hyväksymispäivä arvosana

arvostelija

COBOL ja Python: Näkyvyysalueet ja laskennan ohjaus

Erkki Heino, Tero Huomo, Eeva Terkki

Helsinki 6.2.2013

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Näkyvyysalueet

Näkyvyysalueet määrittävät, missä osissa koodia muuttujat ovat käytettävissä. Tämän suhteen COBOL ja Python eroavat toisistaan paljon.

1.1 COBOL

COBOLissa kaikki muuttujat on määritettävä `DATA DIVISION` -osiossa ja nämä muuttujat ovat käytettävissä koko kyseisessä ohjelmassa [ans74, s. I-97]. COBOLin lohkorakenne on siis selvästi litteä.

Vielä vuoden 1974 ANSI-standardin mukaisessa COBOLissa jokainen ohjelma oli määriteltävä omassa tiedostossaan eivätkä aliohjelmat voineet käsitellä suoraan näitä kutsuvien ohjelmien muuttujia. Ohjelmalle voidaan kuitenkin välittää muuttujia viiteparametreina, jolloin se voi muuttaa näiden muuttujien arvoja myös kutsuvan ohjelman kontekstissa [ans74, s. XII-2].

COBOL-85 esitteli sisäkkäiset ohjelmat ja `GLOBAL`-lausekkeen. COBOL-85:ssa ohjelman sisällä voi määritellä aliohjelmia. Oletusarvoisesti kunkin ohjelman muuttujat ovat käytettävissä ainoastaan kyseisen ohjelman sisällä, mutta jos muuttujan määrittelyssä on käytetty `GLOBAL`-lauseketta, on se käytettävissä myös sisemmissä ohjelmissa. Jos useammassa sisäkkäisessä ohjelmassa on määritelty sama ”globaali” muuttuja, näkyy sisimpiin ohjelmiin näihin nähden sisin määritelmä muuttujasta [RD89, s. 439-441].

1.2 Python

Python käyttää dynaamista sidontaa [Ros98]. Pythonin lohkorakenne on syvä, ja ohjelman suoritusaikana käytössä on ainakin kolme sisäkkäistä näkyvyysaluetta [Pyt13d]. Näkyvyysalueita käytetään dynaamisesti. Sisimmällä näkyvyysalueella ovat paikalliset nimet. Mahdollisilla funktioita ympäröivillä funktioilla on omat näkyvyysalueensa, joiden sisältämät nimet eivät ole paikallisia eivätkä globaaleja. Toisiksi uloimmalla näkyvyysalueella ovat moduulin globaalit nimet ja kaikkein uloimmalla kieleen rakennetut nimet.

Pythonissa luokan näkyvyysalueella määritellyt nimet eivät näy luokan metodeille [Pyt13b]. Metodin ensimmäinen argumentti, jolle on tapana antaa nimi `self`, edustaa luokan ilmentymää. Sen kautta metodi voi käyttää luokan ilmentymän muita

metodeja ja attribuutteja.

Seuraavassa esimerkissä on kaksi sisäkkäistä funktiota:

```
def f1():
    a =  1
    def f2():
        b = 2
        print a + b
    print a
    f2()

f1()
```

Funktio `f1` määrittelee muuttujan `a` ja funktion `f2`, tulostaa `a`:n arvon ja kutsuu määrittelemäänsä funktiota. Funktio `f2` määrittelee muuttujan `b` ja tulostaa muuttujien `a` ja `b` arvojen summan. Ohjelma tulostaa luvut 1 ja 3. Muuttuja `a` on näkyvissä funktion `f1` ja sen sisäisten funktioiden sisällä. Muuttuja `b` puolestaan on paikallinen muuttuja, joka on näkyvissä vain `f2`-funktion sisällä. Siihen viittaaminen `f2`-funktion ulkopuolella johtaisi virhetilanteeseen.

Seuraavassa esimerkissä käytetään globaalia muuttujaa:

```
g = 1

def f3():
    global g
    g = 2
    print g

f3()
print g
```

Esimerkissä globaalin muuttujan `g` arvoksi alustetaan ensin luku 1. Funktio `f3` asettaa `g`:n arvoksi luvun 2 ja myös tulostaa muuttujan arvon. Kun funktiota `f3` kutsutaan ja sen jälkeen vielä tulostetaan `g`:n arvo, ohjelma tulostaa kaksi kertaa luvun 2. Funktio `f3` siis käsittelee globaalia muuttujaa. Avainsana `global` on tärkeä, sillä se ilmaisee, että kyseinen tunnus tulkitaan globaalin muuttujan tunnukseksi. Ilman

koodiriviä `global g` funktion määritelmän sisällä oleva muuttuja olisi paikallinen muuttuja, ja esimerkkiohjelma tulostaisi luvut 2 ja 1.

Pythonissa kaikki esitetään olioina. Lisäksi kaikki asiat, jotka voidaan nimetä, ovat ensimmäisen luokan arvoja – myös funktiot, metodit ja moduulit [Gui09].

2 Kontrollin ohjaus

Ohjelmointikielen tarjoamat rakenteet kontrollin ohjaukseen ovat hyvin tärkeitä kielen käytön kannalta. Monipuoliset kontrollinohjausrakenteet mahdollistavat erilaisten ongelmien ratkaisemisen selkeällä tavalla.

2.1 Python

Modernina kielenä Python sisältää tehokkaita keinoja ohjata laskentaa.

2.1.1 Laskentajärjestys

Pythonin operaattoreiden laskentajärjestys on 16-tasoinen [Pyt13c]. Ensimmäisenä laskentajärjestyksessä huomioidaan esimerkiksi listojen ja hajautustaulujen alkioiden erottimet. Viimeisenä lasketaan lambda-lausekkeet, joilla Pythonissa ilmaistaan anonyymeja funktioita. Aritmeettiset operaatiot ja bittioperaatiot lasketaan samassa järjestyksessä kuin C-pohjaisissa kielissä. Laskentajärjestyksen tasojen runsaan määrän vuoksi operaatioiden laskentajärjestyksen ilmaisemiseen kannattaa yleensä käyttää sulkuja.

2.1.2 Valinta ja toisto

Pythonissa käytetään valintaan ehtolauseita `if`, `elif` ja `else`. Ehtolauseita käytettäessä ehto tulkitaan epätodeksi, jos ehdon arvo on `False`, `None` (olio, jota käytetään ilmaisemaan arvon puuttumista), minkä hyvänsä tyyppinen numeerinen arvo nolla tai tyhjä merkkijono tai tietorakenne (kuten lista tai hajautustaulu) [Pyt13a]. Muussa tapauksessa ehto tulkitaan todeksi, ellei oliota ole erikseen määritelty tulkittavaksi epätodeksi.

Toistoon voidaan käyttää `for`-toistolausetta, ehdollista `while`-lausetta tai rekursiota. Käytettäessä `while`-lausetta ehto tulkitaan samalla tavalla kuin ehtolausei-

den yhteydessä. Muutoin `while` toimii samalla periaatteella kuin Javassa. Monessa muussa kielessä esiintyvää `do-while`-toistorakennetta Pythonissa ei ole.

Pythonin `for`-lause muistuttaa Javan `for-each`-silmukkaa, ja sillä voidaan iteroida iteroitavia olioita, kuten listoja, merkkijonoja ja iteraattoreita [Pyt13a]. Iteraattori on olio, jolla on metodi `next()`, joka palauttaa seuraavan alkion. Iteraattori voi käydä läpi esimerkiksi listan alkioita tai luonnollisten lukujen joukkoa.

Seuraava esimerkkiohjelma tulostaa parittomat luvut väliltä [1, 10[:

```
for luku in range(1, 10):
    if (luku % 2 != 0):
         print luku
```

Ohjelmassa käytetään Pythonin (version 2) funktiota `range`, joka tässä esimerkissä palauttaa järjestetyn listan kaikista kokonaisluvuista funktion argumentteina annettujen kahden luvun välillä. Lista käydään läpi `for`-silmukassa, ja jokaisen parittoman luvun kohdalla kyseinen luku tulostetaan.

Pythonin erikoisuutena `while`- ja `for`-lauseiden yhteydessä voidaan käyttää `else`-lausetta [Pyt13a]. Jos `while`-silmukan lopussa on `else`, `else`-haaran koodi suoritetaan, kun `while`:n ehto on epätosi ja silmukka päättyy. Jos `for`-lauseen lopussa on `else`, `else`-haara suoritetaan, kun iteroitavan olion kaikki alkiot on käyty läpi. Kummassakaan tapauksessa `else`-haaran koodia ei suoriteta, jos silmukka päättyy `break`- tai `continue`-komentoon. Pythonin `break` ja `continue` toimivat samalla tavalla kuin Javan vastaavat komennot.

Pythonissa toisto voidaan toteuttaa myös käyttämällä rekursiota. Python ei kuitenkaan tue häntärekursion eliminointia [Ros09b], minkä vuoksi häntärekursiota hyödyntäviä funktioita ei voida toteuttaa tehokkaasti.

2.2 COBOL

COBOLissa on Pythonia yksinkertaisempi kontrollin ohjaus. Vanhana kielenä se ei alunperin sisältänyt nykyaikaisissa kielissä lähes itsestään selviä ominaisuuksia kuten rekursiota.

2.2.1 Laskentajärjestys

COBOLin aritmeettisten lausekkeiden evaluoinnin järjestys riippuu COBOLin versiosta ja kääntäjän toteuttajasta. Compaqin COBOLissa laskentajärjestyksessä on neljä tasoa [Com02]. Ensimmäisenä määritetään muuttujien etumerkit. Tämän jälkeen potenssit, jonka jälkeen kerto- ja jakolaskut. Viimeisenä suoritetaan yhteen- ja vähennyslaskut.

COBOLin viimeisimmissä versioissa kieleen tulivat mukaan myös bittioperaatiot. Micro Focus'n Visual COBOL -toteutuksessa tämä on nostanut laskentatasoja seitsemään [Mic13]. Bittioperaatioista `NOT` evaluoidaan yhdessä etumerkkien kanssa. Muuten bittioperaatiot evaluoidaan viimeisinä järjestyksessä `AND`, `XOR` ja `OR`.

2.2.2 Valinta

Valinta suoritetaan rakenteella `IF - THEN - ELSE - ENDIF` [Cou99a].

```
* Yksinkertainen valinta
    IF NUMERO = 2
      DISPLAY 'KAKSI'
    END-IF

* IF ELSE -lause
    IF NUMERO IS EQUAL TO 2 THEN
        DISPLAY 'KAKSI'
    ELSE
        DISPLAY 'EI KAKSI'
    END-IF

* sisäkkäinen IF ELSE -lause
    IF NUMERO = 1
        DISPLAY 'NUMERO'
    ELSE
        IF NUMERO = 2
            DISPLAY 'KAKSI'
        ELSE
            DISPLAY 'EI YKSI EIKÄ KAKSI'
        END-IF
```

```
    END-IF
```

Esimerkistä huomataan, että rakenne vastaa pitkälti Javan If-valintalausetta. Avainsana `THEN` on vapaaehtoinen [Kar13], mutta mahdollistaa luonnollisen englanninkielen kaltaisen lausemuodon. If-rakenne lopetetaan avainsanalla `END-IF`, joka on mahdollista korvata myös pisteellä. COBOL ei kuitenkaan tue monen modernin kielen käyttämää else-if -valintaa. Else-if on korvattava esimerkin kaltaisilla sisäkkäisillä if-else -valintalauseilla.

COBOLin vastaavuus Javan tai C:n switch-case -rakenteelle on `EVALUATE`-verbi [Cou99a].

```
EVALUATE VALIKKO-SYOTE
    WHEN "0"
        DISPLAY 'VALITSIT 0'
    WHEN "1" THRU "9"
        DISPLAY 'VALITSIT 1-9'
END-EVALUATE.
```

Esimerkissä EVALUATE -verbiä seuraa evaluaation subjekti, `VALIKKO-SYOTE`. Avainsana `WHEN` vastaa pitkälti switch-case -rakenteen `case`a. Evaluaation lopettaa avainsana `END-EVALUATE`. Esimerkissä jos `VALIKKO-SYOTE` on 0, näytetään käyttäjälle "VALITSIT 0". Jos se on yhden ja yhdeksän väliltä, näytetään "VALITSIT 1-9". Toisin kuin Javan switch-case, EVALUATE tukee useita samanaikaisia vertailuja - muuttujia ja ehtoja voi ketjuttaa avainsanan `ALSO` avulla.

2.2.3 Toisto

COBOL tukee pitkälti samoja toistorakenteita kuin Java. Tässä lueteltavat toistorakenteet esittelee Cobol Tutorial [Cou99b]. COBOLin vastaavuus while-rakenteelle ja do-while-rakenteelle on `PERFORM UNTIL`, jossa toistoehdon voi sijoittaa alkuun tai loppuun.

Javan for-toistorakennetta vastaa `PERFORM - TIMES` ja `PERFORM VARYING`. Seuraava

esimerkki tulostaa käyttäjälle viisi kertaa sanan "HEI":

```
PERFORM 5 TIMES
    DISPLAY "HEI"
END-PERFORM
```

`PERFORM VARYING` mahdollistaa Javan tapaisen for-toistorakenteen, jossa muuttujan arvoa korotetaan yhden kierroksen jälkeen muulla kuin yhdellä.

```
PERFORM VARYING NUMERO FROM 1 BY 2
    UNTIL NUMERO > 5
DISPLAY 'Numero on nyt: ' NUMERO
END-PERFORM
```

Esimerkissä muuttujaa NUMERO korotetaan jokaisen kierroksen jälkeen kahdella. Esimerkin tulostus:

```
Numero on nyt: 1
Numero on nyt: 3
Numero on nyt: 5
```

2.2.4 Rekursio

Alunperin COBOL ei mahdollistanut rekursiota lainkaan [Wik13]. Uudemmista COBOLin versioista ainakin IBM:n COBOL-toteutus tukee rekursiota. IBM:n toteutuksessa rekursiivisesti kutsuttavan ohjelman tai aliohjelman Program-ID:ssä on annettava ehto `RECURSIVE` [IBM04]. Mikäli ohjelmaa kutsuu rekursiivisesti ilman rekursiosta eksplisiittisesti kertovaa `RECURSIVE`-ehtoa, ohjelman suoritus päättyy.

3 Etuja ja haittoja

Pythonissa kaikki arvot ovat ensimmäisen luokan arvoja, ja funktiot ensimmäisen luokan arvoina mahdollistavat kielen käyttämisen myös funktionaaliseen ohjelmointiin. Häntärekursion optimoinnin puuttuminen tosin rajoittaa tehokasta funktionaalista ohjelmointia.

COBOLin valintalauseiden haittana on else if -avainsanan puuttuminen, jolloin haaroittuvan valinnan voi joutua rakentamaan useilla sisäkkäisille if-else -valinnoilla. Toisaalta kielelle ominainen `EVALUATE` helpottaa useiden ehtojen yhtäaikaista vertailua.

Rekursion ja laskentatasojen vähäisyys voi vaikeuttaa koodillisesti siistien algoritmien ohjelmoimista. COBOLin pääkäyttökohteena on kuitenkin yritysmaailman ohjelmistot, joissa rekursiolle lienee vähemmän tarvetta kuin tieteellisen laskennan piirissä.

Lähteet

ans74 *American national standard programming language COBOL.* ANSI, New York, USA, 1974.

Cou99b Cobol tutorial: Iteration, `http://www.csis.ul.ie/cobol/course/Iteration.htm`. [5.2.2013].

Cou99a Cobol tutorial: Selection, `http://www.csis.ul.ie/cobol/course/Selection.htm`. [5.2.2013].

Com02 Compaq cobol reference manual: Arithmetic expressions, `http://h30266.www3.hp.com/odl/vax/progtool/cobol57a/6296/6296_profile_019.html`. [5.2.2013].

Kar13 Cobol - if else endif statement, `http://code.xmlgadgets.com/2012/03/28/cobol-if-else-endif-statement/`. [5.2.2013].

IBM04 Enterprise cobol for z/os - programming guide: Making recursive calls, `http://pic.dhe.ibm.com/infocenter/pdthelp/v1r1/index.jsp?topic=%2Fcom.ibm.entcobol.doc_3.4%2Ftpsubw03.htm`. [5.2.2013].

Pyt13a The python language reference – compound statements, `http://docs.python.org/2/reference/compound_stmts`. [6.2.2013].

Pyt13b The python language reference – execution model, `http://docs.python.org/2/reference/executionmodel.html#naming-and-binding`. [4.2.2013].

Pyt13c The python language reference – expressions, `http://docs.python.org/2/reference/expressions.html`. [6.2.2013].

Pyt13d The python tutorial – classes, `http://docs.python.org/2/tutorial/classes.html`. [2.2.2013].

RD89 Roy, K. ja Dastidar, D., *Cobol Programming.* Tata McGraw-Hill, 1989. URL `http://books.google.fi/books?id=N066w1XgJXcC`.

Mic13 Micro focus visual cobol for visual studio 2010: Formation and evaluation rules, `http://documentation.microfocus.com/help/index.jsp?topic=%2Fcom.microfocus.eclipse.infocenter.visualcobol.vs%2FHRLHLHCLANU058.html`. [5.2.2013].

Ros98 van Rossum, G., Glue it all together with python, http://www.python.org/doc/essays/omg-darpa-mcc-position.html, 1998. [6.2.2013].

Gui09 van Rossum, G., First-class everything, http://python-history.blogspot.fi/2009/02/first-class-everything.html, 2009. [2.2.2013].

Ros09b van Rossum, G., Tail recursive elimination, http://neopythonic.blogspot.com.au/2009/04/tail-recursion-elimination.html, 2009. [6.2.2013].

Wik13 Cobol, http://en.wikipedia.org/wiki/COBOL. [5.2.2013].