

Python

Seminaarityö

Kristian Ovaska

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos
Ohjelmointikielten periaatteet 2004
Helsinki 28. maaliskuuta 2004

Sisältö

1	Yleiskatsaus	1
1.1	Historia	1
1.2	Kielen toteutuksia	2
1.3	Kehitysprosessi	2
2	Tietoabstraktiot	2
2.1	Oliot	3
2.2	Moduulit	4
3	Tyypitys	4
3.1	Alkeistyytit	5
3.2	Yhdistelmätyypit	5
3.2.1	Listakomprehensiot	6
4	Nimien hallinta	6
5	Ohjausrakenteet	7
5.1	Funktiot	7
5.2	Iteraatio ja generaattorit	8
5.3	Poikkeukset	8

1 Yleiskatsaus

Python [Ros03a] on oliopohjainen skriptikieli, joka sopii myös laajempien ohjelmistojen toteuttamiseen. Kielen piirteitä ovat dynaaminen tyyppitys, pelkistetty syntaksi ja monipuoliset sisäänrakennetut tietotyypit sekä standardikirjasto. Python on moniparadigmakieli: oliopohjaisuuden lisäksi se sopii proseduraaliseen ja funktionaaliseen ohjelmointiin. Dynaamisen luonteensa vuoksi Python sopii parhaiten tulkittavaksi kieleksi; konekoodia tuottavaa Python-kääntäjää ei toistaiseksi ole olemassa.

Kirjoituksessa käydään läpi Pythonin ominaisuuksia ja historiaa. Joidenkin ominaisuuksien kohdalla mainitaan, mistä kielestä Python on saanut vaikutteita. Tällöin tyypillisesti rajataan tarkastelu välittömiin vaikuttajiin eikä seurata historiaa pidemmälle.

Merkeillä `>>>` alkavat koodiesimerkit kuvaavat suoritusta interaktiivisessa tulkissa.

1.1 Historia

Pythonin kehitti hollantilainen Guido van Rossum 80-luvun lopussa ja 90-luvun alussa [Ven03].

Kielen edeltäjä oli opetuskäyttöön suunniteltu kieli ABC [Mee84], jonka parissa Van Rossum työskenteli 1983-86/87. ABC-kielen työnimi oli B, joka sittemmin muutettiin ABC:ksi [Mee92, s. 2]; kyseinen B on eri kieli kuin Ken Thompsonin B-kieli, joka oli C-kielen taustalla [Ker88, s. 1].

ABC:stä Van Rossum otti Pythoniin hyvänä pitämiään ominaisuuksia, kuten sisennyksen käyttäminen ohjelmalohkojen merkitään [Kuc98], tärkeimmät tietorakenteet (listat ja hajautustaulukot), interaktiivinen tulkki ja for-silmukan toiminta.

80-luvun lopussa Van Rossum työskenteli Amoeba-nimisen hajautetun käyttöjärjestelmän parissa ja kehitti Pythonia alunperin sen skriptikieleksi. Kie-

len toteuttaminen alkoi loppuvuodesta 1989 [FAQ]. Ensimmäinen yleiseen käyttöön tarkoitettu versio 0.9 julkaistiin helmikuussa 1991 [Ros91].

Python 1.0 julkaistiin vuonna 1994 ja Python 2.0 vuonna 2000. Tämä kirjoitus käsittelee versiota 2.3, joka on toistaiseksi kielen uusin versio.

1.2 Kielen toteutuksia

Pythonin alkuperäinen ja de-facto -toteutus on CPython [Pyt], jota kutsutaan usein vain Pythoniksi. CPython on C-kielillä toteutettu tavukooditulkki.

Jython on Java-kielinen toteutus, joka kääntää Python-koodia Javan tavukoodiksi ja mahdollistaa Python-ohjelmien ajamisen Javan virtuaalikoneessa.

Psyco on laajennusmoduuli CPythoniin, joka kääntää CPythonin tavukoodia ajonaikaisesti x86-konekoodiksi (just-in-time -käännös).

PyPy [PyPy] (Python in Python) on tutkimushanke, joka tähtää Python-toteutuksen luomiseen kielellä itsellään. Tavoitteena on CPythonia tehokkaampi Python-koodin suorittaminen.

1.3 Kehitysprosessi

Pythonia kehitetään avoimen lähdekoodin projektina. Guido van Rossum päättää viime kädessä kaikesta (C)Pythoniin liittyvästä, mutta kuka tahansa voi osallistua kehitystyöhön. Merkittävät muutosehdotukset vaativat yleensä Python Enhancement Proposal -dokumentin [PEP1], joka avulla Van Rossum ja muu ydinryhmä arvioi ehdotusta.

2 Tietoabstraktiot

Tässä luvussa käsitellään kahta kapselointimenetelmää: olioita ja moduuleita.

2.1 Oliot

Olio on keskeinen rakenne Pythonissa [Ros03a, k. 3.1]. Kaikki suoritusaikaiset rakenteet, kuten luokat, luokkien ilmentymät, funktiot ja lukuarvot ovat olioita. Kaikilla olioilla ei kuitenkaan ole luokkaa, vaan olio saattaa olla jonkin sisäänrakennetun tyypin ilmentymä. Tyypin ja luokan raja on häilyvä: monet tyypit käyttäytyvät kuin luokat ja niitä voi periä omiin luokkiin. Esimerkiksi kokonaislukutyyppejä voi käyttää ”kantatyyppinä”, mutta funktiotyyppi ei tätä kuitenkaan salli.

Luokkamäärittelyissä metodien ja attribuuttien nimet noudattavat seuraavia konventioita: `_x` vastaa jossain määrin `protected`-määrettä, `__x` `private`-määrettä ja `__x__` merkitsee erikoismetodia, joka kuuluu kielen määrittelyyn [Ros03a, k. 3.3]; esimerkiksi `__init__` on luokan alustaja.

Python ei ole ”puhdas” oliokieli. Funktiot voi sijoittaa moduulin päätasolle ja käyttää kieltä proseduraaliseen tapaan. Standardikirjastossa [Ros03b] on paljon tällaisia moduuleita. Luokkien attribuutit ovat oletuksena julkisia, ja yksityisten attribuuttien (ja metodien) käyttäminen on vaivalloista, koska on joka kerta kirjoitettava nimeen kuuluvat alaviivat (`__x`). ”Protected-määre” on pelkkä nimeämiskonventio eikä kuulu varsinaisesti kieleen.

Pienenä erikoisuutena metodien formaalissa parametrilistassa on ensimmäisenä parametrina ilmentymä, jonka kautta metodia kutsutaan:

```
class Luokka:
    def metodi(self, x): pass
```

Tämä on Van Rossumin mukaan [Kuc98] peräisin Modula-2/3:sta, jossa proseduuri saa usein parametriksi moduulin hallinnoiman tyypin arvon [Sco00, s. 126]. Pythonissa `ilmentyma.metodi(x)` ja `Luokka.metodi(ilmentyma, x)` tuottavat saman kutsun.

2.2 Moduulit

Olioiden ohella keskeinen rakenne on moduuli. Python-kielinen moduuli voi sisältää luokkien ja funktioiden määrittelyitä sekä sellaisenaan suoritettavaa koodia, ja C-kielellä toteutettu moduuli mahdollistaa kielen yhdenmukaisen laajentamisen [Ros03c].

Moduulijärjestelmä on saanut vaikutteita Modula-2/3:sta. Esimerkiksi import-lause on hyvin samanlainen [Car89, s. 33].

3 Tyypitys

Python on dynaamisesti ja melko vahvasti tyypitetty. Nimensidontamalli (kts. luku 4) sallii mielivaltaisen olion sitomisen nimeen ajoaikana, joten staattista tarkistusta ei voida tehdä. Ajoaikana tehdään tyyppitarkistuksia: esimerkiksi `1+'2'`, `5[0]` ja `x=5; x()` aiheuttavat poikkeuksen.

Funktio- ja metodikutsuissa sovelletaan rakenne-ekvivalenssia: kutsu `obj.met(5)` onnistuu, jos oliolla `obj` on metodi `met`, jota voi kutsua yhdellä argumentilla (jos argumentti on väärän tyyppinen, metodin suoritus voi tietysti aiheuttaa poikkeuksen). Periaatteessa esim. funktion argumenttina saatu `obj` voi olla vääränlainen olio, jolla vain sattuu olemaan metodi `met`.

Tyypit jaetaan muokattaviin (mutable) ja muokkaamattomiin (immutable). Muokkaamattomien tyyppien arvoa ei voi muuttaa olion luomisen jälkeen; näin ne sopivat hajautustaulun avaimiksi (kts. kappale 3.2).

Seuraavissa kappaleissa käsitellään yleisimpiä sisäänrakennettuja tyypejä [Ros03a, k. 3.2] [Ros03b, k. 2.3], joita ohjelmoija käyttää suoraan. Pythonissa kaikilla ajonaikaisilla olioilla on tyyppi, joten on myös moduulityyppi, funktiotyyppi ym. Niitä ei käsitellä.

3.1 Alkeistyytit

Lukutyyppijä ovat kiinteän- (int) ja mielivaltaisen (long) mittaiset kokonaisluvut, liukuluvut ja kompleksiluvut. Totuusarvo (bool) on teknisesti int-tyytin alityyppi ja sitä käytetään totuusarvovakioiden (True ja False) ilmaisemiseen. bool-tyyppi on varsin uusi ominaisuus kielessä; ennen totuusarvovakiot ilmaistiin C-kielen kaltaisesti [Ker88, s. 42] kokonaisluvuilla.

NoneType-tyyppi ja sen singleton-ilmentymä None vastaa muiden kielten null- tai nil-arvoa.

Kaikki alkeistyytit ovat muokkaamattomia.

3.2 Yhdistelmätytit

Yhdistelmätytit voidaan jakaa kuvaus- (map) ja sekvenssityyppeihin (sequence).

Ainoa kuvaustyyppi on hajautustaulukko (dict[ionary]). Se kuvaa avainolion mielivaltaiseksi arvo-olioksi. Avaimen on oltava muokkaamaton, koska avain etsitään hajautusarvon (hash) avulla, joka ei saa muuttua avainolion elinkaaren aikana. Hajautustaulukko on sisäisesti tärkeä tyyppi, koska nimiavaruudet toteutetaan sen avulla (kts. luku 4).

Sekvenssityyppejä ovat listat (list), monikot (tuple) ja merkkijonot (str ja unicode). Listat ja monikot sisältävät järjestetyn kokoelman olioita. Listat on toteutettu muuttuvanmittaisina taulukoina, ei siis Lispin [McC60] kaltaisina linkitettyinä listoina. Monikot ovat muuten listojen kaltaisia, mutta muokkaamattomia. Tämän vuoksi ne sopivat hajautustaulun avaimiksi, tosin vain, jos myös kaikki monikon alkiot ovat muokkaamattomia.

Sekvensseistä voi luoda uusia sekvenssejä viipaloimalla (slicing). Syntaksi on saanut vaikutteita Algol-68:sta [Tan76, s. 170], jossa sitä kutsutaan termillä trimming, sekä Iconista [Gri96, s. 56]. Esimerkki viipaloinnista:

```
>>> lista = [0, 1, 2, 3, 4]
>>> lista[1:3]
[1, 2]
```

3.2.1 Listakomprehensiot

Yksi tapa tuottaa listoja on listakomprehensiot (list comprehensions) [PEP202], jotka tuottavat listan alkiot for-silmukassa toistettavan lausekkeen ja valinnaisen if-lauseen avulla:

```
>>> [x**2 for x in [0, 1, 2, 3, 4] if x>=2]
[4, 9, 16]
```

Piirre on periytynyt Haskellista [Jon03, k. 3.11], jossa em. lista generoidaan lausekkeella `[x**2 | x <- [0, 1, 2, 3, 4], x>=2]`, ja piirteen juuret ovat matemaattisessa joukkomerkinä $\{x^2 \mid \forall x \in [0, 4] : x \geq 2\}$.

4 Nimien hallinta

Yhtenäisen oliomallin vuoksi on luontevaa, että muuttujat noudattavat Pythonissa viitesemantiikkaa: muuttuja on nimi, jotka viittaa (eli on sidottu) johonkin olioön [Ros03a, k. 4.1]. Nimen voi sitoa mihin tahansa olioön, joten nimillä ei ole tyyppiä, vain olioilla on.

Nimiavaruus (namespace) on Pythonissa rakenne, joka sisältää nimisidonnat. Moduuleilla, luokilla, ilmentymillä ja funktiokutsuilla on ajoaikana omat nimiavaruutensa, jotka on sisäisesti toteutettu hajastustaulukkoina. Nimen etsintä etenee hierarkiassa ylöspäin, esimerkiksi funktiosta moduuliin ja sisäänrakennettuun nimiavaruuteen.

Hieman epäortogonaalisena piirteenä metodista ei näe suoraan luokan nimiavaruuteen, vaan viittaukset metodeihin ja staattisiin luokkamuuttujiin

on tehtävä self-parametrin kautta. Ilmentymämuuttujiin viitataan samalla tavalla.

Funktiokutsujen yhteydessä sovelletaan leksikaalista skooppia. Luokalla ja ilmentymällä on eri nimiavaruudet. Luokan nimiavaruus sisältää metodit ja luokkamuuttujat, ilmentymän nimiavaruus ilmentymämuuttujat.

Nimiavaruus muuttuu olemassaoloaikanaan, kuten seuraavasta käy ilmi (`locals` on funktio, joka löytyy sisäänrakennetusta nimiavaruudesta):

```
def f(a, b):
    print locals()          # Tulostaa {'a': 1, 'b': 2}
    z = 5; print locals()  # Tulostaa {'a': 1, 'b': 2, 'z': 5}
    del b; print locals()  # Tulostaa {'a': 1, 'z': 5}
f(1, 2)
```

5 Ohjausrakenteet

Tässä luvussa käsitellään joitakin Pythonin ohjausrakenteita.

5.1 Funktiot

Funktiot ja metodit määritellään `def`-lauseella. `def` on ajoaikana suoritettava lause samoin kuin `if` tai `while`. `def`-lauseen suorittaminen luo funktio-olion ja sitoo sen ympäröivään nimiavaruuteen. Myös mahdolliset parametrien oletusarvot evaluoidaan tässä vaiheessa. Funktion (tavu)koodi on käännetty etukäteen.

Koska funktiot ovat yhtenäisen oliomallin ansiosta ensiluokan olioita (first-class objects), sopii Python tältä osin funktionaaliseen ohjelmointiin. Myös listakomprehensiot tukevat funktionaalista mallia. Toisaalta Python ei muuta häntärekursiota iteratiivikseksi rakenteeksi, mikä yleensä on funktionaalisten kielten piirre.

Pythonissa on funktionaalisista kielistä tutut lambda-lausekkeet, jotka tuottavat nimettömän funktion, jonka voi suorittaa halutuilla parametreilla. Lambda-rakenne on Pythonissa rajoitettu: se voi sisältää vain lausekkeitä, ei lauseita kuten if- ja for-lauseet. Lambda-avainsana on peräisin lambda-kalkyylistä; ohjelmointikielissä sen esitteli Lisp [McC60, s. 189].

5.2 Iteraatio ja generaattorit

Iteraatio tapahtuu for-lauseella: `for x in kokoelma`. Iteroitavan olion on tuettava Pythonin iteraatioprotokollaa [PEP234], joka mahdollistaa yhtenäisen iteraation erilaisille rakenteille. Sisäänrakennetut sekvenssi- ja kuvaustyytit tukevat protokollaa, samoin tiedostotyyppi `file`, joka mahdollistaa tekstitiedoston rivien iteroimisen. Omiin luokkiin voi rakentaa tuen iteraatiolle.

Yksi keino toteuttaa iteraattori on generaattorifunktio [PEP255], joka muistuttaa tavallista funktiota, mutta `return`-lauseen sijasta käyttää `yield`-lauseetta. Generaattorit ovat saaneet vaikutteita CLU:sta [Lis81, s. 70], joka myös käyttää `yield`-avainsanaa ja samankaltaista `for`-lauseetta iterointiin.

5.3 Poikkeukset

Python käyttää poikkeuksia virheiden hallitsemiseen ja myös iteraatioprotokollassa ilmoittamaan iteraation päättymisen `for`-silmukalle. Standardikirjastossa on poikkeushierarkia, josta sovellusohjelmat voivat periä omia poikkeusluokkia. Poikkeusmekanismi on Van Rossumin mukaan [Kuc98] saanut vaikutteita Modula-2/3:sta. Avainsanat (`raise`, `try..except`, `try..finally`) ovat kielissä samat [Car89, s. 21].

Lähteet

- [Car89] Luca Cardelli, et al. *Modula-3 Report (revised)*. No. 52, Digital Systems Research Center, Palo Alto (CA), 1989.
- [FAQ] *General Python FAQ: "Why was Python created in the first place?"*
<http://www.python.org/doc/faq/general.html> (26.3.2004)
- [Gri96] Ralph E. Griswold, Madge T. Griswold. *The Icon Programming Language, Third Edition*. Peer-to-Peer Communications, 1996.
 [Myös <http://www.cs.arizona.edu/icon/1b3.htm>, 26.3.2004]
- [Jon03] Simon Peyton Jones. *Haskell 98 Language and Libraries, The Revised Report*. Cambridge University Press, 2003.
 [Myös <http://www.haskell.org/onlinereport/>, 26.3.2004]
- [Ker88] Brian W. Kernighan, Dennis M. Ritchie. *The C Programming Language, Second Edition*. Prentice Hall, Inc., 1988.
- [Kuc98] Andrew Kuchling. *LJ Interviews Guido van Rossum*. Linux Journal, Issue 55, November 1998. [Myös
<http://www.linuxjournal.com/article.php?sid=2959>,
 26.3.2004]
- [Lis81] Barbara Liskov, et al. *CLU Reference Manual*. Lecture Notes in Computer Science 114, Springer-Verlag, Berlin, 1981.
- [McC60] John McCarthy. *Recursive functions symbolic expressions and their computation by machine, Part I*. Communications of the ACM, Volume 3, Issue 4 (April 1960), sivut 184-195. ACM Press, New York, USA.
- [Mee84] L.G.L.T. Meertens, S. Pemberton. *Description of B*. CWI Note CS-N8405, CWI, Amsterdam, 1984.
- [Mee92] L.G.L.T. Meertens, S. Pemberton, G. van Rossum. *The ABC structure editor: Structure-based editing for the ABC programming environment*. CWI Report CS-R9256, sivu 2, CWI,

- Amsterdam, 1992. [Myös
<http://www.cwi.nl/ftp/CWIREports/AA/CS-R9256.ps.Z>,
 26.3.2004]
- [PEP1] Barry A. Warsaw, Jeremy Hylton, David Goodger. *Python Enhancement Proposal 1: PEP Purpose and Guidelines*.
<http://www.python.org/peps/pep-0001.html> (26.3.2004)
- [PEP202] Barry Warsaw *Python Enhancement Proposal 202: List Comprehensions*.
<http://www.python.org/peps/pep-0202.html> (28.3.2004)
- [PEP234] Ka-Ping Yee, Guido van Rossum. *Python Enhancement Proposal 234: Iterators*.
<http://www.python.org/peps/pep-0234.html> (28.3.2004)
- [PEP255] Neil Schemenauer, Tim Peters, Magnus Lie Hetland. *Python Enhancement Proposal 255: Simple Generators*.
<http://www.python.org/peps/pep-0255.html> (28.3.2004)
- [PyPy] *PyPy-projektin kotisivu*.
<http://codespeak.net/pypy/> (28.3.2004)
- [Pyt] *Python-kielen ja CPython-toteutuksen kotisivu*.
<http://www.python.org/> (28.3.2004)
- [Ros91] Guido van Rossum. *Python distribution posted to alt.sources*. Uutisryhmässä comp.sys.sgi 20.2.1991.
 Message-ID: <2986@charon.cwi.nl>
 Google Groups -arkisto: <http://groups.google.com/groups?selm=2986%40charon.cwi.nl> (26.3.2004)
- [Ros03a] Guido van Rossum, Fred L. Drake, Jr. *Python Reference Manual, Release 2.3.3*. PythonLabs, 2003.
<http://www.python.org/doc/2.3.3/ref/ref.html>
 (26.3.2004)

- [Ros03b] Guido van Rossum, Fred L. Drake, Jr. *Python Library Reference, Release 2.3.3*. PythonLabs, 2003.
<http://www.python.org/doc/2.3.3/lib/lib.html>
(26.3.2004)
- [Ros03c] Guido van Rossum, Fred L. Drake, Jr. *Extending and Embedding the Python Interpreter, Release 2.3.3*. PythonLabs, 2003.
<http://www.python.org/doc/2.3.3/ext/ext.html>
(27.3.2004)
- [Sco00] Michael L. Scott. *Programming Language Pragmatics (International student edition)*. Morgan Kaufmann, 2000.
- [Tan76] Andrew S. Tanenbaum. *A Tutorial on Algol 68*. ACM Computing Surveys, Volume 8, Issue 2 (June 1976), s. 155-190. ACM Press.
- [Ven03] Bill Venners. *The Making of Python: A Conversation with Guido van Rossum, Part I*.
<http://www.artima.com/intv/python.html> (26.3.2004)