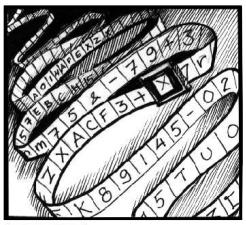


Ohjelmoinnin alkeita Python-kielellä

Antti Rasila
Tutkija
Matematiikan ja tilastotieteen laitos, Helsingin yliopisto

Johdanto

Tämän artikkelin tarkoituksena on esitellä lukijalle Python-ohjelmointikieltä matemaattisten sovellusten näkökulmasta. Artikkeli on tarkoitettu ohjelmoinnin peruskurssiksi, jossa käydään läpi yksinkertaisten ohjelmien kirjoittamista Python-kielellä, mutta sen voi vaihtoehtoisesti ymmärtää Pythonin yleisesittelyksi jo johonkin toiseen ohjelmointikieleen perehtyneelle lukijalle. Tarkoituksena on kirjoittaa artikkeliin myöhemmin jatkoa, jonka sisällöksi olen kaavaillut yksinkertaisten numeerisen matematiikan sovellusten kirjoittamista *Numerical Python* -kirjastoa käyttämällä sekä graafisten käyttöliittymien kirjoittamista ja matemaattisen datan visualisointia *Tkinter*-laajennuksen avulla.

Python on interaktiivinen olio-ohjelmointikieli, jossa yksinkertainen ja selkeä syntaksi yhdistyy kokeneemmankin ohjelmoijan arvostamiin monipuolisiin ominaisuuksiin. Python-kieli on helppo oppia, ohjelmien kirjoittaminen on sillä nopeaa ja virheiden etsiminen valmiista ohjelmasta vaivatonta. Python sopii hyvin ensimmäiseksi ohjelmointikieleksi ja on osoittautunut suosituksi monenlaisissa ohjelmointiprojekteissa.

Tavallisimmin Pythonia käytetään erilaisten tietoliikenne- ja ylläpitosovellusten kirjoittamisessa. Pythoniin on saatavissa laajennuksia, joiden avulla sitä voi tehokkaasti käyttää moniin muihinkin tarkoituksiin. Yksi tällainen laajennus on tämän artikkelin seuraavassa osassa esiteltävä *Numerical Python* -kirjasto, jonka avulla Python on laajennettavissa ominaisuuksiltaan lähes kaupallista MATLAB-ohjelmistoa vastaavaksi numeerisen matematiikan ohjelmointiympäristöksi. Osoitteesta `http://www.python.org` ladattavissa oleva Python-tulkki on vapaasti levitettävissä kaupalliseen ja ei-kaupalliseen käyttöön. Tulkki on saatavissa kaikille tavallisimmille laitteistoille ja käyttöjärjestelmille, joita ovat esimerkiksi Linux, Windows, Apple ja UNIX. Yhdessä ympäristössä kirjoitetut Python-ohjelmat toimivat yleensä muissa ilman muutoksia (poikkeuksena esimerkiksi käyttöjärjestelmän palveluita `os.system`-kutsun kautta käyttävät ohjelmat).

Kokonais- ja liukuluvut

Monien muiden tulkattujen ohjemointikielten tapaan Pythonia voi käyttää interaktiivisesti, eli eräänlaisena laskimena. Tällöin komennot syötetään suoraan Python-tulkin kehotteeseen. Python-tulkista poistuminen tapahtuu painamalla `CTRL+D`[1]. Interaktiivinen

[1] Tämä on UNIX-maailmasta tuttu tiedoston loppumiseen viittaava kontrollimerkki.

käyttötapa sopii hyvin kielen ominaisuuksiin tutustumiseen ja ohjelmoinnin harjoitteluun. Ohjeita jonkin käskyn tai kirjaston toiminnasta saa kirjoittamalle `help([käskyn nimi])`. Seuraavassa yksinkertainen esimerkki Python-tulkin interaktiivisesta käytöstä:

```
>>> 1+1
2
>>> 123+456
579
>>>
```

Lukujen esittämiseen tietokoneessa käytetään kokonaisluku- (Pythonissa `int`) ja liukulukuesityksiä (vast. `float`). Liukulukuja käytetään tavallisesti murto- ja irrationaalilukujen esittämiseen. Liukuluku on kuitenkin aina tarkkuudeltaan rajoitettu approksimaatio, joka vastaa likimäärin taskulaskimista tuttua luvun eksponenttiesitystä. Äärellinen esitystarkkuus aiheuttaa ongelmia, jos esimerkiksi lasketaan yhteen suuruusluokaltaan paljon toisistaan poikkeavia liukulukuja, kuten 10^{-10} ja 10^{10}. Erityisesti kahden liukuluvun yhtäsuuruuden vertaaminen suoraan ei yleensä johda toivottuun lopputulokseen. Tehtäessä laskutoimituksia kokonaisluvuilla lopputulos on kokonaisluku, liukuluvuilla vastaavasti liukuluku. Tämä voi aiheuttaa odottamattomia lopputuloksia. Liukulukuesitystä käytettäessä kokonaislukujen kokonaisosan jälkeen merkitään `.` (tai `.0`).

```
>>> 7/3
2
>>> 7./3.
2.3333333333333335
>>> 7./9.-(1./3.)*7./3
1.1102230246251565e-16
```

Koska `7` ja `3` käsitellään ensimmäisellä rivillä kokonaislukuina, jakolaskun lopputulos on myös kokonaisuluku, eli osamäärän kokonaisosa. Liukulukujenkaan laskutoimituksen tuloksena ei äärellisen esitystarkkuuden vuoksi saada matemaattisesti oikeaa arvoa `2.3333...`. Tämä aiheuttaa ongelmia, jos halutaan esimerkiksi testata ovatko, kahden eri lausekkeen antamat arvot samat. Tällöin pitää samoiksi arvoiksi hyväksyä ne tapaukset, joissa arvot ovat riittävän lähellä toisiaan; käsitteen riittävän lähellä tulkinta riippuu sovelluksesta.

Muuttujista

Sijoittaminen muuttujaan tapahtuu sijoitusoperaattorin `=` avulla. Jos laskutoimituksen tai muun operaation lopputulos sijoitetaan muuttujaan, sitä ei tulosteta. Useampia muuttujia voi sijoittaa samanaikaisesti erotamalla muuttujat ja sijoitettavat arvot pilkulla.

```
>>> a,b=5,6
>>> a
5
>>> b
6
```

Jokaisella Python-muuttujalla (kuten muillakin lausekkeilla) on tyyppi, jonka voi selvittää komennolla `type`.

Merkkijonot ja listat

Kokonais- ja liukulukujen ohella muita hyödyllisiä muuttujia ovat esimerkiksi merkkijonot (`str`) ja listat (`list`). Merkkijonon merkkeihin osamerkkijonoihin voi viitata hakasulkujen `[]` avulla. Ilmaisu `[j:k]` tarkoittaa suljettua väliä j:stä $(k-1)$:een. Jos jompikumpi päätepiste puuttuu, tarkoitetaan väliä j:stä loppuun tai k:sta alkuun. Merkkijonon pituuden voi selvittää käskyllä `len`. Merkkijonojen (ja listojen) indeksointi Pythonissa alkaa 0:sta; indekseihin merkkijonon lopusta lukien viitataan negatiivisilla luvuilla. Merkkijonoja voi yhdistää operaation `+` avulla. Esimerkki merkkijonojen käytöstä:

```
>>> s='merkkijono'
>>> len(s)
10
>>> type(s)
<type 'str'>
>>> s[0]
'm'
>>> s[0:5]
'merkk'
>>> s[-5:]
'ijono'
>>> s[-1]
'o'
>>> t=' ja toinen merkkijono'
>>> s+t
'merkkijono ja toinen merkkijono'
```

Listat ovat indeksöityjä muuttujajonoja. Samaan listaan voidaan tallettaa erityyppisiä arvoja. Listan käsittely ja listan alkoihin viittaaminen muistuttaa merkkijonojen vastaavia operaatioita. Listan alkioilla on luonnollisesti omat tyyppinsä.

```
>>> li=['nolla','yksi','kaksi',3,4,5.0,6.0]
>>> li
['nolla', 'yksi', 'kaksi', 3, 4, 5.0, 6.0]
>>> li[1]
'yksi'
>>> li[2:4]
['kaksi', 3]
>>> len(li)
7
>>> type(li)
<type 'list'>
>>> type(li[2])
```

```
<type 'str'>
>>> type(li[6])
<type 'float'>
```

Tyyppimuunnokset

Muunnoksia erityyppisten muuttujien välillä voi tehdä tyyppimuunnosfunktioiden avulla. Niiden syntaksi on `tyyppi(·)`. Kaikkia tyyppejä ei voi luonnollisesti muuttaa toisikseen, ja lisäksi jotkin tyyppimuunnokset voivat hävittää informaatiota.

```
>>> s1='0.5'
>>> int(s1)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: invalid literal for int(): 0.5
>>> float(s1)
0.5
>>> int(float(s1))
0
>>> float('1.0')
1.0
>>> str(1.0)
'1.0'
```

Tulostaminen `print`-käskyn avulla

Muuttujan arvon voi tulostaa käskyllä `print <muuttujan nimi>`. Käskyn yhteydessä on mahdollista antaa tulostuksen muodon määräävä merkkijono[2], joka erotetaan tulostettavista muuttujista prosenttimerkillä. Jos tulostettavia muuttujia on useita, niiden ympärille pitää merkitä sulkeet. Myös listan voi tulostaa `print`-käskyllä.

```
>>> a,b=5,6
>>> print a,b
5 6
>>> print '%f %f' % (a,b)
5.000000 6.000000
>>> print [a,b]
[5, 6]
```

Tulostuksen formatoinnin määrittämisessä voidaan käyttää mm. seuraavia optioita:

- `%d` kokonaisluku,
- `%f` liukuluku,
- `%e` liukuluku, eksponenttiesitys,
- `%s` merkkijono.

[2]Tämä vastaa C-kielen `printf`-käskyn toimintaa.

Tulostettavan merkkikentän pituuden ja mahdollisen esitystarkkuuden voi määrittää `print`-käskyn yhteydessä. Nämä annetaan pisteellä erotettuna prosenttimerkin jälkeen.

```
>>> a=1.23456789
>>> print "%10.6f" % a
  1.234568
>>> print "%10.3f" % a
     1.235
>>> print "%6.3f" % a
 1.235
>>> print "%6.3e" % a
1.235e+00
>>> print "%6.3d" % a
   001
```

Python-ohjelmat ja syötteen lukeminen käyttäjältä

Python-kielisten ohjelmatiedostojen tunnisteena käytetään tarkennetta `.py`. Ohjelmat ovat itsessään tavallisia tekstitiedostoja, ja niiden editoimiseen voi käyttää mitä hyvänsä ohjelmointiin soveltuvaa tekstieditoria, kuten esimerkiksi `notepad` tai `emacs`. Kommenttirivejä merkitään Pythonissa #-merkillä ja käskyn voi jakaa useammalle riville kenoviivan \ avulla. Kommenteissa olevat skandinaaviset merkit voivat aiheuttaa ongelmia joissakin järjestelmissä. Windowissa Pythontulkki asettaa itsensä automaattisesti `.py`-tiedostojen oletusarvoiseksi avausohjelmaksi. Ohjelmien ajaminen sujuu klikkaamalla ohjelmatiedoston kuvaketta. Ohjelmia voi ajaa myös komentoriviä käyttäen kirjoittamalla `python ohjelma.py`.

Pythonissa syötteen lukeminen käyttäjältä tapahtuu `input`-käskyllä. Käsky ottaa parametrinaan merkkijonon, joka on käyttäjälle syötettävä kehote.

```
>>> x=input('Anna luku: ')
Anna luku: 3
>>> print x
3
```

Käyttäjän antama syöte voi aiheuttaa myös ongelmia. Saattaa olla, että lukua pyydettäessä käyttäjä onkin antanut jonon kirjaimia. Tämän voi kuitenkin helposti estää tarkastamalla annetun syötteen tyyppi. Siihen tarvitaan kuitenkin seuraavaksi esitettäviä ehtorakenteita.

Ehto- ja toistorakenteet

Toistorakenteen `for` syntaksi poikkeaa Pythonissa useimmista muista ohjelmointikielistä. Toistolauseelle ei anneta parametriksi ehtoa, vaan lista, jonka jokaiselle alkiolle toisto suoritetaan. Toistettavaan haaraan liittyvät käskyt merkitään sisennyksen (tabuloinnin) avulla, eli `for [muuttuja] in [lista]: ....`

Jos asia halutaan toistaa k kertaa, voidaan käyttää käskyä `range(k)`, joka tuottaa listan, jossa ovat luvut $0, \ldots, k-1$. Parametrina voidaan antaa myös toinen luku j. Tässä tapauksessa `range(j,k)` tuotaa listan luvuista $j, \ldots, k-1$.

```
# Esimerkki: Lukujen toiset potenssit
for x in range(1,5):
        xx=x*x
        print '%d*%d = %d' % (x,x,xx)
```

```
Testiajo:

1*1 = 1
2*2 = 4
3*3 = 9
4*4 = 16
```

Lähes kaikista ohjelmointikielistä löytyvän `if-then-else` -rakenteen syntaksi Pythonissa on seuraava:
`if [ehto]: ...`
`elif [ehto2]: ...`
`else: ...`

Näistä `elif` ja `else` -haarat voi jättää pois. Haarassa `if` olevat käskyt suoritetaan, jos ehto toteutuu. Jos ehto ei toteudu, tutkitaan järjestyksessä `elif`-haarojen ehtoja. Jos mikään annetuista ehdoista ei toteutunut, suoritetaan `else`-haaran käskyt. Samaan haaraan kuuluvat käskyt merkitään `for`-lauseen tapaan sisennyksen avulla.

Tavallisimpia ehtoja ovat vertailut `<`, `<=`, `==`, `!=`, `>=` ja `>`. Huomaa, että yhtäsuuruutta verrattaessa käytetään merkintää `==`, jotta tehtäisiin ero sijoitusoperaation `=` kanssa[3]. Erisuuruudelle käytetään merkintää `!=`.

Toistorakennetta `while` käytetään, kun jotakin halutaan toistaa niin kauan, että jokin ehto ei enää ole voimassa. Tämä rakenne on erityisen hyödyllinen luettaessa syötettä käyttäjältä tai tiedostosta. Tällöin halutaan ehkä jatkaa lukemista, kunnes tiedosto on loppunut tai odottaa, että käyttäjä on antanut haluttujen rajojen puitteissa olevan syötteen. While-lauseen syntaksi on `while [ehto]: ....` Päättymättömän silmukan välttämiseksi täytyy toistettavan rakenteen sisällä luonnollisesti olla jotakin, mikä asettaa annetun ehdon epätodeksi, kun haluttu määrä toistoja on suoritettu.

```
# Esimerkki: Parilliset ja parittomat luvut
x='x'
# Syotetta kysytaan uudelleen, kunnes kayttaja
# antaa kokonaisluvun
while type(x)!=type(1):
        x=input('Anna kokonaisluku: ')
        if type(x)!=type(1): print x,' \
          ei ole kokonaisluku.\n'
if x%2==0: print "Luku %d on parillinen" % x
else: print "Luku %d on pariton" % x
```

```
Testiajo:

Anna kokonaisluku: 0.5
0.5  ei ole kokonaisluku.

Anna kokonaisluku: 4
Luku 4 on parillinen
```

Ohjelmassa esiintyvä merkintä `n%k` tarkoittaa n:n jakojäännöstä jaettaessa k:lla.

Kirjastojen käyttö

Monet hyödyllisistä Python-kielen ominaisuuksista on sijoitettu kirjastoihin, jotka täytyy ladata `import`-käskyllä ennen käyttöä. Kirjastoja ovat esimerkiksi matemaattisia funktioita sisältävä `math` ja käyttöjärjestelmään liittyviä käskyjä sisältävä kirjasto `os`. Kirjastoja voi helposti tehdä itsekin kirjoittamalla Python-tiedoston, joka sisältää pelkkiä funktioita. Kirjasto aktivoidaan käskyllä
`from [kirjasto] import [metodi]` tai
`import [kirjasto]`.
Ensimmäisessä tapauksessa metodit (funktiot) haetaan kirjastosta oletusnimiavaruuteen, eli metodeita voi kutsua suoraan kirjoittamalla `metodi([parametrit])`, jälkimmäisessä tapauksessa kirjaston metodeja pitää kutsua nimellä `[kirjasto].[metodi]([parametrit])`.

```
>>> from math import sin
>>> sin(3)
0.14112000805986721
>>> import math
>>> math.cos(3)
-0.98999249660044542
```

Jos kirjastosta halutaan hakea kaikki metodit, niin metodin paikalle voidaan merkitä `*`, siis esimerkiksi `from math import *`.

[3] Tämä ero on tärkeä, koska joissakin ohjelmointikielissä esimerkiksi ehto `if (x=1)` on aina tosi. Pythonissa sijoituslauseketta ei voi kirjoittaa ehtolauseeseen, esimerkiksi `if x=1: ...` antaa virheilmoituksen.

Funktioiden määrittely

Python-kielessä funktioiden määrittely tapahtuu `def`-rakenteen avulla. Käskyn syntaksi on `def [funktion nimi]([parametrilista]):`. Toistorakenteiden tapaan funktioon kuuluvat käskyt erotetaan sisennyksellä.

```
# Esimerkki: Toisen asteen yhtalon ratkaiseminen
from math import *

def solve2(a,b,c):
        D=b*b-4*a*c
# Liukulukua ei voi suoraan verrata nollaan
        if abs(a)<1.e-6:
                print "Virhe: Parametrin ",\
                      "a arvona ei saa olla 0.\n"
                return []
        elif abs(D)<1.e-6: return [-b/(2.*a)]
        elif D<0.: return []
        else: return [(-b+sqrt(D))/(2.*a), \
                      (-b-sqrt(D))/(2.*a)]
```

Testiajo:

```
>>> from esim3 import *
>>> solve2(1,0,0)
[0.0]
>>> solve2(1,0,1)
[]
>>> solve2(1,0,-1)
[1.0, -1.0]
```

Funktio `solve2` ratkaisee toisen asteen yhtälön reaaliset juuret. Funktion parametrit ovat kolme lukua a, b ja c, jotka vastaavat vakiokertoimia ratkaistavassa yhtälössä

$$ax^2 + bx + c = 0.$$

Funktio palauttaa listan yhtälön juurista. Jos reaalisia juuria ei ole, palautetaan tyhjä lista. Esimerkkiajon ensimmäinen tapaus vastaa yhtälöä $x^2 = 0$ (juuri 0:ssa), toinen $x^2 + 1 = 0$ (ei reaalisia juuria) ja kolmas $x^2 - 1 = 0$ (kaksi juurta, ± 1).

Satunnaisluvuista

Satunnaislukujen generoimiseen on Pythonissa käytössä kirjasto `whrandom`. Tästä kirjastosta löytyvä metodi `random` generoi tasaisesti jakautuneita satunnaislukuja (liukulukuja) puoliavoimelta väliltä $[0, 1)$. Vastaavasti metodi `uniform(a,b)` generoi satunnaislukuja väliltä $[a, b)$. Satunnaisten kokonaislukujen generoimiseen on käytössä metodi `randint(a,b)`, joka generoi satunnaisia kokonaislukuja väliltä $[a, b]$, eli mukaanlukien välin päätepisteet.

```
# Esimerkki: Nopanheiton simulointi
# Heitetaan noppaa 100 kertaa ja lasketaan
# silmalukujen jakauma
from whrandom import *
jak=[0,0,0,0,0,0]
for j in range(100):
        k=randint(1,6)
        jak[k-1]=jak[k-1]+1

for j in range(6): print '%d: %d'% (j+1,jak[j])
```

Testiajo:

```
1: 14
2: 15
3: 13
4: 18
5: 18
6: 22
```

Tiedostojen lukeminen ja kirjoittaminen

Jotta tiedostoa voitaisiin lukea tai siihen kirjoittaa, tiedosto on avattava `open`-metodilla. Käsittelyn jälkeen tiedosto suljetaan `close`-metodilla. Tiedoston jättäminen sulkematta voi aiheuttaa erilaisia virhetilanteita, kuten tiedostoon kirjoitetun datan menettämisen osittain tai kokonaan.

Käskyn `open` syntaksi on seuraava: `[osoitin]=open('[tiedosto]', [moodi])`, missä `[tiedosto]` on avattavan tiedoston nimi ja `[moodi]` on joko `'r'` tai `'w'`, riippuen siitä, halutaanko tiedostoa lukea vai kirjoittaa siihen. Tiedostoon voidaan myöhemmin viitata nimen `[osoitin]` avulla. Tiedoston lukemiseen ja kirjoittamiseen on käytössä esimerkiksi `read`, `readlines` ja `write` -metodit.

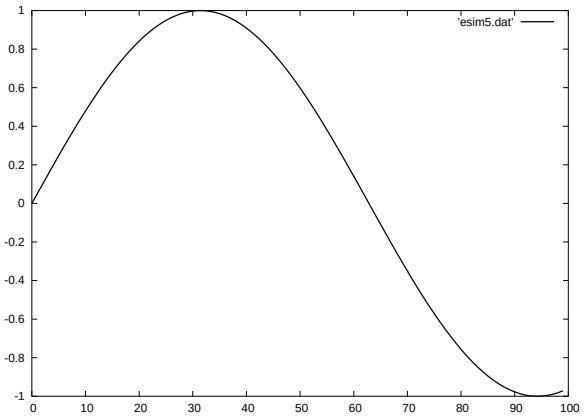
```
# Esimerkki: funktion arvojen kirjoittaminen
# tiedostoon
from math import *
import sys
dataf='esim5.dat'
fp=open(dataf,'w') # avataan tiedosto

# kirjoitetaan tiedotoon funktion sin(x) arvoja
for j in range(100):
        fp.write('%10.6f\n' % sin(0.05*j))
fp.close()          # suljetaan tiedosto
fp=open(dataf,'r')  # avataan tiedosto uudestaan
                    # lukemista varten
vals=fp.readlines() # luetaan tiedosto: tuloksena on
                    # lista merkkijonoja, vastaten
                    # tiedoston riveja
for v in vals: print '%10.6f' % float(v)
fp.close()          # suljetaan tiedosto
```

```
Testiajo:

  0.000000
  0.049979
  0.099833
  0.149438
  0.198669
  0.247404
...
```

Funktion kuvaajan voi piirtää esimerkiksi käyttäen ilmaista Gnuplot-ohjelmaa[4]. Gnuplotille annettava käsky on `plot 'tiedosto.dat' w l`. Esimerkin tapauksessa saadaan seuraava kuva:



Linkkejä

- Pythonin kotisivu
  `http://www.python.org/`
- Aloittelijan opas Pythoniin – linkkejä alkuun pääsemiseksi
  `http://www.python.org/topics/learn/`
- Python Reference Card (pikaohje)
  `http://ourworld.compuserve.com/homepages/JasonRandHarper/PyQuickRef.pdf`
- Python FAQ (usein esitettyjä kysymyksiä)
  `http://www.python.org/doc/FAQ.html`
- Johdatus Pythoniin
  `http://archive.dstc.edu.au/python/python/Introduction.html`
- Guido van Rossumin (Pythonin isä) Python-opas
  `http://archive.dstc.edu.au/python/doc/tut/`
- Numerical Python
  `http://www.numpy.org`
- Gnuplotin kotisivu
  `http://www.gnuplot.info`

[4]Ohjeita Gnuplotin käyttöön löytyy sivulta `http://www.ucc.ie/gnuplot/gnuplot.html`.