Metaohjelmointi Python-kielellä

Mikko Koho

Seminaarityö
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 20. marraskuuta 2014

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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	20. marraskuuta 2014	9

Tiivistelmä — Referat — Abstract

Tiivistelmä.

ACM Computing Classification System (CCS):

Avainsanat — Nyckelord — Keywords
Python, metaohjelmointi

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

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].

Tärkeä osa metaohjelmointia on ohjelman ajonaikainen tilansa tarkastelu ja muuntelu eli reflektio. Reflektioon kuuluu käsite introspektio, jolla tarkoitetaan ajonaikaista muistissa olevien olioiden tarkastelua [Pil04].

Tässä seminaarityössä tarkastellaan Python-ohjelmointikielen tarjoamia työkaluja metaohjelmointiin. Alussa käydään läpi Python-kielen perusteita ja tämän jälkeen tutustutaan metaohjelmointiin Python-kielellä. Metaohjelmoinnista tarkastellaan lähinnä suoritusaikaista 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 sovelluskehykset eivät ole siirtyneet versioon 3. Tämän seminaarityön esimerkit toimivat sekä Python 2.7:llä että Python 3:lla, ellei toisin mainta. Esimerkeissä olevat rivin alusta alkavat kommentit sisältävät ohjelman tulosteita, koodia koskevat kommentit on laitettu kommentoitavan rivin perään.

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. Muita suosittuja kääntäjiä ovat Java-tavukoodiksi kääntävä Jython sekä IronPython, joka kääntää Python-koodia .NET-ympäristön käyttämäksi CIL-tavukoodiksi. PyPy on Python-kielellä to-

teutettu useissa eri ympäristöissä toimiva suoraan konekielelle koodia kääntävä ajonaikainen (just-in-time) kääntäjä. PyPy on toteutettu RPython-kielellä, joka on Python-kielen osajoukko.

Pythonin standardikirjasto on toteutettu osittain C:llä ja osittain Pythonilla.

CPythonilla käännettyä tavukoodia voidaan ajaa C-kielellä toteutetulla virtuaalikoneella [Mar06]. Virtuaalikone tulkitsee ajettaessa tavukoodia lause kerrallaan.

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 tyypitetty 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. Rivillä 1 asetetaan muuttujan `a` arvoksi merkkijono `Hello world!`, joka tulostetaan rivillä 2. Rivillä 5 tulostetaan merkkijonon `a` pituus.

Modernit IDE:t kuten PyCharm[1] pystyvät koodin perusteella usein päättelemään muuttujan tyypin.

```
1 a = 'Hello world!'
2 print(a)
3 # Hello world!
4
5 print(len(a))
6 # 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.

Pythonin sisäänrakennettuja tietotyyppejä on mm. numeeriset `int` ja `float`, sekvenssityypit `list`, `str` ja `tuple`, joukko `set`, ”sanakirja” `dict` sekä tiedosto `file`. Näiden lisäksi standardikirjaston moduuleista löytyy lisää tietotyyppejä kuten `datetime` ja `array`.

2.3 Sekvenssit

Sisäänrakennetuista tietotyypeistä mm. 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), joukkokehitelmä (set comprehension) ja sanakirjakehitelmä (dictionary comprehension) ovat helppoja tapoja luoda lista-, joukko- tai sanakirjaolioita jonkin syötteen perusteella. Esimerkkejä listakehitelmän käytöstä sekä joidenkin Python-kielen funktioiden käytöstä

[1] `https://www.jetbrains.com/pycharm/`

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.

3 Metaohjelmointi Pythonilla

Metaohjelmointi on Pythonilla hyvin luontevaa, koska olioita voidaan yleensä muokata täysin vapaasti ajon aikana ja niistä saadaan paljon metatietoa ulos Pythonin peruskirjaston työkaluilla.

3.1 Reflektio

Funktio `type` palauttaa olion tietotyypin [Pil04]. Tämä on yleensä luokka, mutta vanhan tyylisillä 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. Kahdella alaviivalla nimen alussa ja lopussa merkitään Python-kielen ”maagisia” metodeita, attribuutteja ja olioita.

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 olioiden tilan tutkimiseen ohjelman ajon aikana.

3.3 Dynaamiset luokat

Pythonilla on mahdollista luoda uusia luokkia dynaamisesti ohjelman ajon aikana `type`-funktiolla.

```
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.

3.4 Metaluokat

Pythonissa on sisäänrakennetut funktiot `compile`, `eval` ja `exec`, joiden avulla voidaan kääntää ja ajaa Python-koodia ohjelman ajon aikana [Cro14, Mar06].

Esimerkki ajonaikaisesta koodin kääntämisestä 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 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`.

```
code_str = 'print("Hello world!")'
code_obj = compile(code_str, '<string>', 'single')

print(code_obj)
# <code object <module> at 0x7f06341f2cb0, file "<string>", line 1>

exec(code_obj)
# Hello world!
```

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

3.5 Tavukoodin tarkastelu ja muokkaus

Standardikirjaston `dis`-moduulilla voidaan tutkia Python-tavukoodia. Listauksessa 5 eräs esimerkkifunktio sekä sen tavukoodin muodostaminen `dis`-moduulin `dis`-metodilla sekä tulostettu tavukoodi. Tavukoodi on tulostettu Python 2:lla ja se on erinäköinen Python 3:lla.

Moduuli `parser` antaa rajapinnan Python-kääntäjän sisäiseen jäsennyspuuhun ja mahdollistaa sen muokkaamisen. Tämän moduulin lisäämisen jälkeen on kuitenkin standardikirjastoon lisätty `ast`-moduuli, joka mahdollistaa kääntäjän abstraktin syntaksipuun (abstract syntax tree) luomisen annetun ohjelmakoodin perusteella ja sen muokkaamisen. Funktiolle `compile` voidaan antaa parametrina muokatun AST-puun sisältävä `AST`-olio ja kääntää se tavukoodiksi.

3.6 Käännösaikainen metaohjelmointi

Käännösaikaista metaohjelmointia ei ole suoraan tuettu Pythonin standardikirjastossa. Template-metaohjelmointi on mahdollista esimerkiksi Jinja2 -template-moottorilla (template engine) [Ron14].

Käännösaikainen metaohjelmointi on myös lisätty suoraan osaksi kahta Pythonista jatkokehitettyä kieltä, Mythoniin[Rie08] ja Convergeen[Tra05]. Mython kääntää ohjelmakoodia suoraan Python-tavukoodiksi, mutta Convergen tuottamaa koodia ajetaan kielen omalla virtuaalikoneella.

```
1  # -*- coding: utf-8 -*-
2
3  import dis
4
5  def neliot(iteroitava):
6    '''Palauta lista parametrin alkioiden neliöistä'''
7    return [x**2 for x in iteroitava]
8
9  dis.dis(neliot)
10 #  4           0 BUILD_LIST               0
11 #              3 LOAD_FAST                0 (iteroitava)
12 #              6 GET_ITER
13 #        >>    7 FOR_ITER                16 (to 26)
14 #             10 STORE_FAST               1 (x)
15 #             13 LOAD_FAST                1 (x)
16 #             16 LOAD_CONST               1 (2)
17 #             19 BINARY_POWER
18 #             20 LIST_APPEND              2
19 #             23 JUMP_ABSOLUTE            7
20 #        >>   26 RETURN_VALUE
```

Listaus 5: Python-tavukoodin tarkastelu dis-moduulilla.

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].

[Ron14] Ronacher, A.: *Jinja2 (The Python Template Engine)*, 2014. `http://jinja.pocoo.org/` [20.11.2014].

[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].