitettyä ohjelmakoodia.

```
#tiedostossa fonttikoko.py
from pylab import*
def fonttikokomuutos():
    ax = axes()
    xlabels = ax.get_xticklabels()
    xlabel0 = xlabels[0]
    xlabel0.get_fontsize()
    xlabel0.set_fontsize(15.5)
    for xlabel_i in ax.get_xticklabels():
        xlabel_i.set_fontsize(15.5)
    ylabels = ax.get_yticklabels()
```

```
ylabel0 = ylabels[0]
ylabel0.get_fontsize()
ylabel0.set_fontsize(15.5)
for ylabel_i in ax.get_yticklabels():
    ylabel_i.set_fontsize(15.5)
return ax
```

D Liite 4: Lataus- ja asennusohjeet

Internet-osoitteesta www.python.org/download löydät kaikki tämänhetkiset versiot Python-tulkista. Jos tarjolla on useampia eri versioita, kannattaa yleensä valita uusin mahdollinen. Samasta versiosta on yleensä tarjolla useita asennustiedostoja. Valitse näistä se vaihtoehto, joka vastaa tietokoneesi käyttöjärjestelmää. Tässä esitetty asennusohje on laadittu suomenkieliselle 64-bittiselle Windows 7 -käyttöjärjestelmälle.

Lataa asennustiedosto klikkaamalla sitä hiirellä kahdesti ja valitsemalla avautuvasta ikkunasta kohta Tallenna tiedosto. Kun lataus on valmis, paina Lataukset-ikkunassa näkyvää Python-kuvaketta hiiren oikealla painikkeella ja valitse luettelosta kohta Avaa. Valitse avautuvasta ikkunasta kohta Suorita. Asenna ohjelma seuraamalla näytölle avautuvan ikkunan ohjeita. Asetukset voit pitää vakioina, ellet sitten halua esimerkiksi asentaa Python-tulkkia johonkin toiseen tiedostoon kuin oletuksena annetaan. Paina jokaisen vaiheen jälkeen Next-painiketta. Asennuksen päätyttyä paina Finish-painiketta.

Muut työssä esitetyt ohjelmat ladataan vastaavin ohjein luvussa 3.1 ja lähdeluettelossa mainituista Internet-osoitteista.