Metaohjelmointi Python-kielellä

Mikko Koho

Seminaarityö
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 17. 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	17. marraskuuta 2014	7

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 **1**

2 Python-ohjelmointikieli **1**

2.1 Pythonin syntaksi 2

2.2 Operaattorit 2

2.3 Muuttujat 2

2.4 Luokat ja oliot 2

2.5 Muuttujatyyppejä 3

2.6 Iteroitavat 3

3 Python-kielen metaohjelmointimaisia komponentteja **4**

3.1 Introspektio 4

3.2 Standardikirjaston inspect-moduuli 6

3.3 Käännösaikainen metaohjelmointi 6

4 Yhteenveto **6**

Lähteet **7**

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 suoritusaikaista metaohjelmointia.

2 Python-ohjelmointikieli

Ensimmäinen Python-kielen versio julkaistu 1991 [TODO: lähde]. 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 tekstin esimerkit toimivat sekä Python 2.7:llä että Python 3:lla, ellei toisin mainta.

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.

Tässä luvussa käydään läpi Python-kielen perusteet.

2.1 Pythonin 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 Operaattorit

2.3 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 2.3 sisältää yksinkertaisen esimerkin Python-kielen syntaksista.

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

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).

2.4 Luokat ja oliot

Python on olio-ohjelmointikieli, joka tukee moniperintää.

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

```
# -*- coding: utf-8 -*-

import sys

a = 'Hello world!'
print(a)
# Hello world!

a = len(a)

print(a)
# 12
```

Listing 1: Yksinkertainen esimerkki Python-kielen syntaksista.

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

2.5 Muuttujatyyppejä

2.6 Iteroitavat

Listakehitelmä (list comprehension), joukkokehitelmä (set comprehension) ja sanakirjakehitelmä (dictionary comprehension) ovat tapoja luoda lista-, joukko- tai sanakirjaolioita.

Esimerkki listakehitelmän käytöstä sekä joistain Python-kielen funktioista on listauksessa 2.6. 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.

```
print(range(10))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(range(0, 30, 3))
# [0, 3, 6, 9, 12, 15, 18, 21, 24, 27]

print( [x**2 for x in range(11)] )
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

some_list = [1, 4, 7, 'foo', 12, 'bar']
print( [x for x in some_list if str(x).isalpha()] )
# ['foo', 'bar']

# Alkulukuja
print(
  [x for x in range(2, 50) if all( [x % y for y in range(2, x-1)] )]
)
# [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Listing 2: Esimerkki funktion range käytöstä ja listakehitelmistä.

3 Python-kielen metaohjelmointimaisia komponentteja

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 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 meto-

deja 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 3.1. 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”.

```
# -*- coding: utf-8 -*-

import __builtin__

def x(*args):
  return args

print(x(1, 5, 'kissa'))
# (1, 5, 'kissa')

print(dir(__builtin__))
# ['ArithmeticError', 'AssertionError', 'AttributeError', ... ]

__builtin__.dir = x

print(dir(__builtin__))
# (<module '__builtin__' (built-in)>,)
```

Listing 3: Standardikirjaston funktion ylikirjoittaminen omalla funktiolla.

3.2 Standardikirjaston inspect-moduuli

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

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

```
code_str = 'print "Hello world!"'

code_obj = compile(code_str, '<string>', 'single')

exec(code_obj)
```

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

3.3 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.

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].