

T-106.3105 Ohjelmoinnin harjoitustyö (2 op)

Kurssin järjestelyt, skriptaus, Python

Sami Liedes

29. huhtikuuta 2008

Tänään luvassa

- ▶ Kurssin esittely
- ▶ Skriptaus: Mitä, miksi ja milloin?
- ▶ Skriptikielten ja erityisesti Pythonin ominaisuuksista
- ▶ Lyhyt johdanto Python-kieleen
 - ▶ Noudattelee webissä olevaa Python-tutoriaalia (<http://docs.python.org/tut/tut.html>)
 - ▶ Aikataulusyistä ei mene yhtä syvälle
 - ▶ Tutoriaali kannattaa käydä läpi jos haluaa oppia Pythonia kunnolla

Kurssin esittely

- ▶ Vapaaehtoinen, palkkio 2 op
- ▶ Ainoana suorituksena kaksi pientä ohjelmointitehtävää
- ▶ Tehtävät saa tehdä Pythonilla, Rubyllä tai Perlillä
 - ▶ Kurssihenkilökunnalta saa apua Pythonin kanssa
- ▶ Tehtävät kurssin webbisivuilta
- ▶ Palautus sähköpostitse
- ▶ Deadline su 18.5.2008 klo 24:00

Skriptaus: Mitä, miksi ja milloin? 1/2

- ▶ Jotkut asiat eivät ole käteviä C-kielellä
 - ▶ Merkkijonojen käsittely
 - ▶ Muiden ohjelmien ja komentojen suorittaminen
- ▶ Näitä tehtäviä varten on kehitetty skriptikieliä
- ▶ Aivan alunperin skriptikielet kehitettiin kääntämisen, linkittämisen ja ohjelmien ajamisen automatisointiin
- ▶ Näissä käytöissä edelleen esim. MS-DOSin ja Windowsin .bat-tiedostot ja Unixin (Linuxin) shell-skriptit

Skriptaus: Mitä, miksi ja milloin? 2/2

- ▶ Yleensä C:tä korkeamman tason tulkattavia kieliä
- ▶ Sisältävät usein kehittyneitä merkkijononkäsittelyrutiineita
- ▶ Yleensä kieleen on rakennettu sisään kehittyneitä rakenteita, kuten hash-taulut, linkitettyt listat
- ▶ Skriptit ovat yleensä paljon pienempiä kuin saman asian tekevät C-ohjelmat

Skriptikielten käyttötarkoituksia

Skriptikieliä käytetään muun muassa

- ▶ komentotulkkitasolla mekaanisten tehtävien automatisointiin
- ▶ kuvankäsittelyohjelmissa uusien algoritmien toteuttamiseen ja muun mekaanisen toiminnallisuuden automatisointiin
- ▶ tekstieditoreissa ja kehitysympäristöissä
- ▶ peleissä esim. pelaajahahmojen ja tekoälyn kontrollointiin tai tehtävien logiikan toteuttamiseen
- ▶ muissa isoissa ohjelmissa vastaavasti

Python-kielestä 1/2

- ▶ Julkaistu alunperin 1991, kehitetään edelleen aktiivisesti
- ▶ Kehitetty osin Perl-kielen (sinänsä ansaitun) *write only* -maineen vuoksi
- ▶ Imperatiivinen kieli, tukee oliopohjaisuutta ja poikkeuksia
- ▶ Monella tavalla ”yhtä hyvä” kieli moniin skriptaustehtäviin kuin Perl tai Ruby (makuasia)
- ▶ Siisti, luettava syntaksi
- ▶ Hoitaa itse muistinvarauksen ym. matalan tason putkimieshommat
- ▶ Yksi käytetyimmistä skriptikielistä

Python-kielestä 2/2 – Pythonin filosofiaa

Pythonin tavoitteita:

- ▶ Helposti laajennettava (esim. uusia tyyppejä C:llä)
- ▶ Tukee useita ohjelmointiparadigmoja
- ▶ Pyrkii olemaan sopiva sekä itsenäisten, isojenkin ohjelmien toteuttamiseen että laajennuskieleksi
- ▶ Soveltuu ns. *rapid prototyping* -kieleksi
- ▶ Yksinkertainen kieli, laajat standardikirjastot

Keskeistä kielessä:

- ▶ Dynaaminen tyyppitys (ns. "duck typing": "*If it walks like a duck and quacks like a duck, I would call it a duck.*")

Epämuodollisia rohkaisun sanoja

(Pythonin ja Rubyn kehumista)

Python-tulkki 1/2

- ▶ Python-tulkki käynnistetään komentoriviltä komennolla *python*
- ▶ Skripti voidaan ajaa komentamalla *python skripti.py*
- ▶ Tulkissa voi ajaa suoraan kaikkia Python-komentoja, ja sitä voi käyttää esim. laskimena:

```
1 sliedes@lh:~$ python
2 Python 2.5.2 (r252:60911, Apr 16 2008, 23:58:07)
3 [GCC 4.2.3 (Debian 4.2.3-3)] on linux2
4 Type "help", "copyright", "credits" or "license" for more information.
5 >>> import math
6 >>> math.sin(math.pi/2)
7 1.0
8 >>>
```

Python-tulkki 2/2

Tulkki kertoo virheestä havainnollisesti:

```
1 >>> 1/0
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 ZeroDivisionError: integer division or modulo by zero
5 >>>
```

Kirjoittamalla `help()` pääsee help-tilaan.

Syntaksi

- ▶ Kommenttimerkki on #, kommentit jatkuvat rivin loppuun
- ▶ C:n lohkoja { } vastaava rakenne on sisennys
- ▶ Merkkijonot voi erottaa joko "-merkeillä tai '-merkeillä
- ▶ Useamman rivin merkkijonossa erottimena "" tai ''

```
1 >>> 'foobar'
2 'foobar'
3 >>> "foobar"
4 'foobar'
5 >>> "foo" + 'bar'
6 'foobar'
7 >>> """useampi
8 ... rivi"""
9 'useampi\nrivi'
10 >>>
```

Tietotyyppejä 1/2

- ▶ Listat, `[1, 2, 3]`, ovat olioita, jotka voidaan sijoittaa muuttujiin
- ▶ Sekä merkkijonojen merkkeihin että listan alkioihin voi viitata samalla tavalla:

```
1 >>> a = [1,2,3]
2 >>> a[0]
3 1
4 >>> b = 'abcdef'
5 >>> b[0]
6 'a'
7 >>>
```

- ▶ Lopusta päin voi viitata negatiivisella indeksillä:

```
8 >>> b[-2]
9 'e'
10 >>>
```

Tietotyyppjä 2/2

- ▶ Listoja ja merkkijonoja voi viipaloida (splice):

```
1 >>> b
2 'abcdef'
3 >>> b[1:4] # indeksistä 1 indeksiin 4, indeksiä 4 ei mukaan
4 'bcd'
5 >>> b[2:100] # splicessä indeksit saa mennä yli (tai ali):
6 'cdef'
```

- ▶ Kätevä metodi merkkijonojen pilkkomiseen esim. välilyöntien kohdalta on `split()`:

```
7 >>> 'Monta sanaa lauseessa.'.split(' ')
8 ['Monta', 'sanaa', 'lauseessa.']
```

- ▶ Lisää merkkijonojen ja listojen metodeista: `help(str)` ja `help(list)`

Kokonaisempi esimerkki

► **Fibonaccin luvut:**

```
1 >>> a, b = 0, 1
2 >>> while b < 2000:
3     ...     print b, # Pilkku lopussa: ei rivinvaihtoa
4     ...     a, b = b, a+b # Useampi yhtäaikainen sijoitus
5     ...
6 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

- Ohjelma toimii täysin samanlaisena (samat 4 riviä) myös tiedostossa `fibonacci.py`

if-lause

```
1 x = int(raw_input("Please enter an integer: "))
2 if x < 0:
3     x = 0
4     print 'Negative changed to zero'
5 elif x == 0:    # else if -> elif
6     print 'Zero'
7 else:
8     print 'Positive'
```

for-lause

```
1 # Measure some strings:
2 a = ['cat', 'window', 'defenestrate']
3 for x in a:
4     print x, len(x)
```

Ajettuna:

```
5 cat 3
6 window 6
7 defenestrate 12
```

C-tyylinen for kokonaisluvuille saadaan xrange()- tai range()-
-funktioilla:

```
8 >>> for i in xrange(0,10):
9     ...     print i,
10 ...
11 0 1 2 3 4 5 6 7 8 9
```

break ja continue

break ja continue toimivat yllätyksettömästi samoin kuin C:ssä.

```
1 >>> for n in range(2, 7):
2     ...     for x in range(2, n):
3         ...         if n % x == 0:
4             ...             print n, 'equals', x, '*', n/x
5             ...             break
6     ...
7 4 equals 2 * 2
8 6 equals 2 * 3
```

funktiot, pass

- ▶ Funktio määritellään def-lauseella:

```
1 def square(x):  
2     return x * x
```

- ▶ Monimutkaisemmat parametrit välitetään viittauksina (vrt. Java)
- ▶ Useamman parametrin ottavat funktiot, defaulttiparametrit ym. hienoudet: ks. [tutoriaali 4.7](#)
- ▶ Alilohkot eivät voi Pythonissa olla tyhjiä. Tätä varten on pass-lause, joka ei tee mitään:

```
3 while True:  
4     pass # loputon silmukka  
5  
6 def nop(): # funktio joka ei tee mitään  
7     pass
```

Lisää listoista

- ▶ Listoilla on käteviä metodeja:
 - ▶ `append()` lisää alkion listan loppuun
 - ▶ `insert()` lisää alkion keskelle
 - ▶ `index()` etsii listasta alkioita
 - ▶ `sort()` järjestää listan (in-place, ei palauta järjestettyä listaa (joskus hämäävää))
- ▶ Listoja voi käyttää pinoina käyttämällä `append()`- ja `pop()`-metodeita.
- ▶ Listan a alkioden neliöinti, tai muut temput,
[<lauseke> for <muuttuja> in <lista>]:

```
1 >>> a = [1,2,3,4,5,6]
2 >>> [x*x for x in a]
3 [1, 4, 9, 16, 25, 36]
```

dictionary

Sanakirjatieterakenne *dictionary* on käytännössä hajautustaulu. Sen tunnistaa {}-merkeistä:

```
1 >>> a = {}
2 >>> a['heikki'] = 30
3 >>> a['liisa'] = 20
4 >>> a
5 {'liisa': 20, 'heikki': 30}
6 >>> a['heikki']
7 30
8 >>> a[13] = 'luku'
9 >>> a
10 {'liisa': 20, 13: 'luku', 'heikki': 30}
11 >>> a['heikki']
12 30
```

Merkkijonojen muotoilu

Minkä tahansa olion voi muuttaa merkkijonoksi funktiolla `str()`

- ▶ `str(1) → '1'`
- ▶ `str(1) + '+' + str(1) + '=' + str(1+1) → '1+1=2'`

Vaihtoehtoisesti C-tyylinen syötteen muotoilu %-operaattorilla, joskin kaikille tyypeille voi halutessaan käyttää %s:ää:

- ▶ `'%s+%s=%s' % (1,1,1+1) → '1+1=2'`

Tiedostot

- ▶ Tiedosto avataan `open()`-funktiolla; moodiparametrit ovat samat kuin C:ssä:

```
1 >>> f=open('heippa.txt', 'w')
2 >>> print f
3 <open file 'heippa.txt', mode 'w' at 0x7f7ed94718a0>
4 >>> f.write('Heippa, maailma!\n')
5 >>> f.close()
```

Yksittäisen rivin voi lukea `readline()`-metodilla. Kokonaisen tiedoston voi lukea tiedosto-olion `read()`-metodilla.

Vaihtoehtoisesti tiedoston voi lukea riveittäin merkkijonolistaksi `readlines()`-metodilla.

Tai:

```
7 for line in f:
8     tee_jotain_riville(f)
```

Luokat ja oliot

Luokan voi tehdä `class`-lauseella. Metodien ensimmäinen parametri on konventionaalisesti *self*, joka vastaa Javan *this*-muuttujaa:

```
1 import math
2
3 class Ympyra: # Periyttäminen: class KulmikasYmpyra(Ympyra): ...
4     def init(self, r):
5         self.r = r
6
7     def sade(self):
8         return self.r
9
10    def ala(self):
11        return math.pi * self.r * self.r
```

Poikkeukset

- ▶ Poikkeukset toivottavasti tuttuja Javasta
- ▶ Poikkeuksia voi heittää raise-käskyllä (minkä tahansa olion, mukaanlukien primitiivit, voi heittää)
- ▶ Poikkeusten käsittely try-except -lohkolla:

```
1 def hasardi_operaatio():
2     raise IOError("Kaikki hajos.")
3
4 try:
5     hasardi_operaatio()
6 except e:
7     if isinstance(e,IOError):
8         print "IO-virhe"
9     else:
10        raise # heitä tullut poikkeus uudestaan
```

Mistä lisätietoa?

- ▶ <http://www.python.org/doc/>
- ▶ Kannattaa lukea ainakin standardikirjaston hakemisto:
<http://docs.python.org/lib/lib.html>
- ▶ Tämä tutoriaali laajemmin:
<http://docs.python.org/tut/tut.html>