hyväksymispäivä arvosana

arvostelija

COBOL ja Python: Tyypitys ja laskennan kapselointi

Erkki Heino, Tero Huomo, Eeva Terkki

Helsinki 13.2.2013

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Tyyppijärjestelmät **1**

1.1 COBOL 1

1.1.1 Tyypitys 1

1.2 Python 1

1.2.1 Alkeis- ja perustyypit 1

1.2.2 Tyypitys 2

2 Laskennan kapselointi **4**

2.1 COBOL 4

2.1.1 Parametrivälitys 4

2.2 Python 5

2.2.1 Parametrivälitys 5

2.2.2 Poikkeuskäsittely ja rinnakkaisuus 7

3 Etuja ja haittoja **8**

Lähteet **9**

1 Tyyppijärjestelmät

1.1 COBOL

Kielen eri toteutuksille yhtenäisten perustyyppien lisäksi COBOLin kääntäjien toteuttajat ja levittäjät tarjoavat usein lisää valmiita perustyyppejä [Wik13a]. Suurin osa tarjoaa esimerkiksi osoittimet ja bittikentät (bit field).

1.1.1 Tyypitys

COBOLin tyypityksestä on ristiriitaisia käsityksiä. Wikipedian [Wik13a] mukaan COBOL on vahvasti ja staattisesti tyypitetty kieli. Michael Coughlanin COBOL Tutorial [Cou99a] kuitenkin väittää, ettei kääntäjä tee tyyppitarkistuksia, ja COBOL olisi heikosti tyypitetty. Coughlanin mukaan esimerkiksi ei-numeerinen arvo on mahdollista sijoittaa numeerista tietoa sisältävään muuttujaan. Kun väärää tietotyyppiä sisältävällä muuttujalla yritetään tehdä laskutoimituksia, COBOL-ohjelma kaatuu suorituksenaikaiseen virheeseen.

Vuonna 2002 julkaistu Object Oriented COBOL sisältää sekä vahvan että heikon tyypityksen piirteitä [AC96].

1.2 Python

1.2.1 Alkeis- ja perustyypit

Pythonissa ei ole erikseen alkeistyyppejä, vaan Pythonissa kaikki on ilmaistu olioina tai olioiden välisinä suhteina [Pyt13a]. Jokaisella oliolla on *identiteetti* (identity), tyyppi ja arvo. Identiteetti on olion luonnin jälkeen muuttumaton. Identiteettiä voidaan ajatella olion osoitteena muistissa. Myös olion tyyppi on muuttumaton. Tietyissä kontrolloiduissa tilanteissa tyyppiä on mahdollista muuttaa, mutta usein tyyppimuunnoksilla voi olla odottamattomia seurauksia.

Kun kaikki on ilmaistu olioina, Pythonissa myös esimerkiksi None on olio. None on vastaavuus Java-kielen null-arvolle. Toisin kuin Pythonin None, Javan null ei kuitenkaan ole olio, eikä sillä ole tyyppiä.

Kielessä on valmiina useita kymmeniä sisäänrakennettuja tyyppejä, jotka on lueteltu Pythonin tyyppihierarkiassa [Pyt13a]. Valmiita numeerisia tyyppejä ovat esimerkiksi totuusarvot, kompleksiluvut, kokonaisluvut ja reaaliluvut. Pythonin versiossa 2

kokonaislukuja kuvaavat kiinteän mittainen `int` ja rajattoman pituisen kokonaisluvun mahdollistava `long`. Pythonin versiossa 3 ei enää ole kiinteän pituisia kokonaislukuja lainkaan, vaan ainoastaan `long` [Pyt13b]. Reaalilukujen ja kompleksilukujen arvoalueet voivat riippua käytettävästä konearkkitehtuurista, virtuaalimuistin määrästä sekä kääntäjän asetuksista. Pythonissa ei ole valmiina olemassa erillistä character-tyyppiä, vaan yksittäiset merkit ovat merkkijonoja. Erilaiset laajennusmoduulit lisäävät tyyppejä Javan kirjastojen tapaan.

1.2.2 Tyypitys

Python on vahvasti tyypitetty kieli [Stat09]. Tietyn tyyppiselle muuttujalle ei voida tehdä toisen tyypin operaatioita ennen eksplisiittistä tyyppimuunnosta.

```
a = 5
b = "9"
c = a + int(b)
```

Esimerkissä `b` sisältää merkkijonon "9", mutta yhteenlaskussa merkkijonosta jäsennetään kokonaisluku. Jos kokonaislukujäsennyksen jättää tekemättä, antaa ohjelma kyseisellä rivillä poikkeuksen.

Suoritusaikana muuttujan tyyppi ei ole sidottu, vaan muuttujaan voi dynaamisesti sitoa eri vaiheessa eri tyyppisiä olioita. Seuraavassa esimerkissä muuttuja `a` saa ensin kokonaislukuarvon 5. Sen jälkeen muuttujan `a` arvoksi muutetaan merkkijono "Hei".

```
a = 5
a = "Hei"
```

Pythonissa käytetään lisäksi *duck typing* -ohjelmointiparadigmaa [Wik13b]. Olioiden tyyppiä tärkeämpää ovat niiden toteuttamat ominaisuudet ja metodit. Käytännössä tämä tarkoittaa sitä, että funktioiden parametreina annettujen olioiden tyyppiä ei tarkisteta suoritettavassa funktiossa. Jos parametrina ollut olio ei toteuta funktion parametrilta kutsuttavia metodeja, funktio signaloi suorituksenaikaisen virhetilan-

teen.

```
class Ankka:
    def vaaku(self):
        print("Vaaaak!")
    def ui(self):
        print("Ankka pysyy pinnalla.")

class Ihminen:
    def vaaku(self):
        print("Ihminen imitoi ankkaa.")
    def ui(self):
        print("Ihminen uppoaa.")
    def name(self):
        print("Arto Wikla")

def ankkalampi(ankka):
    ankka.vaaku()
    ankka.ui()

aku = Ankka()
arto = Ihminen()
ankkalampi(aku)
ankkalampi(arto)
```

Esimerkissä määritellään kaksi luokkaa, `Ankka` ja `Ihminen` sekä funktio `ankkalampi`. Funktio ei välitä siitä, onko sen parametrin tyyppinä `Ankka`, vaan sitä voi kutsua myös luokan `Ihminen` ilmentymällä, sillä luokassa toteutetaan funktion vaatimat metodit. Javassa samantapainen käyttäytyminen toteutettaisiin perinnän tai rajapintojen avulla.

Duck typingin lisäksi Pythonissa käytetään tyyppipäättelyä ennen versiota 3 [Pyt13a]. Tyyppipäättelyn säännöt vaihtelevat Pythonin varhaisempien versioiden välillä suuresti, jonka vuoksi ohjelmointikielen referenssi ei tarjoa tarkkaa tyyppipäättelysäännöstöä, vaan ainoastaan epävirallisia ohjenuoria. Pythonin versiosta 3 tyyppipäättely on kokonaan karsittu.

2 Laskennan kapselointi

2.1 COBOL

2.1.1 Parametrivälitys

COBOLissa parametreja voidaan välittää joko arvo- tai viiteparametreina. Kutsuja määrittää, välitetäänkö parametri arvo- vai viitesemantiikalla. Kuvassa 1 nähdään esimerkki aliohjelmakutsuista.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ENSIMMAINEN.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 OMA    PIC X(20) VALUE "EKAN ARVO".
PROCEDURE DIVISION.
   * KUTSUTAAN TOISTA (ARVOPARAMETRI)
   CALL "TOINEN" USING BY VALUE OMA.
   * ENSIMMAISEN MUUTTUJA EI MUUTTUNUT
   DISPLAY OMA.
   * KUTSUTAAN TOISTA (VIITEPARAMETRI)
   CALL "TOINEN" USING BY REFERENCE OMA.
   * ENSIMMAISEN MUUTTUJA MUUTTUI
   DISPLAY OMA.
   END RUN.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TOINEN.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
77 PARAMETRI    PIC X(20).
PROCEDURE DIVISION USING PARAMETRI.
   MOVE "TOKAN ARVO" TO PARAMETRI.
   EXIT PROGRAM.
```

Kuva 1: Parametrivälitys COBOLissa

Ohjelmassa `ENSIMMAINEN` kutsutaan kaksi kertaa ohjelmaa `TOINEN`: ensimmäisellä kerralla välitetään muuttujan `OMA` arvo ja toisella viite samaiseen muuttujaan. Aliohjelma `TOINEN` muuttaa parametrina saadun muuttujan arvoa. Ohjelma tulostaa:

```
EKAN ARVO
TOKAN ARVO
```

Kutsu

```
CALL "TOINEN" USING BY VALUE OMA
```

tarkoittaa, että kutsutaan ohjelmaa `TOINEN` ja välitetään parametrina muuttujan

`OMA` arvo.

```
CALL "TOINEN" USING BY REFERENCE OMA
```

sen sijaan tarkoittaa, että samaiselle aliohjelmalle välitetään parametrina viite muuttujaan `OMA`.

Muistilistaa:

- "Support for complexity management is minimal in COBOL, with no language features which support any structures larger than subprogram modules."(http://archive.adaic.com/d
)

- Funktioista COBOL ANSI-85 (http://public.support.unisys.com/aseries/docs/clearpath-mcp-13.0/pdf/86001518-310.pdf)

- Object Oriented COBOL (2002) tukee nimettyjä funktioita ja funktioprototyyppejä, sekä tietysti olioita/perintää/polyformismia (http://en.wikipedia.org/wiki/COBOL)

- Kielessä ei luontaista tukea rinnakkaislaskennalle (http://archive.adaic.com/docs/reports/lawlis/
) mutta osa vendoreista tarjoaa apuvälineitä rinnakkaisuuteen (esim. http://supportline.microfocus
)

2.2 Python

Pythonissa funktioiden parametrit ovat arvoparametreja. Lähestulkoon kaikki esitetään kielessä olioina, ja arvoparametrit ovat viitteitä olioihin; ne siis toimivat samalla tavalla kuin parametrina annetut oliot Javassa.

2.2.1 Parametrivälitys

Funktioiden parametrit ovat tavallisesti ns. *positional-or-keyword*-parametreja [Pyt13c]. Tämä tarkoittaa sitä, että funktiokutsussa voi välittää argumentin joko sen sijainnin perusteella tai käyttämällä avainsanaa. Parametrille voi määritellä oletusarvon. Tällöin vastaavan argumentin voi jättää funktiokutsusta pois ja parametrin arvoksi annetaan määrätty oletusarvo. Ilman oletusarvoa olevat parametrit on listattava funktion määritelmässä ensimmäisenä, ja niiden jälkeen tulevat oletusarvolliset parametrit.

Seuraavan funktion parametreista kahdella on oletusarvo ja yhdellä ei:

```
def tervehdi(nimi, tervehdys="Hei", viesti=None):
    print tervehdys, nimi
    if viesti:
        print viesti
```

Parametrilla `tervehdys` on oletusarvo `hei` ja parametrilla `viesti` oletusarvo `None`. Funktio tulostaa parametrit `tervehdys` ja `nimi`, ja tutkii parametrin `viesti` arvoa. Jos se on jotakin muuta kuin tyhjä merkkijono tai `None`, arvo tulostetaan.

Pythonissa funktiokutsussa funktiolle välitettävät argumentit voivat olla avainsana-argumentteja (keyword argument) tai "tavallisia"argumentteja [Pyt13c]. Avainsana-argumentissa argumenttia edeltää tunniste, joka kertoo, mistä funktion parametrista on kyse. Avainsana-argumentit voivat olla funktiokutsussa keskenään eri järjestyksessä kuin vastaavat parametrit funktion määritelmässä. Argumentit, joissa ei käytetä parametrin tunnistetta, on annettava funktion määritelmän parametrilistauksen mukaisessa järjestyksessä.

Yllä määriteltyä funktiota `tervehdi` kutsuttaessa vain yksi argumentti on pakollinen. Funktiokutsut erilaisilla argumenteilla tuottavat seuraavanlaisia tulostuksia:

```
>>> tervehdi("Pekka", "Moi")
Moi Pekka

>>> tervehdi("Pekka", viesti="Kvaak kvaak")
Hei Pekka
Kvaak kvaak

>>> tervehdi(viesti="Opettele Pythonia", nimi="Pekka", tervehdys="Terve")
Terve Pekka
Opettele Pythonia
```

Funktio voidaan määrittää myös siten, että sitä voidaan kutsua mielivaltaisilla ar-

gumenteilla:

```
def mielivaltainen(*parametrit, **avainsanaparametrit):
    for parametri in parametrit:
        print parametri
    for avain, arvo in avainsanaparametrit.items():
        print avain, arvo
```

Funktio `mielivaltainen` tulostaa sille annetut argumentit ja avainsana-argumenttien nimet. Esimerkiksi kutsulla `mielivaltainen("eka", "toka", kolmas="kolmonen")` funktio tulostaa:

```
eka
toka
kolmas kolmonen
```

Avainsanattomat argumentit ovat saatavilla monikossa (tuple) `parametrit` ja avainsana-argumentit assosiaatiotaulussa (dictionary) `avainsanaparametrit`. Asteriskit määrittävät muodollisten parametrien tarkoituksen: yhdellä asteriskilla alkava parametri kuvaa avainsanattomia argumentteja ja kahdella asteriskilla alkava avainsana-argumentteja. Funktion määrittelyssä voidaan määrittää myös nimettyjä muodollisia parametreja, kunhan ne ovat ennen asteriskeilla varustettuja parametreja. `*`-alkuinen parametri on oltava aina ennen `**`-alkuista parametria funktion määrittelyssä [Pyt13e].

2.2.2 Poikkeuskäsittely ja rinnakkaisuus

Edellisessä luvussa määritellyn `tervehdi`-funktion kutsuminen esimerkiksi seuraavilla argumenteilla johtaisi suoritusaikaisiin virheisiin:

```
>>> tervehdi()
TypeError: tervehdi() takes at least 1 argument (0 given)

>>> tervehdi("Pekka", "Moi", tervehdys="Terve")
TypeError: tervehdi() got multiple values for keyword argument 'tervehdys'
```

Suoritusaikaisen virheen sattuessa Python-tulkki nostaa poikkeuksen [Pyt13d]. Poikkeus voidaan nostaa myös ohjelmakoodissa käyttämällä `raise`-lausetta. Poikkeus-

käsittelijä määritellään `try..except`-lauseella, joka muistuttaa Javan `try..catch`-rakennetta. Poikkeuskäsittelijällä voi olla useita `except`-haaroja, ja Javan tavoin tilanteeseen sopiva haara valitaan poikkeuksen luokan mukaan.

Pythonissa on tuki rinnakkaiselle ohjelmoinnille. Kielen `threading`-kirjasto tarjoaa tähän esimerkiksi säikeet, lukot ja semaforit. `Queue`-moduuli tarjoaa turvallisen tavan säikeiden käsittelyyn.

3 Etuja ja haittoja

Pythonissa laajasti hyödynnetty duck typing mahdollistaa samojen funktioiden käyttämisen useille olioille ilman rajapintoja ja perintää. Tyyppitarkistusten puuttuessa duck typing kuitenkin turvautuu siihen, että sitä hyödyntävät funktiot ja ohjelmakoodi ovat luotettavasti dokumentoituja ja testattuja. Esimerkiksi Javan rajapinnat ja perintä toimivat osaltaan selkeytyksenä ja dokumentaationa sille, mitä eri metodit vaativat parametreiltään.

Lähteet

AC96 Arranga, E. C. ja Coyle, F. P., Object-oriented cobol.

Cou99a Cobol tutorial: Selection, http://www.csis.ul.ie/cobol/course/DataDeclaration.htm. [12.2.2013].

Wik13b Duck typing, http://en.wikipedia.org/wiki/Duck_typing. [11.2.2013].

Pyt13d Python language reference – execution model, http://docs.python.org/2/reference/executionmodel.html#exceptions. [12.2.2013].

Pyt13c Glossary, http://docs.python.org/2/glossary.html. [12.2.2013].

Pyt13e The python tutorial - 4. more control flow tools, http://docs.python.org/2/tutorial/controlflow.html. [12.2.2013].

Pyt13a Python v2.7.3 documentation language reference - data model, http://docs.python.org/2/reference/datamodel.html. [12.2.2013].

Pyt13b Python v3.3 documentation language reference - data model, http://docs.python.org/3.3/reference/datamodel.html. [12.2.2013].

Stat09 Static vs. dynamic typing of programming languages, http://pythonconquerstheuniverse.wordpress.com/2009/10/03/static-vs-dynamic-typing-of-programming-languages/, 2009. [11.2.2013].

Wik13a Cobol, http://en.wikipedia.org/wiki/COBOL. [11.2.2013].