Oskari Lahtinen

Keskeisistä haasteista funktionaalisen ohjelmointiparadigman oppimisessa

Tietotekniikan kandidaatintutkielma

4. toukokuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Oskari Lahtinen

Yhteystiedot: oskari.a.lahtinen@jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Keskeisistä haasteista funktionaalisen ohjelmointiparadigman oppimisessa

Title in English: On the key challenges in learning the functional programming paradigm

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 17+0

Tiivistelmä: Funktionaalisten ohjelmointikielten ominaisuuksia on lisätty viime vuosikymmeninä myös muihin ohjelmointikieliin. Funktionaalisen ohjelmointiparadigman oppimiseen liittyy kuitenkin useita haasteita. Näitä haasteita ovat esimerkiksi erilainen syntaksi, funktionaalisten ohjelmointikielten erityisominaisuudet sekä opiskelijoiden negatiiviset ennakkokäsitykset aiheen hyödyllisyydestä. Tässä kandidaatintutkielmassa tutkitaan ja eritellään tarkemmin esille tulevia haasteita ja esitettyjä ratkaisuehdotuksia.

Avainsanat: funktio-ohjelmointi, ohjelmointi, ohjelmoinnin opetus

Abstract: In recent decades, functional programming features have also been added to other programming languages. However, there are several challenges in learning the functional programming paradigm. These challenges include the different syntax, the unique features of functional programming languages, and negative preconceptions among students about the usefulness of the subject. This thesis will explore and elaborate on the identified challenges and proposed solutions.

Keywords: functional programming, programming, teaching programming

Sisällys

1 Johdanto

Tässä kandidaatintutkielmassa tutkitaan funktionaalisten ohjelmointikielten opiskelussa esiin nousevia haasteita. Funktionaalinen paradigma eli FP on ohjelmointityyli, jossa käytetään funktioita ohjelman perusrakenteena muiden rakenteiden sijaan. FP:n keskeisiä etuja ovat sillä tehtyjen ohjelmien toimintavarmuus sekä koodin luettavuus ja ylläpidettävyys.

FP:n ominaisuudet ovat viime vuosina tulleet myös muihin kuin puhtaisiin funktionaalisiin kieliin. Suosituimpia FP:n ominaisuuksia tarjoavia kieliä ovat Java, C# sekä C++. Funktionaalinen paradigma tarjoaa vaihtoehtoisia tapoja ratkaista yleisiä ohjelmointitehtäviä. Siksi funktionaalisten ohjelmointikielten opetusta järjestetään myös korkeakouluissa. (ACM Computing Curricula Task Force 2013)

Funktionaalisen paradigman oppimisessa on useita haasteita, kuten tiivis syntaksi, rekursion käyttö sekä korkeamman kertaluokan funktiot. Nämä toistuvat aihetta koskevassa tutkimuskirjallisuudessa keskeisinä vaikeina asioina. (Tirronen, Uusi-Mäkelä ja Isomöttönen 2015) (Motara 2020) (Chakravarty ja Keller 2004)

Tässä kirjallisuuskatsauksessa perehdytään FP:n oppimista käsittelevään tutkimuskirjallisuuteen. Tutkimuskirjallisuudessa painotetaan erityisesti korkeakouluissa järjestettyjä ohjelmointikursseja. Tätä kirjallisuuskatsausta voidaan toivottavasti hyödyntää tulevan tutkimuksen tukena muun muassa siten, että kohdennetaan resursseja selvittämään vastauksia auki jääneisiin kysymyksiin. Kirjallisuuskatsauksen toivotaan myös toimivan FP:n opetuksen tukena, jotta opiskelussa esiintyviä haasteita voitaisiin välttää.

Suuri osa löydetystä kirjallisuudesta koskee yksittäisten korkeakoulujen opetuskokeiluja, joten niiden tuloksia voi olla vaikeaa tai jopa mahdotonta yleistää kaikkeen opetukseen. (Tirronen, Uusi-Mäkelä ja Isomöttönen 2015)

Tutkimuksissa käytettyjä kieliä olivat muun muassa Haskell (Tirronen, Uusi-Mäkelä ja Isomöttönen 2015), Javascript (Nurminen, Niemelä ja Järvinen 2021), F# (Motara 2020), Standard ML (Hansen ja Kristensen 2008) sekä erilaiset Lisp-suvun kielet (Trivodaliev ym. 2017).

Luvussa 2 esitellään FP:n ominaisuuksia lyhyesti. Luvussa 3 kerrotaan tutkimuskirjallisuu-

dessa esitetyistä haasteista paradigman oppimisessa sekä esitellään haasteisiin esitettyjä ratkaisuja. Tutkielman lopuksi keskeiset havainnot kootaan yhteenvetoluvussa yhteen.

2 Funktionaalinen ohjelmointiparadigma

Tässä luvussa esitellään FP:n keskeiset ominaisuudet tarkemmin vapaamuotoisten koodiesimerkkien avulla. Esimerkeissä käytetty kieli mukailee vahvasti Haskellin syntaksia. (Hudak ja Fasel 1992)

2.1 Funktionaalisen paradigman ominaisuudet lyhyesti

Eri ohjelmointiparadigmojen väliset erot ovat usein häilyviä eikä ole olemassa tiettyä joukkoa ominaisuuksia, jotka määrittelisivät tiettyyn paradigmaan täysin kuuluvan ohjelmointikielen. Lisäksi suurin osa käytössä olevista ohjelmointikielistä hyödyntää usean paradigman ominaisuuksia. Erityisesti FP:stä tuttuja ominaisuuksia on lisätty 2000-luvun aikana alkujaan imperatiivisiin ja oliosuuntautuneisiin kieliin, kuten C++:n (McNamara ja Smaragdakis 2000) ja Javaan (Mazinanian ym. 2017). Myös monista korkean tason kielistä tuttu roskienkeruu (engl. *garbage collection*) on peräisin Lispistä.

Funktionaalisten ohjelmointikielten yleisiä ominaisuuksia ovat muun muassa korkeamman kertaluokan funktiot (engl. *higher-order functions*), laiska laskenta (engl. *lazy evaluation*), hahmonsovitus (engl. *pattern matching*), staattinen ja vahva tyyppijärjestelmä, algebralliset tietotyypit (engl. *algebraic datatypes*), rekursio tai taitos (engl. *fold*) silmukoiden sijaan, sivuvaikutusten välttäminen sekä nimettömät funktiot (engl. *lambda expression*). (Hudak 1989)

2.2 Korkeamman kertaluokan funktiot sekä nimettömät funktiot

Korkeamman kertaluokan funktiolla tarkoitetaan funktiota, joka joko palauttaa funktion tai ottaa yhden tai useamman funktion parametreinaan. Erityisesti jälkimmäiseen kytkeytyy vahvasti nimettömän funktion käsite; funktiota kutsuttaessa parametrilistaan voidaan antaa nimetön funktio. Ohessa koodiesimerkki, jossa lisätään lukuun 0 luku 1 kaksi kertaa:

Määritellään seuraavat funktiot:

```
twice f x = f (f x)
addOne x = x + 1
```

ja muodostetaan funktiokutsu, jossa ensimmäinen argumentti on funktio:

```
twice addOne 0
```

Sama nimetöntä funktiota käyttäen:

```
twice (\x -> x + 1) 0
```

2.3 Laiska laskenta

Laiska laskenta tarkoittaa sitä, että funktion arvo lasketaan vasta silloin kun sitä tarvitaan. Helppo tapa havainnollistaa laiskaa laskentaa on loputtoman mittainen lista. Loputtoman listan laskeva funktio palauttaa arvoja sitä mukaa kun niitä pyydetään, esimerkiksi viisi ensimmäistä kokonaislukua Fibonaccin lukujonosta. Ohjelma kaatuu, jos pyydetään lukujonon viimeistä lukua.

2.4 Rekursio ja hahmonsovitus

Rekursio tarkoittaa funktiota, joka kutsuu itseään laskennan suorittamiseksi. Rekursiolle määritellään perustapaus, jolloin rekursio lopetetaan ja palautetaan saatu arvo. Mikäli perustapausta ei ole saavutettu, rekursiivinen funktio kutsuu itseään, usein muokatuilla arvoilla. Rekursion käyttäminen voi olla tilanteesta riippuen epätehokasta varsinkin suurilla aineistoilla, sillä jokaisesta funktiokutsusta on pidettävä ohjelman muistissa kirjaa kutsupinossa, mikäli käytetty kieli ei tue häntäkutsujen optimointia (engl. *tail-call optimization*). (Bailey ja Weston 2001) Funktionaalisessa ohjelmoinnissa käytetäänkin usein taitosta rekursion sijaan paremman suorituskyvyn ja luettavuuden vuoksi. (Hutton 1999)

Hahmonsovituksella voidaan katsoa käsiteltävänä olevaa argumenttia ja tehdä sille jotain sen arvon perusteella. Yksinkertaisena esimerkkinä hahmonsovituksella voidaan tarkistaa, onko argumenttina saatu kokonaisluku esimerkiksi nolla, ja suorittaa tällöin funktio eri tavalla. Monimutkaisemmin hahmonsovitusta voidaan käyttää purkamaan algebrallisia tietotyyppejä

osiin.

Ohessa yksinkertainen koodiesimerkki rekursiosta sekä hahmonsovituksesta:

```
fibonacci 0 = 0
fibonacci 1 = 1
fibonacci n = fibonacci (n-1) + fibonacci (n-2)
```

Fibonacci-funktio kutsuu itseään rekursiivisesti pienemmillä luvun n arvoilla ja hahmonsovituksella tarkistetaan, onko määritellyt perustapaukset saavutettu vielä. Mikäli hahmonsovitusta ei olisi, jouduttaisiin käyttämään if-lausetta.

2.5 Algebralliset tietotyypit ja tyyppiluokat

Algebralliset tietotyypit ovat tyyppejä, jotka koostuvat muista tyypeistä. Esimerkiksi yksinkertainen linkitetty lista koostuu sekä itse listasta että listan sisällä olevista alkioista.

Algebrallisten tietotyyppien muodostuksessa käytetään usein tyyppimuuttujia. Tyyppimuuttujien avulla tietotyypit voivat sisältää kaiken tyyppisiä arvoja, eikä esimerkiksi listaa ja sen operaatioita tarvitse määritellä jokaiselle tyypille erikseen. Tätä kutsutaan parametriseksi polymorfismiksi. Parametrisella polymorfismilla päästään yhä tyyppiturvallisiin ohjelmiin, mutta kielen ilmaisuvoima kasvaa merkittävästi. Valitettavasti polymorfismin sisällyttäminen ohjelmointikieleen aiheuttaa myös hankalasti tulkittavia virheilmoituksia.

Tyypit voivat kuulua tyyppiluokkiin, jotka lupaavat kaikkien siihen kuuluvien tyyppien toteuttavan tiettyjä funktioita. Haskellissa yleisiä tyyppiluokkia ovat esimerkiksi Eq yhtäsuuruuden tarkistamiseen, Show merkkijonoksi muuttamiseen sekä Num numerotyypeille. Tyyppiluokkiin kuuluvien tyyppien on toteutettava tyyppiluokkien määrittelemät funktiot, esimerkiksi Eq-tyyppiluokkaan kuuluvan tyypin on toteutettava vähintään yhtäsuuruutta vertaileva ==-operaattori. Funktiot voivat lisäksi vaatia argumenteiltaan johonkin tyyppiluokkaan kuulumista. Tätä kutsutaan tyyppiluokkarajoitteeksi.

Määritellään ensin itse minimaalinen tyyppiluokka Eq, jolla voidaan tarkistaa, onko kaksi samantyyppistä asiaa sisällöiltään samat:

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
```

Luodaan algebrallinen datatyyppi Pari (tyyppiteoriassa niin kutsuttu tulotyyppi), jossa on kaksi alkiota ja joka kuuluu tyyppiluokkaan Eq:

```
data Pari a b = Prod a b
instance Eq Pari where
  (Prod a b) == (Prod c d) = a == c && b == d
  a /= b = not (a == b)
```

Kyseisessä määritelmässä oletetaan, että tyypit a ja b toteuttavat tyyppiluokan Eq.

Tehdään vielä funktio, joka summaa kaksi samantyyppistä parametria yhteen. Tässä käytetään hyödyksi funktioiden mahdollisuutta tyyppiluokkarajoitteisiin.

```
f :: (Num a) => a -> a
f x y = x + y
```

3 Funktionaalisen ohjelmointiparadigman opiskelusta

Tässä luvussa käsitellään FP:n opiskelussa vaikeiksi osoittautuneita asioita sekä niihin ehdotettuja ratkaisuja. Nämä vaikeat asiat voidaan jakaa karkeasti kolmeen eri ryhmään: FP-kielten syntaksi, FP:n ominaisuudet sekä opiskelijoiden ennakkokäsitykset FP:stä.

3.1 Syntaksi

Sekä Lispeissä että Haskellissa syntaksi on huomattavasti erilaista kuin esimerkiksi syntaksiltaan C-tyylisissä kielissä (C/C++, Java, C#, osittain Python ja Javascript), jotka ovat suosittuja alkeisohjelmoinnissa. Uudenlaisen syntaksin opettelu tuottaa usein aloittelevalle ohjelmoijalle ongelmia. (Tirronen, Uusi-Mäkelä ja Isomöttönen 2015)

Lisäksi kääntäjän tuottamat virheilmoitukset voivat olla hankalasti ymmärrettäviä. (Heeren, Leijen ja IJzendoorn 2003) Puutteellisten tai harhaanjohtavien virheilmoitusten vuoksi syntaksivirheiden korjaaminen voi olla monimutkaista, aikaa vievää sekä turhauttavaa. Virheilmoitusten ymmärrettävyydessä on paljon eroja jopa saman kielen eri kääntäjien ja tulkkien välillä. (Denny, Luxton-Reilly ja Tempero 2012)

Haskellin opiskelua varten on kehitetty Helium-niminen kääntäjä, jonka virheilmoitukset ovat huomattavasti helpommin ymmärrettäviä. (Heeren, Leijen ja IJzendoorn 2003) Heliumissa ei ole tukea tyyppiluokille. Myös Haskellista paljon vaikutteita ottanut ohjelmointikieli Elm tähtää helppolukuisiin virheilmoituksiin kääntäessä. (Czaplicki ja Chong 2013) Elmissäkään ei ole tukea tyyppiluokille ja sitä käytetään pääasiassa web-ohjelmoinnissa.

Tyyppiluokkien puutteen vuoksi Helium ja Elm eivät ole yhtä tehokkaita kuin Haskell. Ne sopivat vain rajattuihin käyttökohteisiin, kuten FP:n opetteluun. Heliumin käyttämisestä opetuksessa on positiivisia kokemuksia. (Heeren, Leijen ja IJzendoorn 2003)

Myös funktionaalisten kielten modulaarisuus ja tiivis ilmaisutapa mainitaan haastavina ymmärtää.

3.2 Ominaisuudet

FP:n ominaisuudet poikkeavat monin tavoin imperatiivisten ohjelmointikielten ominaisuuksista. Niiden oppiminen ja hyödyntäminen onkin keskeisessä osassa funktionaalista ohjelmointia käsittelevillä kursseilla.

Rekursio saattaa olla vaikea konsepti, mikäli taustalla on kokemusta imperatiivisista kielistä. Opiskelijat pyrkivät käyttämään itselleen tuttuja rakenteita, kuten silmukoita. (Motara 2020) Rekursion perustapauksien muodostaminen koetaan myös vaikeaksi tai unohdetaan kokonaan. (Haberman ja Averbuch 2002)

Myös vahva, Hindley-Damas-Mildner -tyyppinen tyyppijärjestelmä voi olla vaikea. Haskellin tyyppijärjestelmää ajatellaan usein samanlaiseksi kuin esimerkiksi Javassa. Opiskelijat saattavat ajatella funktion tyypin olevan sama kuin sen paluuarvon. Tämä lähestymistapa osoittautuu ongelmalliseksi, kun hyödynnetään korkeamman kertaluokan funktioita. (Tirronen ja Isomöttönen 2015)

Hindley-Damas-Mildner -tyyppijärjestelmää varten on ehdotettu vaihe kerrallaan laskutoimituksin etenevää ratkaisutapaa funktion tyyppien selvittämiseksi. (Tirronen ja Isomöttönen 2015)

Joissakin tutkimuksissa todettiin kielen ominaisuuksien opetusjärjestyksen vaikuttavan huomattavasti oppimiskynnykseen. Todetaan, että kurssin loppupuolen asiat saattavat jäädä vähemmälle huomiolle ja siten tuntua vaikeammilta oppia. Esimerkiksi mikäli siirräntä (engl. *input/output*) käytiin ensimmäisillä luennoilla, ohjelmointitehtävistä tuli huomattavasti konkreettisempia ja motivoivampia. (Hughes 2008) Vaikka siirräntä onkin Haskellissa monadi, sen käyttäminen on helppoa. Siirräntämonadiin liittyvän teorian voi jopa tarvittaessa jättää kurssin oppimistavoitteiden ulkopuolelle.

Yksittäiset ominaisuudet eivät välttämättä ole vaikeita itsekseen. Vaikeuksia aiheuttaa ominaisuuksien yhdistäminen ja/tai ohjelmointitehtävän pilkkominen hallittavan kokoisiksi osaongelmiksi. (Motara 2020)

Myös muissa kielissä ominaisuuksien rajaaminen on osoittautunut tehokkaaksi. Yksinkertaisuudestaan tunnettu Scheme onkin suosittu opetuskieli. Scheme on hyvin yksinkertainen

Lisp. Schemen yleisimmin käytetty standardi Revised[5] Report on the Algorithmic Language Scheme mahtuu 50 liuskaan tekstiä. Sitä vastoin esimerkiksi ANSI Common Lispin standardin pituus on noin 1800 liuskaa.

3.3 Ennakkokäsitykset

Mielenkiintoinen haaste on myös opiskelijoiden ennakkokäsitykset funktionaalisista kielistä. Funktionaalisia kieliä saatetaan pitää turhina työllistymisen kannalta, sillä harvassa ohjelmistoalan yrityksessä etsitään funktionaalisten kielten osaajia. Ensimmäisen vuoden opiskelijat saattavatkin odottaa oppivansa ensimmäisenä ohjelmointikielenään jonkin ohjelmistoteollisuudessa laajasti käytetyn. (Hughes 2008)

Onkin tärkeää esitellä funktionaalisen paradigman ominaisuuksia ja niiden käyttöä myös ei-funktionaalisissa kielissä. Tärkeää on myös painottaa funktionaalisen ajattelutavan olevan suuremmassa asemassa kurssilla kuin jonkun tietyn kielen opiskelun. (Motara 2020) Ohjelmointikursseilla yleisesti kannattaa opettaa pääpainon olevan ohjelmoinnin peruskäsitteiden oppimisessa, eikä yksittäisten kielten oppimisessa. (Chakravarty ja Keller 2004)

Useamman eri ohjelmointiparadigman opettaminen opintojen varhaisessa vaiheessa lieventää edellä mainittuja negatiivisia ennakkokäsityksiä. (Joosten, Van Den Berg ja Van Der Hoeven 1993) Eri paradigmojen käyttö opettaa myös erilaisia lähestymistapoja ohjelmointiongelmiin. Funktionaalinen ohjelmointikieli saattaa tasoittaa opiskelijoiden välisiä tasoeroja, erityisesti opintojen varhaisessa vaiheessa. (Chakravarty ja Keller 2004)

Huomattavaa on myös se, että opettajien ennakkokäsitykset eivät välttämättä vastaa todellisuudessa hankalaksi koettuja asioita. (Brown ja Altadmri 2014) Tämän vuoksi opetuksessa saatetaan epähuomiossa painottaa opiskelijoille helppoja asioita ja sivuuttaa vaikeat asiat. Resurssien mahdollistaessa olisi tärkeää selvittää opiskelijoilta jo kurssin aikana, mistä aiheista he toivoisivat lisäopetusta ja muokata opetusta tämän mukaan.

4 Yhteenveto

Tässä kandidaatintutkielmassa käsiteltiin yleisimpiä FP:n opiskelussa kohdattavia haasteita. Haasteet voitiin jakaa karkeasti kolmeen ryhmään: syntaksi, FP:n ominaisuudet ja ennakkokäsitykset.

Vaikka tutkielmassa käsitellyssä tutkimuskirjallisuudessa mainittiin useita funktionaalisia ohjelmointikieliä, suurin osa keskittyi ML-sukuisten kielten opiskelussa ilmenneisiin haasteisiin. Lispiä ja Schemeä käyttäneissä tutkimuksissa korkeintaan sivuttiin kielen opiskelua ja niissä keskityttiin enemmän kurssiin ja paradigman opiskelun mielekkyyteen kokonaisuutena. Tämän vuoksi olisi mielenkiintoista selvittää, mitä jossakin Lispin sukuisessa kielessä koetaan vaikeaksi. On hyvä muistaa, että Lispit ovat luonteeltaan yksinkertaisia eikä niissä ole paljoa monimutkaisia ominaisuuksia.

Toinen kiinnostava tutkimusongelma on, kuinka paljon opiskelijoiden ennakkokäsitykset haittaavat opiskelua. Tutkielmassa todettiin opiskelijoilla olevan paljon ennakkokäsityksiä funktionaalisista ohjelmointikielistä. Eräs tapa edistää aiheeseen liittyvää tutkimusta olisi selvittää, vaikuttavatko ennakkokäsitykset kurssin läpipääsyprosenttiin, arvosanaan tai koettuun oppimiseen. Myös opiskelijan käsityksen mahdollinen muuttuminen kurssin aikana kiinnostaa.

Lähteet

ACM Computing Curricula Task Force, toimittanut. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM, Inc, 1. tammikuuta 2013. ISBN: 978-1-4503-2309-3, viitattu 8. maaliskuuta 2023. https://doi.org/10.1145/2534860. http://dl.acm.org/citation.cfm?id=2534860.

Bailey, Mark W ja Nathan C Weston. 2001. *Performance benefits of tail recursion removal in procedural languages.* Tekninen raportti. Tech. Rep. TR-2001-2, Hamilton College, Clinton, NY.

Brown, Neil C.C. ja Amjad Altadmri. 2014. "Investigating Novice Programming Mistakes: Educator Beliefs vs. Student Data". Teoksessa *Proceedings of the Tenth Annual Conference on International Computing Education Research,* 43–50. ICER '14. Glasgow, Scotland, United Kingdom: Association for Computing Machinery. ISBN: 9781450327558. https://doi.org/10.1145/2632320.2632343. https://doi.org/10.1145/2632320.2632343.

Chakravarty, Manuel M. T. ja Gabriele Keller. 2004. "The risks and benefits of teaching purely functional programming in first year" [kielellä en]. *Journal of Functional Programming* 14, numero 1 (tammikuu): 113–123. ISSN: 0956-7968, 1469-7653, viitattu 13. tammikuuta 2023. https://doi.org/10.1017/S0956796803004805. https://www.cambridge.org/core/product/identifier/S0956796803004805/type/journal_article.

Czaplicki, Evan ja Stephen Chong. 2013. "Asynchronous Functional Reactive Programming for GUIs". *SIGPLAN Not.* (New York, NY, USA) 48, numero 6 (kesäkuu): 411–422. ISSN: 0362-1340. https://doi.org/10.1145/2499370.2462161. https://doi.org/10.1145/2499370.2462161.

Denny, Paul, Andrew Luxton-Reilly ja Ewan Tempero. 2012. "All Syntax Errors Are Not Equal". Teoksessa *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education,* 75–80. ITiCSE '12. Haifa, Israel: Association for Computing Machinery. ISBN: 9781450312462. https://doi.org/10.1145/2325296.2325318. https://doi.org/10.1145/2325296.2325318.

Haberman, Bruria ja Haim Averbuch. 2002. “The Case of Base Cases: Why Are They so Difficult to Recognize? Student Difficulties with Recursion”. Teoksessa *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education,* 84–88. ITiCSE ’02. Aarhus, Denmark: Association for Computing Machinery. ISBN: 1581134991. https://doi.org/10.1145/544414.544441. https://doi.org/10.1145/544414.544441.

Hansen, Michael R. ja Jens Thyge Kristensen. 2008. “Experiences with Functional Programming in an Introductory Curriculum” [kielellä en]. Teoksessa *Reflections on the Teaching of Programming,* toimittanut Jens Bennedsen, Michael E. Caspersen ja Michael Kölling, 4821:30–46. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-77934-6, viitattu 13. tammikuuta 2023. https://doi.org/10.1007/978-3-540-77934-6_4. http://link.springer.com/10.1007/978-3-540-77934-6_4.

Heeren, Bastiaan, Daan Leijen ja Arjan van IJzendoorn. 2003. “Helium, for Learning Haskell”. Teoksessa *Proceedings of the 2003 ACM SIGPLAN Workshop on Haskell,* 62–71. Haskell ’03. Uppsala, Sweden: Association for Computing Machinery. ISBN: 1581137583. https://doi.org/10.1145/871895.871902. https://doi.org/10.1145/871895.871902.

Hudak, Paul. 1989. “Conception, Evolution, and Application of Functional Programming Languages”. *ACM Comput. Surv.* (New York, NY, USA) 21, numero 3 (syyskuu): 359–411. ISSN: 0360-0300. https://doi.org/10.1145/72551.72554. https://doi.org/10.1145/72551.72554.

Hudak, Paul ja Joseph H. Fasel. 1992. “A Gentle Introduction to Haskell”. *SIGPLAN Not.* (New York, NY, USA) 27, numero 5 (toukokuu): 1–52. ISSN: 0362-1340. https://doi.org/10.1145/130697.130698. https://doi.org/10.1145/130697.130698.

Hughes, John. 2008. “Experiences from teaching functional programming at Chalmers” [kielellä en]. *ACM SIGPLAN Notices* 43, numero 11 (marraskuu): 77–80. ISSN: 0362-1340, 1558-1160, viitattu 13. tammikuuta 2023. https://doi.org/10.1145/1480828.1480845. https://dl.acm.org/doi/10.1145/1480828.1480845.

Hutton, Graham. 1999. “A tutorial on the universality and expressiveness of fold”. *Journal of Functional Programming* 9 (4): 355–372. https://doi.org/10.1017/S0956796899003500.

Joosten, Stef, Klaas Van Den Berg ja Gerrit Van Der Hoeven. 1993. “Teaching functional programming to first-year students” [kielellä en]. *Journal of Functional Programming* 3, numero 1 (tammikuu): 49–65. ISSN: 0956-7968, 1469-7653, viitattu 13. tammikuuta 2023. https://doi.org/10.1017/S0956796800000599. https://www.cambridge.org/core/product/identifier/S0956796800000599/type/journal_article.

Mazinanian, Davood, Ameya Ketkar, Nikolaos Tsantalis ja Danny Dig. 2017. “Understanding the Use of Lambda Expressions in Java”. *Proc. ACM Program. Lang.* (New York, NY, USA) 1, numero OOPSLA (lokakuu). https://doi.org/10.1145/3133909. https://doi.org/10.1145/3133909.

McNamara, Brian ja Yannis Smaragdakis. 2000. “Functional Programming in C++”. Teoksessa *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming,* 118–129. ICFP ’00. New York, NY, USA: Association for Computing Machinery. ISBN: 1581132026. https://doi.org/10.1145/351240.351251. https://doi.org/10.1145/351240.351251.

Motara, Yusuf Moosa. 2020. “Obstacles when teaching functional programming” [kielellä en]. Teoksessa *Proceedings of the 9th Computer Science Education Research Conference,* 1–2. Virtual Event Netherlands: ACM, lokakuu. ISBN: 978-1-4503-8872-6, viitattu 13. tammikuuta 2023. https://doi.org/10.1145/3442481.3442510. https://dl.acm.org/doi/10.1145/3442481.3442510.

Nurminen, Mikko, Pia Niemelä ja Hannu-Matti Järvinen. 2021. ““Why is this course pushing functional programming?” - educating well-rounded web developers with functional JavaScript” [kielellä en]. SEFI European Society for Engineering Education. ISBN: 978-2-87352-023-6, viitattu 5. helmikuuta 2023. https://trepo.tuni.fi/handle/10024/137737.

Tirronen, Ville ja Ville Isomöttönen. 2015. “Teaching types with a cognitively effective worked example format” [kielellä en]. *Journal of Functional Programming* 25:e23. ISSN: 0956-7968, 1469-7653, viitattu 13. tammikuuta 2023. https://doi.org/10.1017/S0956796814000021. https://www.cambridge.org/core/product/identifier/S0956796814000021/type/journal_article.

Tirronen, Ville, Samuel Uusi-Mäkelä ja Ville Isomöttönen. 2015. “Understanding beginners’ mistakes with Haskell” [kielellä en]. *Journal of Functional Programming* 25:e11. ISSN: 0956-7968, 1469-7653, viitattu 13. tammikuuta 2023. https://doi.org/10.1017/S0956796815000179. https://www.cambridge.org/core/product/identifier/S0956796815000179/type/journal_article.

Trivodaliev, Kire, Biljana Risteska Stojkoska, Marija Mihova, Mile Jovanov ja Slobodan Kalajdziski. 2017. “Teaching computer programming: The macedonian case study of functional programming”. Teoksessa *2017 IEEE Global Engineering Education Conference (EDUCON),* 1282–1289. ISSN: 2165-9567. Huhtikuu. https://doi.org/10.1109/EDUCON.2017.7943013.