

Metaohjelmointi Python-kielellä

Mikko Koho

Seminaarityö

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Helsinki, 18. marraskuuta 2014

Tiedekunta — Fakultet — Faculty		Laitos — Institution — Department	
Matemaattis-luonnontieteellinen		Tietojenkäsittelytieteen laitos	
Tekijä — Författare — Author			
Mikko Koho			
Työn nimi — Arbetets titel — Title			
Metaohjelmointi Python-kielellä			
Oppiaine — Läroämne — Subject			
Tietojenkäsittelytiede			
Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages	
Seminaarityö	18. marraskuuta 2014	8	
Tiivistelmä — Referat — Abstract			
<div>Tiivistelmä.</div> <div>ACM Computing Classification System (CCS):</div>			
Avainsanat — Nyckelord — Keywords			
Python, metaohjelmointi			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

Sisältö

1	Johdanto	1
2	Python-ohjelmointikieli	1
2.1	Syntaksi	2
2.2	Muuttujat	2
2.3	Sekvenssit	3
2.4	Luokat ja oliot	4
3	Metaohjelmointi Pythonilla	5
3.1	Introspektio	5
3.2	inspect-moduuli	5
3.3	dis-moduuli	6
3.4	Käännösaikainen metaohjelmointi	6
4	Yhteenveto	7
	Lähteet	8

1 Johdanto

Metaohjelmoinnilla tarkoitetaan klassisen määritelmän mukaan sellaisen tietokoneohjelman tekemistä, joka kirjoittaa uusia tietokoneohjelmia [HB12]. Tämä on kuitenkin melko yksinkertaistettu määritelmä, eikä metaohjelmointia ole helppo määritellä tarkasti. Toinen yleinen määritelmä esittää metaohjelmoinnin olevan ”tietokoneohjelma, joka manipuloi toisia ohjelmia ajon aikana” [HB12].

Tässä seminaarityössä tarkastellaan Python-ohjelmointikielen tarjoamia työkaluja metaohjelmointiin. Alussa käydään läpi Python-kielen perusteita ja tämän jälkeen tutustutaan metaohjelmointiin Python-kielellä. Metaohjelmoinnista tarkastellaan lähinnä suoritusajasta metaohjelmointia.

2 Python-ohjelmointikieli

Ensimmäinen Python-kielen versio on julkaistu 1991. Python-kielestä on nykyään käytössä eri versioita ja Python 2.7 on edelleen melko suosittu vaikka versio 3 on julkaistu jo 2008. Versio 3 ei ole yhteensopiva aiempien versioiden kanssa. Version 2:n suosion taustalla on se, että monet suositut kirjastot ja sovelluskehikset eivät ole siirtyneet versioon 3. Tämän tekstin esimerkit toimivat sekä Python 2.7:llä että Python 3:lla, ellei toisin mainita.

Pythonin suosio on kasvanut tasaisesti ja se on nykyään käytetyin kieli ohjelmoinnin perusteiden opetukseen Yhdysvaltojen yliopistoissa [Guo14].

Python-ohjelmakoodia voidaan kääntää useilla eri kääntäjillä [Mar06]. Käytetyin kääntäjä on CPython (Classic Python), joka kääntää alkuperäisen koodin Python-tavukoodiksi. Tavukoodia ajetaan C-kielellä toteutetulla virtuaalikoneella [Mar06]. Standardikirjasto on toteutettu osittain C:llä ja osittain Pythonilla.

Muita suosittuja kääntäjiä ovat Java-tavukoodiksi kääntävä Jython sekä IronPython, joka kääntää Python-koodia .NET-ympäristön käyttämäksi CIL-tavukoodiksi. PyPy on Python-kielellä toteutettu useissa eri ympäristöissä toimiva suoraan konekielelle koodia kääntävä ajonaikainen (just-in-time) kääntäjä. PyPy on toteutettu RPython-kielellä, joka on Python-kielen osajoukko.

```
def faculty(x):  
    if x > 1:  
        return x * faculty(x - 1)  
    else:  
        return 1
```

Listing 1: Python code

Tässä luvussa käydään läpi Python-kielen perusteita ja metaohjelmoinnin kannalta olennaisia asioita.

2.1 Syntaksi

Python ohjelma koostuu loogisista riveistä, jotka ovat yhden tai useamman ”fyysisen” rivin mittaisia. [Mar06]. Loogisten rivien päättämiseen ei käytetä mitään merkkiä. Rivien sisennyksen perusteella erotetaan ohjelmakoodin lohkot toisistaan. Suositeltu tapa sisentää on käyttää ensimmäisen tason sisentämiseen 4 välilyöntiä ja seuraavaan 8 ja niin edelleen [VWC13].

Pythonissa on 30 avainsanaa (keyword), jotka ovat kielen varattuja sanoja. Näitä ovat esimerkiksi funktio `print` ja kielen rakenteissa käytetyt sanat kuten `if`, `and` ja `class`. Pythonin standardikirjasto koostuu sisäänrakennettujen funktioiden lisäksi kokoelmasta eri tarkoituksiin soveltuvia moduuleita (module), jotka pitää tarvittaessa tuoda erikseen osaksi suoritettavaa ohjelmaa komennolla `import`.

2.2 Muuttujat

Python on dynaamisesti tyyppitetty kieli eli muuttujien arvon tyyppiä ei tarvitse eksplisiittisesti määrittää vaan tyyppi määräytyy sen perusteella minkälainen arvo muuttujaan sijoitetaan. Muuttujan tyyppiä voi myös vaihtaa sijoittamalla siihen uuden eri tyyppisen arvon. Listaus 1 sisältää yksinkertaisen esimerkin Python-kielen syntaksista.

Modernit IDE:t kuten PyCharm¹ pystyvät koodin perusteella usein päättelemään muuttujan tyyppin.

¹<https://www.jetbrains.com/pycharm/>

```

1  # -*- coding: utf-8 -*-
2
3  import sys
4
5  a = 'Hello world!'
6  print(a)
7  # Hello world!
8
9  a = len(a)
10
11 print(a)
12 # 12

```

Listaus 1: Yksinkertainen esimerkki Python-kielen syntaksista.

Pythonissa kaikki arvot, muuttujat ja funktiot ovat olioita. Olion tyyppi määrittää mitä metodeja ja ominaisuuksia olio tarjoaa. Osa olioista on muuttumattomia (immutable) ja osa muutettavia (mutable). Python-kielessä ei ole erikseen vakioita, mutta käytäntönä on käyttää muuttujan nimessä pelkästään isoja kirjaimia, jos muuttujan arvoa ei ole tarkoitus muuttaa.

2.3 Sekvenssit

Python-kielessä mm. sisäänrakennetut tietotyypit lista, merkkijono ja monikko (tuple) ovat sekvenssejä. Sekvenssit ovat iteroitavia (iterable) olioita eli ne kykenevät palauttamaan jäseniään yksi kerrallaan. Iteroitavia olioita ovat myös muut oliot, joissa on toteutettu jäseniä palauttava `__iter__`-metodi. Iteroitavia olioita voidaan käyttää suoraan osana esimerkiksi `for`-toistolauseissa. kuten esimerkiksi lauseessa `for x in [1,2,3]: print(x)`.

Listakehitelmä (list comprehension), joukkokehiteelmä (set comprehension) ja sanakirjakehitelmä (dictionary comprehension) ovat helppoja tapoja luoda lista-, joukko- tai sanakirjaolioita jonkin syötteen perusteella. Esimerkki listakehitelmän sekä joidenkin Python-kielen funktioiden käytöstä on listauksessa 2. Funktio `range` palauttaa Python 2:ssa listan ja Python 3:ssa generaattoriolion, jota voidaan käyttää listan tapaan. Funktio `all` tarkastaa

kaikkien listan (tai muun iteroitavan olion) totuusarvon. Pythonissa kaikki oliot voidaan evaluoida totuusarvoina, jolloin lukujen tapauksessa aina luku "0" evaluoituu epätodeksi ja muut luvut todeksi.

```
1  print(range(10))
2  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3
4  print(range(0, 30, 3))
5  # [0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
6
7  print( [x**2 for x in range(11)] )
8  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
9
10 some_list = [1, 4, 7, 'foo', 12, 'bar']
11 print( [x for x in some_list if str(x).isalpha()] )
12 # ['foo', 'bar']
13
14 # Alkulukuja
15 print(
16     [x for x in range(2, 50) if all( [x % y for y in range(2, x-1)] )]
17 )
18 # [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Listaus 2: Esimerkki funktion range käytöstä ja listakehitelmistä.

2.4 Luokat ja oliot

Python on olio-ohjelmointikieli, joka tukee moniperintää.

Kahdella alaviivalla nimen alussa ja lopussa merkitään Python-kielen "maagisia" metodeita, attribuutteja ja olioita.

3 Metaohjelmointi Pythonilla

3.1 Introspektio

Introspektiolla tarkoitetaan tietojen hakemista muistissa olevista olioista, moduuleista ja funktioista [Pil04].

Funktio `type` palauttaa olion tietotyypin [Pil04]. Tämä on yleensä luokka, mutta vanhan tyyliä Python 2 -luokilla tämä on merkkijono ”instance”. Varsinainen luokka löytyy aina olion attribuutista `__class__`, mutta Python 3:ssa `type:n` käyttö on suositus.

Funktio `dir` palauttaa listan olion attribuuteista, joihin kuuluu myös olion metodit. Funktio `getattr` ottaa parametrina merkkijonon ja palauttaa parametrin nimisen attribuutin. Tällöin voidaan esimerkiksi kutsua metodeja oliosta, jonka rakennetta ei tunnetta vielä käännösvaiheessa [Pil04]. Funktiolla `isinstance` voidaan tarkistaa onko joku olio tietyn tyyppinen.

Olioiden lyhyet kuvaukset (docstring) saa ajon aikana haettua niiden `__doc__`-attribuutista.

Olioita, luokkia ja funktioita voidaan muokata ajon aikana melko vapaasti. Esimerkki peruskirjaston `dir` funktion ylikirjoittamisesta omalla funktiolla on listauksessa 3. Esimerkin oma funktio palauttaa sille annetut argumentit tekemättä niille mitään. Tämänkaltaisesta kirjastojen ja moduuleiden osien ajonaikaisesta muokkaamisesta käytetään myös tsesta käytetään myös termiä ”monkey patching”.

3.2 inspect-moduuli

Pythonin standardikirjaston inspect-moduuli tarjoaa työkaluja introspektioon ohjelman ajon aikana.

Pythonissa on sisäänrakennetut funktiot `compile`, `eval` ja `exec`, joiden avulla voidaan kääntää ja ajaa Python-koodia ohjelman ajon aikana [Cro14, Mar06]. Esimerkki tällaisesta on listauksessa 4. Esimerkin rivillä 1 luodaan muuttuja, joka sisältää käännettävän koodin merkkijonona. Rivillä 2 käännetään koodi merkkijonosta tavukoodiksi käyttämällä funktiota `compile`, jonka parametrit ovat käännettävä koodi, koodin sisältävän tiedoston nimi, jossa käytetään merkkijonoa ’<string>’ tarkoittamaan, että


```

1  # -*- coding: utf-8 -*-
2
3  import __builtin__
4
5  def x(*args):
6      return args
7
8  print(x(1, 5, 'kissa'))
9  # (1, 5, 'kissa')
10
11 print(dir(__builtin__))
12 # ['ArithmeticError', 'AssertionError', 'AttributeError', ... ]
13
14 __builtin__.dir = x
15
16 print(dir(__builtin__))
17 # (<module '.__builtin__' (built-in)>,)

```

Listaus 3: Standardikirjaston funktion ylikirjoittaminen omalla funktiolla.

koodia ei luettu tiedostosta ja käännöstila (mode), jossa 'single' kertoo, että käännetään yksi lause (statement). Rivi 4 tulostaa `code_obj` -muuttujan merkkijonoesityksen. Rivillä 7 ajetaan tavukoodi muuttujasta `code_obj`.

3.3 dis-moduuli

3.4 Käännösaikainen metaohjelmointi

Käännösaikaista metaohjelmointia ei ole suoraan tuettu Pythonissa, mutta tämä ominaisuus on lisätty ainakin kahteen Pythonista jatkokehitettyyn kehitettyyn kieleen, Mythoniin[Rie08] ja Convergeen[Tra05]. Mython kääntää ohjelmakoodia suoraan Python-tavukoodiksi, mutta Convergen tuottamaa koodia ajetaan kielen omalla virtuaalikoneella.

```
1  code_str = 'print("Hello world!")'
2  code_obj = compile(code_str, '<string>', 'single')
3
4  print(code_obj)
5  # <code object <module> at 0x7f06341f2cb0, file "<string>", line 1>
6
7  exec(code_obj)
8  # Hello world!
```

Listaus 4: Esimerkki Python-komennon kääntämisestä tavukoodiksi ohjelman ajon aikana ja käännetyin koodin ajamisesta [Cro14].

4 Yhteenveto

Yhteenveto.

Lähteet

- [Cro14] Crosta, D.: *Exploring Python Code Objects*, 2014. <http://late.am/post/2012/03/26/exploring-python-code-objects> [17.11.2014].
- [Guo14] Guo, P.: *Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities*. Communications of The ACM Blog, heinäkuu 2014. <http://cacm.acm.org/blogs/blog-cacm/176450/fulltext> [06.11.2014].
- [HB12] Hazzard, K. ja Bock, J.: *Metaprogramming in. NET*. Manning Publications, 2012.
- [Mar06] Martelli, A.: *Python in a Nutshell*. O'Reilly Media, Inc., 2006.
- [Pil04] Pilgrim, M.: *Dive Into Python*, toukokuu 2004. <http://www.diveintopython.net/> [06.11.2014].
- [Rie08] Riehl, J.: *The Mython Programming Language*, 2008. <http://mython.org/> [16.11.2014].
- [Tra05] Tratt, L.: *Compile-time meta-programming in a dynamically typed OO language*. Teoksessa *Proceedings Dynamic Languages Symposium*, sivut 49–64, October 2005.
- [VWC13] Van Rossum, G., Warsaw, B. ja Coghlan, N.: *PEP 8 – Style guide for python code*. 2013. <http://www.python.org/dev/peps/pep-0008> [11.11.2014].