Janne Isoaho

Hahmoperusteinen ohjelmointikieli JSON-prosessointiin

Tietotekniikan pro gradu -tutkielma

16. toukokuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Janne Isoaho

Yhteystiedot: janne.o.isoaho@gmail.com

Ohjaajat: Ville Tirronen ja Paavo Nieminen

Työn nimi: Hahmoperusteinen ohjelmointikieli JSON-prosessointiin

Title in English: Pattern-oriented programming language for JSON processing

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmistotekniikka

Sivumäärä: 45+18

Tiivistelmä: Tutkielmassa tarkastellaan hahmoperusteista ohjelmointia ja json-prosessontia. Tutkielmassa kehitetään vaihtoehtoinen ohjelmointikieli Json-kyselyihin hyödyntäen hahmoperusteista ohjelmointia. Kehitetyn ohjelmointikielen käytettävyyttä evaluoidaan kyselyllä sekä prototyypillä suoritetulla käyttäjätestauksella.

Avainsanat: json, hahmo, hahmontunnistus, ohjelmointikieli, design science

Abstract: This thesis reviews pattern-match oriented programming and json processing. New programming language for Json-queries is developed and evaluated in this thesis. Evaluation is focused on usability of the language and done via survey and user testing on prototype.

Keywords: json, pattern, pattern-matching, programming language, design science

Taulukot

Sisällys

1 Johdanto

Tässä työssä tutkittiin hahmoperusteisen (engl. pattern-match-oriented) ohjelmointikielen käyttöä Json-prosessointiin. Työssä kehitettiin Design science -metodilla artefakti, joka noudattaa paradigmaltaan hahmoperusteista ohjelmointia. Design science -metodologian mukaisesti artefaktia evaluoitiin pitkin kehitystä ja sen perusteella kohdistettiin kehityksen suuntaa (Hevner ym. 2004). Tämän artefaktin tapauksessa erityisesti syntaksi on ollut evaluoinnin kohteena.

Työn tarkoituksena oli luoda helppokäyttöinen artefakti Json-prosessointiin. Nykyiset työkalut kuten jq, ovat kattavia, mutta esimerkiksi juuri jq:n käytettävyydestä ei löytynyt akateemista kirjallisuutta, eikä sen kotisivulla[1] ole edes mainintaa käytettävyys näkökulmasta. Tästä syystä koettiin tarpeelliseksi kehittää työkalu, jonka käytettävyyttä arvioidaan jo kehitysvaiheessa eri keinoin. Tässä työssä kehitettävässä artefaktissa panostettiin erityisesti sen selkeään ja helposti opittavaan käyttöön. Hahmoperusteinen ohjelmointikieli voi olla intuitiivinen monimutkaisille ongelmille (Egi ja Nishiwaki 2020). Tehokkuutta suorituskykymielessä tässä työssä ei haettu. Tilanteet joihin tätä artefaktia kehitetään eivät ole suorituskyky sensitiivisiä. Eräs tämän työn inspiraatioista on hahmoperusteinen ohjelmointikieli Egison (Egi ja Nishiwaki 2018). Egison, toisin kuin tämän työn tavoite, on täysiverinen ohjelmointikieli.

Yksinkertaistuksena voidaan sanoa, että hahmoperusteisessa ohjelmoinnissa ongelmat siirretään yhtäsuuruusmerkin vasemmalle puolelle ja ratkaistaan siellä. Tästä seuraa, että varsinaiset lausekkeet ovat hyvin yksinkertaisia, usein jopa pelkkiä yksittäisen muuttujan palautuksia. Yksinkertaistamalla lausekkeita ja ongelmien esitystapaa saadaan helpommin ymmärrettävä ohjelmointikieli ja samalla uusi yksinkertaisempi työkalu. Artefaktin tavoiteltu hyöty onkin sen helppokäyttöisyys ja yksinkertaisuus. Vastaava tavoite oli esimerkiksi artikkelissa (Broberg, Farre ja Svenningsson 2004), jossa Broberg ja muut lisäsivät Haskelliin säännöllisten lausekkeiden tuen.

Tutkimuskysymyksiä tässä työssä on kaksi:

1. https://stedolan.github.io/jq/

- TK1 Pystyykö Egisonin kaltaista hahmoperusteista kieltä saamaan käytettävämmäksi?
- TK2 Pystyykö kehittämään jq:ta paremman vaihtoehdon hahmoperusteisesti?

2 Hahmoperusteinen ohjelmointi

Hahmoperusteinen ohjelmointi (pattern-match-oriented programming) on ohjelmointiparadigma siinä, missä esimerkiksi olioperusteinen ohjelmointi ja funktio-ohjelmointikin, ohjelmointiparadigmoista lisää luvussa 2.2. Yksinkertaisesti hahmoperusteista ohjelmointia voisi kuvata ohjelmointina yhtäsuuruusmerkin vasemmalla puolella. Ongelmat jäsennetään yhtäsuuruusmerkin vasemmalle puolelle tilanteeseen sopiviksi hahmoiksi ja siten ongelman ratkaisu saadaan yksinkertaistettua yhtäsuuruusmerkin oikealle puolelle. Paradigmana hahmoperusteinen ohjelmointi on erittäin uusi ja sitä puhtaasti mahdollistavia kieliä ei juurikaan ole. Tämä ei kuitenkaan tarkoita, ettei hahmoperusteisia ominaisuuksia olisi ohjelmointikielissä, koska hahmonsovitusta löytyy nykyisin useista kielistä. Tälläisiä kieliä ovat muunmuasssa Haskell (Marlow ym. 2010) ja Scala (Odersky ym. 2004). Hahmonsovitusta eri kielissä esitelty luvussa 2.5.

2.1 Hahmoperusteinen ohjelmointi käsitteenä

Jotta hahmoperusteiseen ohjelmointiin voidaan syventyä, tulee ensiksi määritellä joitain käsitteitä. Aloitetaan hahmontunnistuksesta (engl. pattern matching).

Hahmoperusteisessa ohjelmoinnissa tunnistetaan datasta määriteltyä hahmoja (engl. pattern), joiden avulla pystytään vaikuttamaan ohjelman kulkuun ja/tai palautukseen. Hahmonsovitus voi olla tehokas työkalu siihen sopivissa tilanteissa. Esimerkkinä Fibonaccin lukujono kirjoitettuna Haskell-ohjelmointikielellä.

2.1: Yksinkertainen fibonacci-toteutus Haskell-ohjelmointikielellä

```
1 fibonacci :: Int -> Int
2 fibonacci 0 = 0
3 fibonacci 1 = 1
4 fibonacci n =  fibonacci(n - 1) + fibonacci(n - 2)
```

Esimerkissä 2.1 esitellään hahmonsovitus yksinkertaisimmillaan. Fibonacci-funktiota kutsuttaessa ohjelma tunnistaa parametrin arvon perusteella sopivan hahmon. Sovitettava hahmo on siis parametrina saatu numero, tässä tapauksessa ohjelma ottaa syötteenä kokonaislu-

kuja ja tunnistaa niistä kolme eri hahmoa:

- Syöte on luku 0
- Syöte on luku 1
- Syöte on muu luku

Kaikille tilanteille määritellään sitten oma toimintansa. Vaikka esimerkissä 2.1 tilanne on hyvin yksinkertainen ja hahmo sovitetaan yksinkertaisia numeroarvoja vasten, voi sovitettava hahmo olla paljon monimutkaisempikin. Erilaisia hahmoja on kirjallisuudessakin määritelty useita. Tyypillistä on tarve voida tehdä hahmonsovitusta erilaisille tietotyypeille (engl. data type). Eräs mielenkiintoinen joukko tietotyyppejä ovat ei-vapaat tietotyypit (engl. non-free data types). Ei-vapaita tietotyyppejä ovat sellaiset tietotyypit joilla ei ole kanonista muotoa, esimerkiksi multijoukot (engl. multiset) Esimerkiksi multijoukot 1, 2, 3 ja 3, 2, 1 ovat eri esityksiä, muotoja, samasta datasta, mutta silti yhdenpitäviä.

2.2 Hahmoperusteinen ohjelmointi ohjelmointiparadigmana

Hahmoperusteinen ohjelmointi ei ole kovin tunnettu ohjelmointiparadigma. Jotta voidaan käsitellä hahmoperusteista ohjelmointia ohjelmointiparadigmana täytyy ensin käsitellä ohjelmointiparadigmaa käsitteenä ja sitä hyödyntäen kuvataan hahmoperusteista ohjelmointia ohjelmointiparadigmana, verrattuna tunnetumpiin ohjelmointiparadigmoihin.

Ohjelmointiparadigmoja on useita. Selvyyden vuoksi paradigmalla tässä työssä tarkoitetaan aina ohjelmointiparadigmaa (engl. programming paradigm), ellei erikseen muuta mainita. Ohjelmointiparadigmalle löytyy varmasti useita määritelmiä, monet luultavasti samoja, mutta osa saattaa erota merkittävästikin. Eräs tähän työhön sopiva määritelmä olisikin seuraava: "Ohjelmointiparadigma on lähestymistapa tietokoneen ohjelmointiin, joka perustuu joko matemaattiseen teoriaan tai koherenttiin joukkoon periaatteita"(Van Roy ym. 2009).

Edellinen määritelmä, vaikka tähän työhön sopiva onkin, on otettava kriittisesti arvosteluun, onko se järkevä määritelmä ja toisaalta tulee pohtia onko syytä lukittautua paradigman käsitteeseen. Krishnamurthi (2008) kyseenalaistaa ohjelmointiparadigmojen hyödyllisyyden modernissa ohjelmoinnin opetuksessa ja väittää ettei taksonomisuus ole tieteen hengen mukais-

ta. Tästä syystä otetaan tarkasteluun ohjelmointiparadigmojen kehittymisen syyt. Simmonds (2012) esittää, että ohjelmointiparadigmat kehittyivät tarpeesta jakaa ja hallinnoida ohjelmakehitystä. Tässä työssä se näkyy siten, että ongelma jaetaan hahmoihin ja sitä pyritään hallinnoimaan hahmonsovituksella.

Hahmoperusteisessa ohjelmoinnissa yhdistyvät sekä matemaattinen teoria, että joukko periaatteita. Hahmot, hahmonsovitus, joukko-oppi sekä muut vastaavat ovat vahvasti matemaattisia käsitteitä joiden pohja on paljon syvemmällä matematiikassa, kun tämän työn puitteissa on syytä käsitellä. Matemaattisesta pohjasta huolimatta ohjelmointiparadigmana hahmoperusteista ohjelmointia esitetään Egi ja Nishiwaki (2020) toimesta vasta 2010-luvulla.

2.3 Hahmoperusteisen ohjelmointikielen vaatimukset

On mahdotonta luotettavasti tietää kuinka monta ohjelmointikieltä maailmassa on. Kun joukkoon lasketaan kaikki eri tasoiset ja eri syihin kehityt ohjelmointikielet voitaneen olettaa, että suuri osa niistä on turhia. Siksi on syytä pohtia millainen ohjelmointikielen tulee olla, jotta se ei kasvattaisi tätä turhien ohjelmointikielten listaa entisestään. Egi ja Nishiwaki (2018) asettavat kolme kriteeriä sille, että hahmoperusteinen ohjelmointi olisi käyttökelpoinen.

Ensimmäinen kriteeri on, että sen peruuttavan etsinnän algoritmin ei-lineaarisille hahmoille (engl. Efficiency of the backtracking algorithm for non-linear patterns) on oltava tehokas. Ei-lineaariset hahmot (engl. non-linear patterns) ovat sellaisia hahmoja, jotka sallivat saman muuttujan esiintyvän useamman kerran samassa hahmossa.

Toinen kriteeri on, että hahmojen tulee olla laajennettavissa (engl. "Extensibility of pattern matching") (Egi ja Nishiwaki 2018). Egi perustelee, tätä sillä, ettei ohjelmointikielen kehittämisvaiheessa voida huomioida kaikkia mahdollisia datan muotoja, vaan käyttäjän on voitava lisätä, muokata ja laajentaa niitä tarpeisiinsa sopivaksi.

Kolmas kriteeri on polymorfismi hahmoissa (engl. polymorphism in patterns). Vaikka polymorfismi usein liitetään olioperusteiseen ohjelmointiin tässä Egi ja Nishiwaki (2018) nostavat esille termin polymorfiset hahmot (engl. polymorphic patterns) tarkoittaen, että hahmoja ei tarvi muodostaa erikseen jokaiselle tietotyypille, jolla on sama arvo vaan ohjelmointikieli

osaa hoitaa sen.

Tässä vaiheessa tämän työn kannalta tulee huomata, että Egi ja Nishiwaki (2018) asettavat nämä kriteerit käytännölliselle yleiskäyttöiselle ohjelmointikielelle, (engl. general-purpose language, GPL). Tämän työn tarkoituksena on kehittää kieli, Jia, jonka käyttötarkoitus on rajattu Json-datan prosessointiin. Näin ollen tässä kehitettävän artefaktin tapauksessa kyseessä on hyvin domain spesifi (engl. domain-specific language, DSL). Tästä syystä on oleellista käydä jokainen Egin asettamista kriteereistä läpi ja pohtia sen järkevyyttä käsillä olevassa käyttötapauksessa.

Jian käyttötarkoitus tulee olemaan luoda ohjelmia, joita ajetaan harvakseltaan, joskus jopa vain kertaalleen. Tämän perusteella voidaan poistaa listalta Egin ensimmäinen vaatimus peruutus-algoritmin tehokkuudesta. Luonnollisesti on käyttäjäkokemuksen kannalta oleellista, että ohjelman suoritus ei kestä järjettömän pitkään, mutta mennään oletuksella, että pienet viiveet ovat hyväksyttäviä. Huomiodaan myös, että tässä työssä kehitettävä kieli tulkataan suoraan Haskellilla, kääntämättä sitä binääriseen muotoon tai tavukoodiksi ja sillä on vaikutuksia tehokkuuteen. Artefaktin toimintaa kuvattu tarkemmin luvussa 5

Egin asettama kriteeri hahmojen laajennettavuudesta ei myöskään päde DSL-kielessä, sillä jo suunnitteluvaiheessa on tiedossa, että data on aina Json-muotoista. Tästä johtuen voidaan sivuuttaa myös tämä kriteeri. Samaan kategoriaan menee myös kolmas kriteeri, jossa kyse on jälleen hahmojen muodosta kun kyseessä on useita tietotyyppejä. Näin ollen sivuutetaan kaikki Egin kriteerit tämän työn puitteissa johtuen, kehitettävän artefaktin domain spesifistä luonteesta.

2.4 Hahmoperusteinen ohjelmointikieli Egison

Tätä työtä innoittanut Egison on hahmoperusteinen ohjelmointikieli, joka pyrkii tekemään algoritmien esittämisestä intuitiivista. [1] Egisonissa hahmonsovituksen ollessa pääparadigma on siihen kiinnitetty huomiota ja annetaankin kielen käyttäjälle useita erilaisia valmiita hahmoja hyödynnettäväksi. Näitä erilaisia hahmotyyppejä esitellään Egisonissa eri käyttötarkoituksiin. (Egi ja Nishiwaki 2020)

1. https://www.egison.org/

Egison ohjelmat voidaan kirjoittaa hyvin tiiviisti ja ne ovat erittäin ilmaisuvoimaisia. Seuraavassa Egisonin kotisivulta[2] otettu esimerkki, koodista jolla haetaan alkulukujen joukosta sellaiset alkiot, jotka ovat keskinäinen erotus on 2.

2.2: Alkulukuparit Egisonilla

```
1 def twinPrimes :=
2   matchAll primes as list integer with
3     | _ ++ $p :: #(p + 2) :: _ -> (p, p + 2)
```

Esimerkistä huomataan, että Egisonissa on hyvin erilainen syntaksi kuin esimerkiksi tässä työssä kehitettävässä Jiassa. Perusidea koodissa on määritellä ensiksi matchAll-lauseke, joka ottaa kohteen, tässä **primes (alkulukujen joukko)**, hahmosovittimen, tässä **list integer**, ja hahmonsovituslauseen, tässä **_ ++ $p :: #(p + 2) :: _ -> (p, p + 2)** Hahmonsovituslause puolestaan koostuu hahmosta, tässä **_ ++ $p :: #(p + 2) :: _**, ja rungosta (engl. body), tässä **(p, p + 2)**. Hahmossa sovitetaan kaikki sellaiset alkio parit, joiden keskinäinen erotus on 2 ja joita ennen ja joiden jälkeen on mikä tahansa alkio, myös sen puuttuminen sallitaan. Toimintalogiikka Egisonissa on sellainen, että jos hahmonsovitus onnistuu niin silloin evaluoidaan hahmonsovituslauseen runko.

Egisonissa on määritelty seuraavat hahmot[3]:

- **Villikortti hahmo** (engl. wildcard pattern): Villikortti hahmo tunnistaa minkä tahansa merkkijonon sopivaksi hahmoksi.
- **Hahmomuuttuja** (engl .pattern variable): Hahmontunnistuksessa sidottu muuttuja
- **Indeksoitu hahmomuuttuja** (engl. indexed pattern variable): Nimensä mukaisesti hahmomuuttujan erikoistapaus, jossa muuttujat asetetaan hash map -tietorakenteeseen, joka sidotaan muuttujaan
- **Induktiivinen hahmo** (engl. inductive pattern): Induktiivinen hahmo koostuu Egisonista hahmokonstruktorista (engl. pattern constructor) ja sille annettavista parametrihahmoista
- **Arvohahmo** (engl. value pattern): Arvohahmo tunnistetaan jos sen arvo evaluoituu vertailtavan hahmon mukaisesti

2. https://www.egison.org/demonstrations/primes.html
3. https://egison.readthedocs.io/en/latest/reference/pattern-matching.html#patterns

- **Predikaattihahmo** (engl. predicate pattern): Predikaattihahmo tunnistetaan jos sen luoma ehto täyttyy
- **Ja-hahmo** (engl. and-pattern): Ja-hahmo on Boolen algebraan perustuvaa loogista JA-operaattoria vastaava hahmo
- **Tai-hahmo** (engl. or-pattern): Tai-hahmo on Boolen algebraan perustuvaa loogista TAI-operaattoria vastaava hahmo
- **Ei-hahmo** (engl. not-pattern): Ei-hahmo on Boolen algebraan perustuvaa loogista EI-operaattoria vastaava hahmo
- **Peräkkäishahmo** (engl. sequential pattern): Peräkkäishahmot mahdollistavat hahmontunnistuksen järjestyksen suunnan muuttamisen[4]
- **Toistohahmo** (engl. loop pattern): Toistohahmoilla kuvataan toistuvia hahmoja ja se vastaan säännöllisten lausekkeiden Kleenen tähti -operaattoria[5]
- **Olkoon-hahmo** (engl. let pattern): Olkoon-hahmo mahdollistaa muuttujien sitomisen toisen hahmon sisällä

2.5 Hahmonsovitus eri kielissä

Hahmonsovituksen käyttökelpoisuudesta kertoo se, että vastaavia ominaisuuksia on esitetty ja osittain otettu jo käyttöön myös niin kutsutuissa valtakielissä. [6] Tässä luvussa esitellään hahmonsovitusta Java-, Python- ja C++-ohjelmointikielissä. Kaikki näistä kolmesta kielestä ovat erittäin suosittuja [7]

Java on pitkäikäinen ohjelmointikieli, joka on vuosien varrella saanut useita uusia ominaisuuksia, joista osan käyttö on vakiintunut kun taas toisien ei. (Parnin, Bird ja Murphy-Hill 2011) Ei siis ole yllättävää, että hahmonsovitus on eräs Javan 14 versioon esitellyistä ominaisuuksista [8]

Kaaviossa 2.3 olevassa kaaviossa kuvattu uusia pattern-match ominaisuuksia Java 14:ssa.

4. https://egison.readthedocs.io/en/latest/tutorial/quick-tour.html#sequential-patterns
5. https://egison.readthedocs.io/en/latest/tutorial/quick-tour.html#loop-patterns
6. https://spectrum.ieee.org/at-work/tech-careers/top-programming-language-2020
7. https://insights.stackoverflow.com/survey/2020
8. https://openjdk.java.net/jeps/305

2.3: Hahmonsovitus Javassa

```
if (ohjelmointiKieli instanceof Java java) {
    teeJavaPatternMatch();
} else if(ohjelmointikieli instanceof Jia jia) {
    teeHienoJiaPatternMatch();
}
```

instanceof -avainsana ei ole uusi Javassa, mutta sen hyödyntäminen suoraan olioviitteen saamiseen (ylläolevassa esimerkissä *..instanceof Java java* sekä *..instanceof Jia jia* käyttö poistaa tarpeen erikseen sovittaa olion tyyppi yhteisestä ylemmästä abstraktiosta.

Vastaavasti C++:n 23. versioon on esitetty Michael Parkin kirjastoon C++ 17:lle kehittämään kirjastoon [9] perustuvaa hahmonsovitus mekanismia [10]. Yksinkertainen syntaksi esimerkki Parkin ehdotuksesta kaaviossa 2.4 Vastaavaa on esitetty C++:n jo aikaisemminkin (Solodkyy, Dos Reis ja Stroustrup 2013)

2.4: Hahmonsovitus C++:ssa

```
void hahmonSovitusEsimerkki() {

  using namespace mpark::patterns;

  for (int i = 1; i <= 10; ++i) {
    match(i % 2)(
        pattern(0) = [] { std::printf("parillinen\n"); }
        pattern(1) = [] { std:printf("pariton"\n"); }
  }
}
```

Myös Python-ohjelmointikieleen on esitetty hahmonsovitus mekanismia. Alunperin esitetty PEP-622 [11] korvattiin PEP-634:lla [12] Kaaviossa 2.5 syntaksi esimerkki PEP-634:n mukaisesta hahmonsovituksesta Pythonissa.

9. https://github.com/mpark/patterns
10. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p1371r2.pdf
11. https://www.python.org/dev/peps/pep-0622/
12. https://www.python.org/dev/peps/pep-0634/

2.5: Hahmonsovitus Pythonissa

```
def hahmonSovitusEsimerkki(hahmo) {
    match hahmo%2:
        case 0:
            print("Parillinen")
        case 1:
            print("Pariton")
}
```

Tässä esiteltyjen esimerkkien pohjalta voinee olettaa, että hahmoperusteiselle ohjelmoinnille on sijaa myös akateemisen kirjallisuuden ulkopuolella, vaikka tuleekin pitää mielessä, että nämä eivät vielä ole vakiintuneita ominaisuuksia vaan pääasiassa eri kieliin lisättäväksi esitettyjä ominaisuuksia.

2.6 Hahmoperusteisen ohjelmoinnin etuja

Hahmoperusteisessa ohjelmoinnissa algoritmien esittäminen on intuitiivisempaa (Egi ja Nishiwaki 2020). Egin mukaan samalla on käytännöllisempää toteuttaa monimutkaisia ohjelmia.

- **Monimutkaiset rakenteet voidaan esittää yksinkertaisemmin**
  - Rakenteet pystytään esittämään mallintaen datan rakennetta. Esimerkiksi tässä työssä kehitetty artefakti esittää käsiteltävän Json-datan mahdollisimman samalla tavoin kun se on varsinaisesssa Jsonissa kuvattu.
- **Kääntäjä pystyy kertomaan puuttuvista osuuksista on jätetty pois**
  - Esimerkiksi Haskellin hahmontunnistuksessa kääntäjä pystyy suoraan kertomaan puuttuvat skenaariot. Tämä puolestaan poistaa virheitä ajoympäristöstä.
- **Vastaavan toiminnallisuuden implementointi muuten on työlästä ja joissain tilanteissa hankalaa**
  - Hahmoperusteiset kielet kuten Egison voivat olla erittäin ilmaisuvoimaisia, jolloin monimutkaisten asioiden tekeminen helpottuu.

2.7 Hahmoperusteisen ohjelmoinnin hankaluuksia

Hahmoperusteinen ohjelmointi on vielä kehitysasteella ja siinä on tullut eri implementaatioissa erilaajuisia haasteita. Tässä nostettiin esiin joitain niistä.

- **Marginaalinen paradigma**
  - Kenties isoin ongelma hahmoperusteisen ohjelmonnin levittämisessä on sen tämän hetkeinen marginaalinen käyttö. Luvussa 2.5 esiteltiin, että käytössä hahmoperusteisia ominaisuuksia on esitelty moniin yleisimmästä ohjelmointikielistäkin, mutta ne ovat vielä joko esittely-asteella tai hyvin tuoreita.
- **Harvan kielen tukema**
  - Hahmoperusteinen ohjelmointi ei ole edes mahdollista useimmissa kielissä, joissa esimerkiksi olioperusteinen- tai funktionaalinen ohjelmointi on mahdollista. Hahmontunnistukseen perustuvia ominaisuuksia on kuitenkin tulossa moniin kieliin kuten luvussa 2.5 osoitetaan.
- **Oppimiskäyrä**
  - Hahmoperusteinen ohjelmointi on ainakin osittain erilainen tapa ohjelmoida tällä hetkellä vallitseviin paradigmoihin nähden siksi sen oppiminen on ei-triviaali tehtävä.

3 Json

Json (JavaScript Object Notation), tästä eteenpäin json, on kevyt datan enkoodaus formaatti. Json on ihmiselle helposti ymmärrettävä, mutta silti myös tietokoneelle tehokkaasti käsiteltävä. [1] Jsonin perusidea on esittää dataa kuten JavaScript-ohjelmointikielessä esitetään objekteja. Json koostuu pilkulla eritellyistä jäsenistä, joista jokainen voi olla yksi Jsonin tietotyypeistä. Jsonin tietotyypit kuvattu luvussa 3.1 ja esimerkkejä Json rakenteista ohjelmakoodilohkoissa 3.1, 3.2 ja 3.3. Kuten esimerkeissäkin näkyy Json-objektit on ympäröity aaltosulkeilla.

3.1 Jsonin tietotyypit

Jsonissa on olennaisesti kuusi primitiivistä tietotyyppiä:

- Numero (engl. Number)
- Merkkijono (engl. String)
- Totuusarvo (engl. Boolean)
- Null-arvo (engl. Null)
- Taulukko (engl. Array)
- Objekti (engl. Object)

Jsonin numero-tietotyypin arvot ovat tarkemmin ottaen joko kokonaislukuja (engl. integer) tai liukulukuja (engl. floating point number, float). Merkkijonot, totuusarvot ja null-arvot taas toimivat kuten voi olettaa muiden ohjelmointikielten perusteella. Merkkijonot ovat lainausmerkeillä eroteltuja jonoja merkkejä. Totuusarvot ovat joko true tai false ja null-arvolla kuvataan varsinaisen arvon puuttumista.

Taulukko puolestaan on tapa koota arvoja, kuten esimerkiksi esimerkissä 3.1 on esitelty yksinkertainen taulukko merkkijonoja.

1. https://www.json.org/json-en.html

3.1: JSON-taulukko esimerkki

```
{
    [
        "kissa",
        "koira",
        "hamsteri"
    ]
}
```

Objektit puolestaan yleisiä Json-ilmentymiä. Ne voivat sisältää useita avain-arvo pareja, taulukoita, lisää objekteja, tai mitä tahansa muuta Jsonin sallittuja jäseniä. Esimerkissä 3.2 esimerkki yksinkertaisesta Json-objektista, joka sisältää muita tietotyyppejä.

3.2: Json-objekti esimerkki

```
{
    "jsonObjekti" : {
        "numeroEsimerkki": 3,
        "merkkijonoEsimerkki" : "merkkijono",
        "totuusarvoEsimerkki" : true,
        "null-esimerkki" : null,
        "taulukkoEsimerkki": [
            1,
            2
        ]
    }
}
```

Esimerkistä 3.2 huomataan myös, että taulukkojen sisältämien arvojen tyyppi ei ole sidottu merkkijonoksi, vaan voi olla myös muutakin kuten tässä on numeroita.

3.2 Json verrattuna muihin dataformaatteihin

Severance (2012) kertoo, että Json on noussut maan alta käytetyimpien joukkoon. Hän puhuu xml:n ja jsonin vastakkainasettelusta ja pitää siinä mielessä jsonia parempana, että se esittää datan sellaisena kuin ohjelmointikielet sitä tulkkaavat.

Json on myös kevyempi formaatti ja samaa tietoa ei tarvita toistaa, esimerkiksi xml:n tagmäärittelyt ja sulut. Alla esimerkit miten sama tieto esitetään sekä Json- että xml-muodossa.

3.3: Json-esimerkki

```
{
    "AnkkalinnanAsukas": {
        "name": "Aku Ankka",
        "address": "Paratiisitie 13",
        "huollettavat": [
            "Tupu",
            "Hupu",
            "Lupu"
        ]
    }
}
```

3.4: XML-esimerkki

```
<AnkkalinnanAsukas>
    <name>Aku Ankka</name>
    <address>Paratiisi</address>
    <huollettavat>
        <Tupu/>
        <Hupu/>
        <Lupu/>
    </huollettavat>
</AnkkalinnanAsukas>
```

Näin yksinkertaisessakin esimerkissä huomataan että XML:ssä on hieman enemmän merkkejä kuin Json-versiossa. Tämä yksinään ei toki ole merkki paremmuudesta, vaan otetaan pintapuoliseen tarkasteluun myös alan tutkimus sekä vapaamuotoisemmat kirjoitukset aiheesta.

Tutkimuksia joissa vertaillaan Jsonin ja XML:ää on tehty useita. Google Scholarin mukaan, erilaisilla yhdistelmillä hakusanoista "json", "xml", "versus", "performance"ja "comparison"eniten viittauksia saanut artikkeli on Nurseitov ym. (2009), jossa tutkijat tekivät tapaustutkimuksen, jossa vertailtiin siirtonopeuksia ja työaseman resurssien käyttöä. Tutkimuksen tuloksena oli, että Json on merkittävästi nopeampi kuin xml tutkimukseen valituissa tilanteissa.

Jsonin voittokulusta kertoo myös se miten suosittu se on. Taulukkoon 1 on koottu StackO-

verflow:sta[2] eri esittämistapoihin liittyvien kysymyksien määriä. Taulukko on koottu käymällä läpi StackOverflow:n 3600 (ensimmäiset 100 sivua) tagia, joista on ensiksi karsittu pois erilaisilla merkkijono-hauilla tagin kuvauksesta ne jotka liittyvät eri formaatteihin ja sen jälkeen jätetty jäljelle vain ne, jotka ovat ihmisellekin ymmärrettäviä datan esitysmuotoja. Luonnolisesti tätä ei voi suoraan rinnastaa käyttömääriin, mutta se antaa kohtuullisen kuvan skaalasta miten paljon kutakin käytetään.

Taulukko 1: Merkintätapoihin liittyvien kysymysten määrä StackOverflowssa 20.8.2021

Haettu tag	**Kysymysten määrä**
JSON	324,071
XML	203,688
CSV	78,946
YAML	11,226

Taulukon 1 perusteella käyttömäärissä ei ole mielekästä verrata kuin Jsonia ja XML:ää, koska muut ovat suhteessa niin paljon vähemmän käytettyjä, jos hyödynnetään tätä metriikkaa.

Tarkoituksena tässä työssä ei ole todistaa, että json on parempi tai tehokkaampi, eikä siihen näin kevyin perustein ole pohjaa, kuin muut dataformaatit vaan näyttää, että se on relevantti kohde tämänkaltaisen työkalun kehittämiseen. Näillä perusteilla ja näiden lähteiden mukaan voidaankin todeta, että json on paljon käytetty ja relevantti dataformaatti.

3.3 Jq

Jq on kotisivunsa mukaisesti sed[3] json-datalle[4]. Jq:lla pystyy kyselemään ja muokkaamaan dataa laajasti. Jq:lla on kattava dokumentaatio[5] ja paljon erilaisia käyttötapoja.

Jq, kuten edellä mainittu sed, toimii pääasiassa komentorivikäyttöliittymästä johon parametrina syötetään annettu jq-ohjelma, filtteri, kuten jq:n dokumentaatiossa usein kutsutaan, ja

2. https://stackoverflow.com/tags
3. https://www.gnu.org/software/sed/manual/sed.html
4. https://stedolan.github.io/jq/
5. https://stedolan.github.io/jq/manual/

syöte json joko putkitettuna toisesta komennosta tai tiedostosta luettuna. Esimerkki jq:n käytöstä kuviossa 3.5. Esimerkissä jq:lla haetaan elementin "name", jolla on ylätason elementti "AnkkalinnanAsukas", arvo. Kun syötteenä kuvion 3.5 filtterille annetaan kuvion 3.3 json-tiedosto saadaan tuloksena "Aku Ankka"

3.5: Jq-esimerkki

```
1 $ jq '.AnkkalinnanAsukas.name' json-esimerkki.json
```

Jq:n käytettävyydestä ei löytynyt Google Scholarista hakusanoilla "jq", "jq json", "jq usability"yhtäkään varteenotettavaa lähdettä, jossa olisi käsitelty sen käytettävyyttä. Myöskään yleisesti jq:sta ei löytynyt akateemista kirjallisuutta näillä hakuehdoilla, poislukien muutama maininta Json-aiheisissa oppikirjoissa.

4 Artefaktin kehitys

Työssä kehitettiin artefaktia suunnittelutiede-metodologialla (engl. design science). Suunnittelutieteessä oleellista on artefaktin evaluointi osana kehittämisprosessia. (Hevner ym. 2004)

4.1 Design Science

Design sciencessa on tarkoituksena kehittää artefaktia ja evaluoida sen kehityksen suuntaa ja tehdä ratkaisuja evaluoinnin tuloksien perusteella.(Hevner ym. 2004) Tämän työn asetelma oli herkullinen Design Sciencen näkulmasta, koska tarkoituksena oli kehittää uusi työkalu korvaamaan jo olemassaolevia vastaavia työkaluja, näin ollen pääsimme kehittämään artefaktia jolle on selkeä vertailukohta evaluointiin, Vertailukohtana toimivat olemassaolevat jq ja Egison.

Tässä työssä tehtiin evaluointia kahteen kertaan, alkupuolella ja loppuvaiheessa toimivan prototyypin avulla. Alkupuolen evaluointi tehtiin syntaksin perusteella verraten jo olemassa oleviin työkaluihin. Loppuvaiheen evaluoinnissa puolestaan ideana oli, että artefaktin prototyyppiä käytettiin tehtävien ratkomiseen. Tällöin jälkimmäisessä evaluoinnissa voitiin arvioida näiden käyttäjien kokemuksia perusteella artefaktista ja saada tietoa sen mahdollisen jatkokehityksen avuksi.

Eräs vaihtoehto Jian arvioimiseen olisi ollut tutkia, saako se käyttäjäkuntaa. Sitä, ei kuitenkaan voitu järkevästi toteuttaa Pro gradu -tutkielman mittakaavassa, vaan jouduttiin tyytymään käytettyihin kysely-tutkimuksiin.

4.2 Syntaksin kehitys

Syntaksin kehitys Jialle oli haastavaa. Tarkoituksena on kehittää mahdollisimman hyvä ja erityisesti helppokäyttöinen työkalu. Jian kohderyhmä on henkilöt, jotka ovat jollain tasolla ohjelmointitaitoisia, joten syntaksin tulisi olla olemasasolevien työkalujen kaltainen siinä määrin kun mahdollista. Tällä helpotettaisiin Jian käyttöönottoa. Tässä on huomioitava, että se toisaalta sotii tutkimustietoa vastaan sillä Stefikin (2013) tutkimuksen mukaan C-

sukuisten kielten syntaksi ei ole vasta-alkajille satunnaistettuja avainsanoja parempi.

Tässä tutkimuksessa painopiste pidetään käytettävyydellä ja erityisesti sellaisille käyttäjille joilla on ohjelmointitaustaa. Tästä johtuen joitain ratkaisuja tehtiin prioriteettina yleinen konventio muiden kriteerien yläpuolella.

Usein ohjelmointikielien kehityksessä ei oikeastaan ole käytetty kaikkein selkeimpiä tai intuitiivisimpia avainsanoja jos henkilöllä ei ole ohjelmointitaustaa. (Stefik ja Gellenbeck 2011) Siksi tähänkin työhön liittyvissä evaluoinneissa kohderyhmänä oli ohjelmointitaustaisia henkilöitä.

Eräs ainakin Microsoftin kehittämän C#-ohjelmointikielen käytettävyyden tutkimiseen käytettya viitekehys on "Cognitive Dimensions-viitekehys. (Clarke 2001). Tämä kuitenkin hylättiin ja päädyttiin käyttämään kyselyitä ja koehenkilöitä, jotta saatiin tilanteeseen spesifimpää palautetta. Tämän hyödyntäminen nähtiin kuitenkin hankalaksi ja siitä luovuttiin ja jätettiin kyselyt evaluointimetodiksi.

4.3 Syntaksin tavoitteet

Artefakti on ideaalitilanteessa käytettävyydeltään intuitiivinen ja mahdollisimman helposti opittava. Nämä kaksi kriteeriä ovat lähellä toisiaan, sillä jos kieli toimii niinkuin sen olettaisi on se intuitiivinen ja tällöin myös helposti opittava. Kuvattuun ideaalitilanteeseen pääseminen vaatii kieleltä ja sen kieliopilta monia asia asioita. Oppimisen kannalta virheiden korjaaminen on eräs olennaisimmista toiminnoista.

Jian kehityksessä on Design Sciencen hengessä pyritty tekemään asioita askel kerrallaan toimivaksi. Alusta asti on ollut tiedossa paljon ominaisuuksia joita haluttaisiin sisällyttää Jian, mutta rajauksia on tehtävä ja evaluointimielessä koettiin järkeväksi tehdä vain tietyt asiat kerrallaan loppuun asti. Kehityksessä siis tarkoituksellisesti on pyritty enemmän syklimäiseen, hieman ketteriä menetelmiä [1] muistuttavaan malliin, jossa saataisiin aluksi toimiva versio tarkoituksellisesti pienemmillä ominaisuuksilla jota voi sitten jatkokehittää järkevästi design scienceen kuuluvan evaluointi syklin perusteella. Tämän lähestymistavan tavoiteltu etu on

1. https://agilemanifesto.org/iso/fi/manifesto.html

se, ettei evaluonnin tuloksia tulkittaisi väärin.

Näillä perustein artefaktin kehittämiseen valikoitui jo alkuvaiheessa kaksi evaluointikierrosta. Ensimmäinen evaluointi suoritettiin puhtaalla tekstipohjaisella kyselyllä syntaksille vertaillen Jiaa jq:hun, olemassaolevaan työkaluun joka sen tulisi korvata. Tällä kierroksella järjestettiin Webpropol-kysely, joka on nähtävissä liitteenä B Toinen evaluointikierros järjestettiin kun pienin mahdollinen toimiva versio artefaktista oli kehitetty. Toisella evaluointikierrokselle koehenkilöt ratkoivat json-tiedonhaku pulmia hyödyntäen Jiaa. Tutkimushenkilöille jaettu ohjeistus liitteenä C

4.4 Syntaksin evaluointi

Ohjelmointikielet eivät perinteisesti ole olleet erityisen tutkimustietoon perustuvia (Stefik ym. 2014). Tämän työn tarkoituksena ei ollut kehittää Stefikin tutkimuksen mukaista ohjelmointikieltä, vaan vaihtoehtoinen työkalu jq:lle hyödyntäen pattern matching oriented -ohjelmointiparadigmaa, joten vaikka tässä onkin tavoitteena kehittää jq:ta käytettävämpi ohjelmointikieli, on menty enemmän konvention kuin tutkimustiedon perusteella.

Syntaksin kehityksessä mukailtiin luvussa 2.4 esiteltyjä Egisonin patterneja ja yleisiä käytänteitä. Ennen ensimmäistä evaluointikierrosta artefaktin kehityksessä alettiin kyseenalaistaa syntaksin toimivuutta erityisesti yksinkertaisilla kyseilyillä. Tästä syntyi lyhennesyntaksi, jonka tarkoituksena oli helpottaa lyhyiden tai muuten yksinkertaisten kyselyiden luontia ja samalla mahdollistaa kehitettävän kielen osaajalle parempi käyettävyys.

Evaluoinnin avuksi ei saatu paljoakaan tutkimustietoa, sillä esimerkiksi Stefikin tutkimuksessa on fokus ollut avainsanoissa, joita Jiassa ei ole lähes ollenkaan.

4.5 Virheiden merkitys syntaksissa

Intuitiivisuuden kannalta haluttiin Jian syntaksin kehityksessä tarkastella virheviestejä ja niiden merkitystä. Virheviestejä ovat tutkineet muun muassa Denny, Luxton-Reilly ja Tempero (2012), he tutkivat Java-syntaksivirheitä ohjelmoinnin opetuksessa. He huomasivat, että eniten virheitä tulee seuraavissa kategorioissa: "Cannot resolve identifier", "Type mismatch",

"Missing ;"ja "Token should be deleted". Nämä olivat myös kaikki korkealla siinä, miten kauan korjaus vei aikaa keskimäärin. Jiassa näitä ongelmia pyrittiin selättämään jo syntaksin suunnittelulla. Jiassa ei tarvitse puolipisteitä rivien tai lausekkeiden loppuun ja rakenne mukailee täysin Jsonia, jolloin rakenteen pitäisi olla luonnostaan koherentti syötedatan kanssa. Myös tyypityksen puolesta Jia käyttää puhtaasti Jsonin tietomallin mukaisia tyyppejä ja kaikki on aina esitettävissä sekä vertailtavissa merkkijonoina. "Cannot resolve identifier-syntaksivirheitä vastaavia virheitä toki on mahdollista tehdä jos on huolimaton muuttujien nimeämisessä tai käytössä.

Luxton-Reilly ja Petersen (2017) mukaan hankaluus ohjelmointitehtäviä tehdessä on siinä, että yksinkertainenkin tehtävä saattaa vaatia osaamista monesta asiasta. Tästä johtuen on hankala haarukoida mitä opiskelija osasi tai ei osannut. Tämä sama haaste tulee esiin kyselyssä. Kysymyksiä on toki pyritty jaottelemaan helpompiin ja mahdollisimman paljon vain yhtä asiaa mittaaviin. Tästä huolimatta vastaajan tulee osata yhdistää monia eri konsepteja ainakin voidakseen kertoa eri ohjelmien toiminnasta. Tähänkin Jiassa auttaa suuresti se, että kyseessä on DSL ja syntaksi mukailee pohjadataa.

Eräs haaste Jian kehityksessä oli sen ymmärrettävyys. Tavoitteena oli luoda mahdollisimman helposti ymmärrettävä kieli ja siksi haluttiin asettaa selkeät käytänteet ohjelmakoodin kirjoittamiselle. Solo ja Ehrlich (1984) mukaan Ohjelmointikäytänteiden vastainen ohjelmakoodi voi olla syntaktillisesti ja semanttisesti toimivaa koodia, mutta se ei ole kokeneelle lukijalle odotusten mukaista. Siksi Jian mukana pyrittiin luomaan selkeät käytänteet koodin kirjoittamiseen. Konventioita ei kuitenkaan haluttu pakottaa syntaksipuolella, näin ollen jättäen käyttäjälle päätöksen noudattaa tai olla noudattamatta suositeltuja ohjelmointikäytänteitä.

Kenties tärkein käytäntö oli muuttujien nimeäminen. Json on standardina, kuten luvussa 3 on esitelty, niin löyhä että avaimien ja arvojen nimeämistä ei ole rajattu lähes ollenkaan. Tämä aiheuttaa tilanteet, jossa Jia-ohjelmakoodista voi olla hankala erottaa muuttujia hahmoista. Binkley ym. 2009 mukaan kokemattomilla koehenkilöillä meni kauemman tunnistaa camel case -tyyppiset muuttujien nimet, mutta kaikki koehenkilöt olivat tarkempia tunnistamaan camel case -tyyppisesti nimetyt muuttujat. Tästä syystä Jian konventioiden mukaan käytetään camel case -tapaa muuttujien nimeämiseen.

4.6 Artefaktin kehityksen kulku

Artefaktin alkuperäinen tavoite oli lähes kokonaisuudessaan korvata jq Json-kyselyiden osalta. Tästä kuitenkin jouduttiin luopumaan tutkielman edetessä, johtuen liian laajasta ja kunnianhimoisesta tavoitteessa, suhteessa tutkielman laajuuteen. Luvussa 5 kuvataan, mitä kaikkia ominaisuuksia artefaktissa lopulta on, mutta tässä käsitellään kehityksen vaiheita ja siinä tehtyjä kompromissejä.

Kehityksessä tapahtuneet kompromissit keskittyivät Jian ominaisuuksiin, ei niinkään evaluointiin. Evaluonnit suoritettiin lähes alkuperäisen suunnitelman mukaisesti kahteen otteeseen kehityksen aikana. Jälkimmäisen evaluoinnin sisältöön luonnollisesti vaikutti Jian ominaisuudet, mutta perusperiaate pysyi alkuperäisen suunnitelman mukaisena.

Isoimmat kompromissit Jian ominaisuuksissa liittyivät vastauksen rakentamiseen, ehdollisuuksiin ja vertailuihin sekä listojen käsittelyyn. Vastauksen rakentaminen jq:lla on monipuolista ja sillä pystyy luomaan monenlaisia uusia json-malleja. Heti Jian kehityksen alussa todettiin, että Jiasta tulee tämän tutkielman puitteissa kyselykieli, jolla voi palauttaa vain yksinkertaisia rakenteita. Loppuvaiheessa päädyttiin siihen, että ainoastaan merkkijono palautukset onnistuvat. Kyselyissä erilaisten vertailujen tekeminen sekä ehdollisen palautuksen tekeminen olisi varmasti hyödyllistä, mutta se yhdessä Json-listojen käsittelyn kanssa jäi myöskin pois Jiasta tässä vaiheessa.

Yksi merkittävä oivallus kehityksen ja evaluointien aikana syntaktillisesti Jiaan tuli. Se oli lyhennesyntaksi, jossa siirretään niin sanottua boilerplate-koodia pois kirjoittajalta ja kehitettäisiin Jiaa pidemmälle, jotta se pystyisi itse päättelemään osuudet paremmin. Tätä uutta syntaksia vertailtiin ensimmäisessä kyselyssä. Sitä ei kuitenkaan ehditty Jiaan toteuttamaan muuta kuin syntaksin suunnittelun tasolla.

5 Valmis artefakti

Tässä luvussa esitellään Jia-ohjelmistoa ja sen eri ohjelma-komponentteja. Selkeyden vuoksi tässä luvussa Jialla tarkoitetaan ohjelmistoa, ei kieltä, ellei erikseen ole mainittu. Jian ohjelmakoodit löytyvät MIT-lisenssillä Githubista[1]. Käyttäjän näkökulmasta Jia on yksittäinen ohjelma, joka ottaa syötteenä käyttäjän kirjoittaman ohjelmakoodin ja Jsonin johon sitä sovelletaan (kuten liitteessä A on esitetty). Todellisuudessa kuitenkin Jia on merkittävästi monimutkaisempi ohjelmisto, kuten ohjelmistot yleensä ovat.

Ensimmäiseksi alaluvussa 5.1 esitellään toteutetut ja ehdotetut syntaksin Jialle. Alaluvussa 5.2 puolestaan kuvataan Jian komponentit ja käytetyt kirjastot. Ja lopuksi luvuissa 5.3 sekä 5.4 kuvataan miten Jiaa käytetään ja mitä jatkokehitys mahdollisuuksia sille jäi tämän tutkielman puitteissa.

5.1 Jian syntaksi

Jian syntaksi perustuu mahdollisimman pitkälti Jsonin syntaksiin, joka on esitelty luvussa 3. Pieniä eroavaisuuksia toki on. Jiassa ei ole tarpeen kuvata koko Jsonin rakennetta, vaan riittää vain oleelliset osuudet hahmontunnistamisen onnistumiseksi. Esimerkiksi jos Jiassa haluaa hakea seuraavasta Json-rakenteesta kaikki *etuNimi*-elementit

5.1: Json-esimerkki

```
{
    "etuNimi": "Aku",
    "sukuNimi": "Ankka",
    "osoite": {
        "katu": "Paratiisitie",
        "numero": 13

    }
}
```

Onnistuu se helposti seuraavalla Jia-ohjelmalla.

1. https://github.com/tartti/jia

5.2: Jia-esimerkki

```
1 {"etuNimi": x} <- input
2 return x
```

Vastaavasti jos halutaan hakea syvemmällä rakenteessa olevia arvoja onnistuu se esimerkiksi *numero*-elementin osalta seuraavasti

5.3: Jia-esimerkki

```
1 {"osoite": {"numero": x}} <- input
2 return x
```

Jian lyhennesyntaksissa puolestaan siirretään päättelyä ohjelmointikielen tulkille ja mahdollistetaan ilmaisuvoimaisempien ohjelmien kirjoitus. Lyhennesyntaksissa palautettavien muuttujien nimeäminen jätetään tekemättä ja poistetaan myös esimerkiksi nuolimerkki. Yksinkertaisimmillaan lyhennesyntaksilla kirjoitettu ohjelma voisi olla seuraavanlainen

5.4: Yksinkertainen Jia-lyhenne-esimerkki

```
1 {"etuNimi" : x} input
```

Esimerkkikoodissa 5.4 huomataan, että return-lausetta ei anneta enää eksplisiittisesti vaan lyhennesyntaksissa palautetaan automaattisesti seuraavan arvojärjestyksen mukaisesti arvokkain elementti.

1. Muuttujat
2. Syvin ei määritelty elementti
    - Esimerkki 1. *osoite : {katu}* palauttaa katu-elementin arvon sillä katu on syvimmän tason ei-määritelty elementti
    - Esimerkki 2. *osoite: {katu:"Paratiisitie"}* palauttaa koko osoite-elementin, sillä syvimmän tason ei-määritelty elementti on osoite ja katu on ehto sille

Tämäkään lähestymistapa ei ole mutkaton ja varsinkin syvimmän ei määritellyn elementin valitseminen loogisesti ja Jian käyttäjälle selkeästi vaati hiomsta. Kuitenkin tämä syntaksimalli koettiin hyväksi evaluoinneissa luvussa 6.

5.2 Jian arkkitehtuuri ja komponentit

Perusidealtaan Jia on rakennettu kuin mikä tahansa muu ohjelmointikielen kääntäjä tai tulkki. Se koostuu kääntäjän etu- ja takaosista. Etuosaan sisältyy selaaja ja jäsennin. Selaaja (engl. tokenizer) lukee merkkijonon ja pilkkoo sen sanasiksi (engl. lexical token). Jäsennin puolestaan ottaa syötteenä selaajan pilkkomat sanaset ja rakentaa niistä abstraktin jäsennyspuun (engl. abstract syntax tree, AST).

Kääntäjän takaosaan puolestaan kuuluu kohdekielen kanssa tekemisissä olevat komponentit. Jian tapauksessa kääntäjän takaosa onkin suora tulkki. Suora tulkki hyödyntää suoraan kohdekielen ominaisuuksia ohjelmakoodin ajamiseen eikä varsinaisesti luo erillistä käännettyä ohjelmaa (binääri tai tavukoodi muodossa).

Tehokkuuden näkökulmasta on huomioitavaa, että kun Jiassa käytetään kutsuttua suoraa tulkkia, joka helpottaa isäntäkielen (engl. host language) ominaisuuksien implementointia, on kuitenkin myös hitaampi vaihtoehto sillä ohjelma suoritetaan toisen kielen ajo ympäristössä eikä suoraan prosessorin päällä.

Jiassa käytetyt ohjelmakehykset ja kirjastot valikoituvat pääasiassa aikaisemman tuntemuksen ja tilanteeseen sopivuuden perusteella. Alusta asti oli selvää, että Jia toteutetaan Haskell-ohjelmointikielellä ja sen kirjastoilla. sanastimen ja jäsentimen toteutukseen hyödynnettiin Megaparsec-kirjastoa [2]. Sekä Haskellin, että Megaparsecin valinta olivat selkeitä, koska niiden käyttö koettiin helpoimmaksi sillä niistä oli jo entuudestaan osaamista.

Näiden lisäksi hyödynnettiin Aeson[3], Lens[4] sekä Lens-Aeson[5] kirjastoja json-käsittelyyn. Yksikkötestaukseen puolestaan hyödynnettiin Tasty[6]-kirjastoa. Nämä kirjastot valikoituivat Jian toteutukseen siitä syystä, että ne tarjosivat valmiiksi sellaisia asioita, jotka olisi muuten joutunut toteuttamaan itse, esimerkiksi json-käsittely.

2. https://hackage.haskell.org/package/megaparsec
3. https://hackage.haskell.org/package/aeson
4. https://hackage.haskell.org/package/lens
5. https://hackage.haskell.org/package/lens-aeson
6. https://hackage.haskell.org/package/tasty

5.3 Komentorivikäyttöliittymä

Komentorivikäyttöliittymällä (engl. command line interface, cli) tarkoitetaan tekstipohjaista käyttöliittymää jossa käyttäjä kommunikoi tietokoneen kanssa komentokehotteen avulla. Komentorivikäyttöliittymälle tyypillistä on tekstipohjainen ikkuna ilman graafisia elementtejä. Tämän luvun esimerkeissä komentokehotetta kuvataan $-merkillä ja sen jälkeen tuleva osuus on käyttäjän antama syöte. Rivin alussa oleva >-merkki puolestaan kuvaa sitä, että seuraava osuus on ohjelman tulostamaa outputtia.

Jia on suunniteltu käytettäväksi komentoriviltä, riippumatta siitä haluaako skriptin kirjoittaa erilliseen tiedostoon vai komentoon sisään. Komentojen perusrakenne on seuraava:

5.5: jia-komennon perusrakenne

```
1 $ jia script.jia input.json
2 > "esimerkkiOutput"
```

Toinen toiminto, joka jäi toteuttamatta oli, että käyttäjä antaa skriptin suoraan komentorivillä, ilman että kirjoittaa sitä teksti tiedostoon kuten 5.5 on tehty. Tämä oli suunniteltu esimerkin 5.6 mukaisesti.

5.6: jia-komennon perusrakenne

```
1 $ jia '{firstName: x}' input.json
2 > "Matti"
```

5.4 Jian jatkokehitys

Tämän työn laajuudessa Jian ominaisuudet olivat supistetut eikä ollut oletettavissa, että kehitettävä artefakti tulisi olemaan pitkään kehityksessä olleiden kilpailijoidensa veroinen ominaisuuksiltaan. Erityisesti erilaisten ominaisuuksien kirjo on esimerkiksi jq:ssa [7] niin paljon laajempi, että sen kanssa kilpailu olisi epärealistista tämän työn puitteissa.

Toinen tämän työn tutkimuskysymyksistä, TK2, oli selvittää voisiko Egisonin kaltaisesta hahmoperusteisesta kielestä saada selkeämpää ja helpommin hyväksyttyvää. Eräs tapa miten tähän pyrittiin oli pitää kehitettävä artefakti uskollisena tarkoituksellaan ja välttää niin

7. https://stedolan.github.io/jq/manual/

sanottua ominaisuus väsymystä (engl. feature fatigue). Rust, Thompson ja Hamilton (2006) kuvaavat ominaisuus väsymyksen tekevän tuotteesta ylitsepääsemättömän monimutkaisen loppukäyttäjille ja täten huonontavan tuotteen käyttökelpoisuutta. Tässä työssä hahmoperusteinen ohjelmointi on pyritty esittelemään mahdollisimman yksinkertaisena ja ilman ylimääräisiä monimutkaistuksia.

Kutenkin Jiassa varmasti vielä on ominaisuuksia, jotka olisivat tarpeellisia ihan sen peruskäyttötarkoituksessa. Kyselyissä nousseiden huomioiden ja Egisonin pohjalta olen kerännyt listan mahdollisista jatkokehitys-ominaisuuksista. Listassa olevat kohdat eivät olet mitenkään arvotettu tai missään tietystä järjestyksessä.

- Nykyisten ominaisuuksien hiominen, erityisesti listojen toiminta
- Tiettyjen Egisonin hahmojen hyödyntäminen, esimerkiksi villikortti hahmo
- Eri loogiset operaattorit ehtojen operandeille
- Arvojen vertailuoperaattorit
- Sisäänrakennettujen funktioiden toteutus vrt. xml:n fn-nimiavaruus [8]
- Komentorivikäyttöliittymän jatkokehitys

8. https://docs.microsoft.com/en-us/sql/xquery/xquery-functions-against-the-xml-data-type?view=sql-server-ver15

6 Artefaktin evaluointi

6.1 Evaluoinnissa käytetyt tilastolliset menetelmät

Evaluoinnissa on käytetty tilastollisia menetelmiä siltä osin, kun se on ollut mahdollista ja selkeyttänyt tulosten esittämistä sekä tulkintaa. Ensimmäisessä evaluoinnissa vastausmäärä oli riittävä, että tilastollisten menetelmien käyttö oli mahdollista. Siihen metodiksi valikoitui Pearsonin khiin neliö -testi.

Pearsonin khiin neliö -testiä käytetään kategoriselle datalle ja sen avulla voidaan arvioida miten todennäköisesti muutokset eri joukkojen välillä ovat sattumanvaraisia. Taulukoissa näkyvä p-arvo on saatu siten, että verrokkina on ollut tasaisesti jakautunut jakauma, eli jakauma jossa kaikki vaihtoehdot ovat yhtä todennäköisiä. Näin tehdyllä testillä on saatu p-arvo, joka ollessaan riittävän pieni kertoo meille, että eroavaisuudet vastausdatan ja tasaisen jakauman välillä ovat epätodennäköisesti sattumasta johtuvia. Toisin sanoen, jos p:n arvo on pieni on todennäköistä, että jokin vastausvaihtoehdoista koettu paremmaksi kuin muut. Huomionarvoista on, että testi ei kerro meille suuntaa vaihtoehtojen välillä. (Pearson 1900).

6.2 Ensimmäinen evaluointikierros

Ensimmäisen evaluointikierroksen kysely oli kolme-osainen. Jokainen osa oli jaettu omalle sivulleen. Ensimmäisellä sivulla oli kysymyksiä joissa vastaajaa pyydettiin arvioimaan mikä annetusta kolmesta vaihtoehdosta on vastaajan mielestä selkein. Toisella sivulla vastaajan tuli valita kahden vaihtoehdon väliltä selkeämpi ja kolmannella sivulla oli avoimia kysymyksiä liittyen syntaksin ymmärtämiseen.

Kaikki ensimmäisen sivun kysymykset sisälsivät samat vaihtoehtoiset syntaksit ja lisäksi kaikki vastausvaihtoehdot oli kirjoitettu siten, että ne ovat semanttisesti samat. Ainut vaihtelu syntyi siis syntaksista, sillä myös syöte ja odotettu tuloste olivat samat. Kysymyksissä oli tarkoituksella jätetty selittämättä mitä annetun ohjelmakoodin osuuden tulisi tehdä.

Ensimmäisen evaluointikierroksen kyselyyn saatiin 50 vastausta.

6.3 Ensimmäisen evaluoinnin vastaukset

Kaikki ensimmäisen sivun kysymykset sisälsivät samat kolme syntaksi vastausvaihtoehtoa: jia, jq ja jia-lyhenne. Kaikki ohjelmakoodiotteet olivat lyhyehköjä (maksimissaan kolme rivisiä) ja niissä esiteltiin eri syntaksi ominaisuuksia. Lähtökohtaisesti kyselyt kirjoitettiin sillä idealla, että halutaan tietty data ja se on sitten muutettu ohjelmakoodiksi, sekä jia-syntakseille että jq:lle.

Vastaukset on esitelty taulukossa 2 prosenttiosuuksina ja tarkkoina vastausmäärinä. Kyselyssä kysymykset oli numeroitu, mutta eivät sisältäneet taulukon 2 mukaisia sanallisia selityksiä, ettei vastaajille tule ennakko-olettamuksia. Kysymykset on nimetty taulukkoon luettavuuden sekä ymmärrettävyyden parantamiseksi.

Taulukossa 2 nähdään, että kaikissa kysymyksissä vähintään 50% vastaajista piti jia-lyhenne syntaksia selkeimpänä vaihtoehtona. Lyhenteettömän jia-syntaksin ja jq-syntaksin välillä erot olivat pienempiä, mutta jia-syntaksia pidettiin yhtä vaille kaikissa kysymyksissä selkeämpänä kuin jq:ta. Ainut poikkeus edelliseen on neljäs kysymys jossa molemmat vaihtoehdot saivat yhtä paljon ääniä.

Taulukko 2: Ensimmäisen sivun tulokset

Kysymys	**Jia**	**Jq**	**Jia-lyhenne**	**p-arvo**
1. Muuttujan palautus	22,0% (11)	20,0% (10)	58,0% (29)	0.046
2. Wildcard palautus	24,0% (12)	0,0% (0)	76,0% (38)	< 0.001
3. Sisäkkäisen muuttujan palautus	22,0% (11)	14,0% (7)	64,0% (32)	0.007
4. Listan ensimmäisen palautus	22,0% (11)	22,0% (11)	56,0% (28)	0.074
5. Moni palautus	30,0% (15)	20,0% (10)	50,0% (25)	0.181
6. Ehdollinen palautus	24,0% (12)	12,0% (6)	64,0% (32)	0.005
7. Monen ehdon palautus	24,0% (12)	12,0% (6)	64,0% (32)	0.005

Avoimeen palautteeseen vastasi vain murto-osa kyselyyn vastaajista, sivusta riippuen kuuden ja kahdentoista vastauksen välillä kaikista viidestäkymmenestä kyselyyn vastaajasta. Tämä oli oletettavissa, kun vastauskentät oli jätetty vapaaehtoisiksi.

Ensimmäiseltä sivulta avoimessa palautteessa nousi eniten esille:

- arviointia hankaloittaa kun ei tiedä, mitä kyseessä vaihtoehtoina olevien ohjelmakoodin osasten tulisi tehdä
- destrukturointia muistuttavan syntaksi on luonteva
- muuttujien eroattaminen on hankalaa ilman syntaksinkorostusta
- muuttujat selkeyttävät kyselyitä
- muuttujat ovat turhia ja tekevät kyselyistä monimutkaisempia

Viimeisestä kahdesta erikseen nousseesta havainnosta huomataan, että kaikki kyselyyn vastanneet eivät olleet samaa mieltä. Nämä ja muut muuttujien käyttöön liittyvät havainnot olivat kautta koko kyselyn kiinnostavia, sillä juuri muuttujien käyttö, erityisesti jia-lyhennesyntaksissa herätti kysymyksiä ja epävarmuutta jo ensimmäisessä syntaksin suunnitteluvaiheessa.

Toisella sivulla oli 8 kysymystä jotka oli jaettu neljään kahden kysymyksen joukkoon. Jokaisessa kysymysparissa oli yksinkertaisempi ja monimutkaisempi kysely kirjoitettuna kahdella eri syntaksilla. Taulukossa yksinkertaisempi kysymys on numero 1 ja haastavampi numero 2. Kysymysten osalta tälläkään sivulla ei erikseen kerrottu vastaajalle mitä ohjelmakoodit tarkalleen kuvaavat, vaan vastaajan tuli itse tulkita niitä.

Tällä sivulla oli kaksi uutta syntaksia. Jia-putki syntaksi, eli rivinvaihdot on korvattu |-merkillä, kuten jq:ssa, muuten tämä syntaksi vastaa jia-lyhenne syntaksia. Toinen uusi syntaksi on jia-lyhenne ilman muuttujia. Tässä versiossa ei erikseen esitellä muuttujia jotka palautetaan vaan, kaikki eritellyt arvot palautetaan. Johtuen siitä, että nyt on käytössä kahta eri lyhenne syntaksia puhutaan taulukossa 3 erikseen jia-lyhenne syntaksista muuttujilla, mutta on huomionarvoista, että se on sama kuin aikaisemmin esitelty jia-lyhenne syntaksi.

Kysymyksissä esiintyivät seuraavat syntaksiparit jia-putki ja jia, jia-putki syntaksi ja jia-lyhenne, jia ja jia-lyhenne ilman muuttujia, jia-lyhenne muuttujilla ja jia-lyhenne ilman muuttujia. Näistä pareista parhaiten pärjäsivät molemmat jia-lyhenne syntaksit. Lyhennesyntakseista puolestaan muuttujaton versio koettiin tässä kyselyssä selkeämmäksi, mutta monimutkaisemmassa esimerkissä vai 56,0%-yksikön enemmistöllä.

Taulukko 3: Toisen sivun tulokset

Kysymys	**Vaihtoehto 1**	**Vaihtoehto 2**	**p-arvo**
9. Jia-pipe & jia 1	26,0% (13)	74,0% (37)	0.023
10. Jia-pipe & jia 2	14,0% (7)	86,0% (43)	< 0.001
11. Jia-pipe & jia-lyhenne	26,0% (13)	74,0% (37)	0.023
12. Jia-pipe & jia-lyhenne 2	44,0% (22)	56,0% (28)	0.689
13. Jia & jia-lyhenne 1	38,0% (19)	62,0% (31)	0.313
14. Jia & jia-lyhenne 2	36,0% (18)	64,0% (32)	0.226
15. Jia-lyhenne muuttujilla &ilman muuttujia 1	36,0% (18)	64,0% (32)	0.226
16. Jia-lyhenne muuttujilla &ilman muuttujia 2	44,0% (22)	56,0% (28)	0.689

Toisella sivulla avointa palautetta ei tullut yhtä paljon kuin ensimmäisellä sivulla, sieltä nousi esille seuraavat asiat:

- muutaman esimerkin jälkeen ohjelmakoodien merkitys selkenee
- yksinkertaisissa esimerkeissä muuttujien implisiittinen nimeäminen on jopa häiriö.
- monimutkaisemmissa esimerkeissä muuttujat auttavat silmäilyä merkittävästi

Ylläolevasta listauksesta huomataan, että tälläkin sivulla muuttujien käyttö syntakseissa jakaa mielipiteitä. Toisaalta viimeisessä nostossa vastaaja oli korostanut, että hyöty ilmenee haastavissa erityisesti haastavissa esimerkeissä.

Kolmannella sivulla vastaajien tehtävä oli kertoa mitä annettu ohjelmakoodin ote tekee. Kysymyksiä oli yhteensä 8 kappaletta, mutta kukin vastaaja vastasi vain neljään. Ryhmät oli jaettu satunnaisesti eli kysely työkalu päätti kumman kysymysryhmän kukin vastaaja saa. Ensimmäisessä kysymysryhmässä oli käytössä jia- ja jia-lyhenne ilman muuttujia -syntaksi. Toisessa kysymysryhmässä oli käytössä jq- ja jia-pipe-syntaksit. Kysymykset olivat semanttisesti vastaavat ryhmissä, eli jia-kysymykset vastasivat jq-kysymyksiä ja jia-lyhenne-kysymykset jia-pipe kysymyksiä. Tähän poikkeuksena on jia 2. ja jq 2. kysymys, tässä oli kyselyssä virhe ja jq-kysymys oli merkittävästi yksinkertaisempi joten näiden kahden kysymyksen vastaukset eivät ole keskenään verrattavissa.

Taulukkoa 4 katsomalla huomataan, että eniten oikeita vastauksia on saatu jia-pipe syntaksilla esitettyihin kysymyksiin, yksinkertaisempaa jia-lyhenne kysymykseen ja yksinkertaisempaan jia-kysymykseen. Keskiarvolta oikeita vastauksia oli 63,8% ja edellä mainitut ylittivät tuon keskiarvon. Huomioitavaa on, että jq-syntaksi kysymysten oikein vastanneiden prosentti jäi keskiarvon alle molemmissa kysymyksissä. Toisaalta jq:n toinen kysymykseen vastasi enemmän oikein kuin jia- tai jia-lyhenne -syntaksin haastavampaan toiseen kysymykseen. Tässä toki muistettava, että jq-kysymys oli suhteessa helpompi. Näiden kysymysten vastausten ei koettu olevan keskenään verrannollisia asiaa selventävästi, joten tässä ei ole laskettu p-arvoja.

Jatko-kehityksen kannalta mielenkiintoisin ero on jia-pipe ja jia-lyhenne ilman muuttujia -syntaksin välillä. Tämä siksi, että jia-pipe on muuten sama kuin jia-lyhenne muuttujilla -syntaksi, ainoana eroana on rivinvaihtojen korvaaminen pipe-symbolilla. Vaikuttaa siis siltä, että semantiikan ymmärtäminen vastaajille on helpompaa, kun palautettavat arvot on esitelty implisiittisesti muuttujien avulla.

Kolmannelle sivulle avointa palautetta tuli vielä vähemmän kuin aiemmille ja niistä ilmeni olennaisesti kaksi asiaa:

- vastausrakenteen arvauksen haastavuus
- tehtävien yleinen haastavuus

Ensimmäisestä havainnosta huomaa, että vastaaja oli selkeästi ymmärtänyt syntaksin perusidean ja että oli alkanut jo pohtimaan syvemmin, kuin kyselyssä oli tarkoituskaan, missä muodossa vastaus palautetaan. Eikä haastetta niinkään tuonut enää oikeiden arvojen tai semantiikan ymmärrys.

Tehtävien haastavuus oli ehkä hieman yllättävää, koska tarkoituksena ei ollut valita kaikkein vaikeimpia esimerkkejä tähän osuuteen. Toisaalta tästä nousee hyvin esille, että lyhyetkin ohjelmakoodin otteet voivat olla haastavia tulkita, vaikka niissä olisi ainakin jossain määrin käytetty tuttuja konventioitakin.

Taulukko 4: Kolmannen sivun tulokset

Kysymys	Oikeat vastaukset	Väärät vastaukset	Kysymysryhmä
Jia 1.	70% (14)	30% (6)	1
Jia 2.	45% (9)	55% (11)	1
Jia-lyhenne 1	75% (15)	25% (5)	1
Jia-lyhenne 2	50% (10)	50% (10)	1
Jq 1.	60% (18)	40% (12)	2
Jq 2.	56,7% (17)	43,3% (13)	2
Jia-pipe 1.	76,7% (23)	23,3% (7)	2
Jia-pipe 2.	76,7% (23)	23,3% (7)	2

6.4 Toinen evaluointikierros

Toinen evaluointi järjestettiin kun ensimmäinen toimiva prototyyppi Jiasta oli valmiina. Toisessa evaluoinnissa oli kolme osallistujaa, jotka kaikki olivat samaa kohderyhmää kuin ensimmäiseen evaluointiin osallistujat, tietoa siitä osallistuivatko he ensimmäiseen evaluointiin ei kerätty. Osallistujat saivat kirjallisen ohjeistuksen kera ohjelmointitehtäviä, jotka heidän tuli ratkaista Jian avulla. Evaluointi oli alkuperäistä suunnittelua pienempi johtuen tutkimuksen venymisestä ja aikataulusta.

Toisessa evaluoinnissa käytetyssä Jian versiossa ei ollut läheskään kaikkia alunperin suunniteltuja ominaisuuksia, mutta se sisälsi riittävästi toiminnallisuuksia, että pienen mittakaavan koekäyttö ja evaluointi oli mahdollista. Tässä evaluoinnissa osallistujat käyttivät Visual Studio Code[1]-koodieditoria ja heille oli asetettu siihen valmiiksi markdown-tiedosto, joka sisälsi ohjeet sekä tehtävät joihin he vastasivat samaan tiedostoon. Lisäksi osallistujilla oli mahdollisuus testata Jian toimintaa haluamaansa json-tiedostoa vasten käyttämällä Visual Studio Coden build-toiminnallisuutta, jonka käyttö heille esiteltiin. Liitteenä C PDF-versio markdown-tiedostosta, jossa annettiin ohjeet osallistujille ja johon vastaukset kerättiin.

Osallistujille esitettiin kolme esimerkkiä, joihin kuhunkin liittyi kolme ongelmaa. Jokaseen

1. https://code.visualstudio.com/

esimerkkiin liittyi osuus, jossa kuvattiin Jia:n toiminnallisuutta, jota seuraavissa tehtävissä osallistujan tuli hyödyntää. Varsinaisten ongelmien lisäksi osallistujilla oli mahdollisuus antaa sanallista palautetta Jiasta ja evaluoinnin tehtävistä.

6.5 Toisen evaluoinnin vastaukset

Tässä evaluoinnissa pyrittiin saamaan palautetta siitä mikä Jian käytönoppimisessa oli hyvää ja mikä oli huonoa ja siksi tähän osuuteen ei kerätty suurta määrää osallistujia. Johtuen pienestä osallistujamäärästä, ei luonnollisestikaan voida tehdä mitään tilastollista päättelyä, vaan evaluoinnin päätarkoituksena oli saada palautetta Jian käytöstä ja sen oppimisesta avoimen palautteen avulla

Osallistujien mielestä Jiassa oli hyvää

- Kyselyiden teko oli intuitiivistä, kun pystyi mukailemaan Jsonin rakennetta
- Koko Jsonin rakennetta ei tarvi toistaa, vaan riittää merkitykselliset osuudet
- Kyselyiden kirjoittamnen on intuitiivistä, kun kieli muistuttaa luettavaa tietoa
- Yksinkertaisissa hauissa logiikka on tosi selkeä
- Tehtävien tekeminen Jialla oli tosi nopeaa, kun pääsi alkuongelmasta eteenpäin.

Osallistujien mielestä Jiassa oli kehitettävää

- Virheilmoitukset eivät olleet kuvaavia tai viittanneet oikeaan paikkaan (välillä toimivat hyvin)
- Aaltosulkeiden paikan ja kyselyn sisäisen rakenteen ymmärtäminen vaati aikaa
- Sulkeiden ja lainausmerkkien määrä hankaloitti

Osallistujien vapaa palaute

- Kieltä on hauska kirjoittaa ja olisi hauska kokeilla monimutkaisempiakin rakenteita
- Kieltä on mukava kirjoittaa jos jsonin rakenne on tuttu ja intuitiivinen

Taulukkoon 5 on kerätty oikeiden ja väärien vastausten määrät esimerkeittäin.

Taulukko 5: Toisen evaluoinnin vastaukset

Esimerkki	**Oikeiden vastausten osuus**	**Väärien vastausten osuus**
Esimerkki 1.	100%	0%
Esimerkki 2.	78%	22%
Esimerkki 3.	56%	44%

7 Johtopäätökset

Ensimmäisen evaluointikierroksen vastausten perusteella todettiin, että kehityksen suunta oli tutkimuskysymyksiin nähden oikea sillä vastaajat kokivat Jian syntaksin pääosin helpommaksi ja intuitiivisemmaksi kuin jq-syntaksin, kuten taulukosta 2 nähdään. Kaikissa taulukon 2 kysymyksissä Jia ja Jia-lyhenne -syntakseja pidettiin vähintään yhtä usean vastaajan toimesta parempana kuin jq:ta. Tilastollisesti merkittäviä eroja joissa p-arvo oli riittävän pieni ($p < 0.05$) saatiin viidessä kysymyksessä seitsemästä. P-arvo ei kerro tässä suuntaa, mutta se voidaan päätellä siitä, että kaikissa kysymyksissä joissa p-arvo on pieni, on jq-syntaksi saanut vähiten kannatusta näin ollen voidaan todeta, että Jia-syntaksivariantteja pidettiin parempana vaihtoehtona eikä se todennäköisesti johtunut satunnaisvaihtelusta.

Toisen sivun kysymyksissä taulukossa 3 tilastollisesti merkittäviä eroja on saatu vain kysymyksissä 9, 10 ja 11. Tässä kuitenkin huomattavaa, että vertaillut vaihtoehdot ovat kaikki jia-johdannaisia eikä jq-syntaksi ollut mukana lainkaan. Tilastollisesti merkittävät erot kohdistuivat Jia-pipe ja Jia syntaksien vertailuun sekä Jia-pipe, että Jia-lyhenne -syntaksien vertailuun. Näissä käy ilmi, että harva vastaaja piti Jia-pipe syntaksia parempana kuin muita vaihtoehtoja. Muuten tulokset olivat tasaisempia. Pieniä suuntaumia vastaajien mielipiteissä voidaan huomata siihen suuntaan, että Jia-lyhennettä pidettiin parempana kuin muita ja toisaalta Jia-lyhenne ilman muuttujia oli hieman useamman vastaajan mieleen kuin Jia-lyhenne muuttujilla. Näissä kuitenkin erot olivat pieniä ja menevät todennäköisemmin satunnaisvaihtelun piiriin.

Ensimmäisen kyselyn kolmannen sivun vastauksista ei pystytty tulkitsemaan selkeitä johtopäätöksiä johtuen kysymysasettelusta ja toisen Jia-lyhenne-syntaksin sekä Jia-syntaksi kysymysten vaikeudesta verrattuna esimerkiksi jq-kysymyksiin. Näiden haasteiden vaikutus nähdään myös hajonnassa taulukossa 4. Kolmannen sivun avoimesta palautteesta kuitenkinnousi pohdittavaa lyhennesyntaksin sekä muuttujien käytöstä. Implisiittistä muuttujien käyttöä tulisi tutkia enemmän uudella kyselyllä ja sitä kautta jatkokehittää syntaksia.

Toisessa evaluoinnissa palaute oli pääosin positiivista ja rakentavan palautteen puoleltakin saatiin vahvistusta, että oikeisiin asioihin on kiinnitetty huomiota. Erityisesti eräs huomio

virheviestien selkeydestä korreloi vahvasti luvussa 4.5 pohdittujen käytänteiden kanssa. Toki huomionarvoista on, että virheviesteihin jäi parannettavaa. Toisen arvioinnin pienestä mittakaavasta johtuen tuloksien yleistys on mahdotonta, mutta jatkokehityksen näkökulmasta saatin arvokasta palautetta syntaksin selkeydestä ja asioista joihin jatkokehityksessä tulisi tulisi kiinnittää huomiota. Tiivistelmä avoimesta palautteesta luvussa 6.5. Jatkokehityksessä tulisi kuitenkin erityisesti kiinnittää huomiota siihen, miksi toisen evaluoinnin loppupään esimerkit aiheuttivat ongelmia, kun Jian oli tarkoitus nimenomaan näissä tilanteissa olla jq:ta parempi. Toisaalta vastaavia tehtäviä ei tehty jq:lla joten keskinäinen vertailu on myöskin mahdotonta toisen evaluoinnin pohjalta.

Tämän työn ensimmäisen tutkimuskysymyksen TK1:n vastaus ei ole näiden evaluointikierrosten pohjalta yksiselitteinen, sillä Egisonin hankaluudesta ei löytynyt valmista tutkimustietoa ja sen evaluointi jätettiin tässäkin ulkopuolelle. Toisaalta kuitenkin Jia sai pääosin positiivista palautetta ja siitä saatiin kehitettyä toimiva prototyyppi. Tämän pohjalta voi näillä huomioilla melko luotettavasti sanoa, että hahmoperusteista ohjelmointikieltä voi saada selkeämmäksi ja hyväksyttävämmäksi ainakin DSL-kielelle.

TK2:n voidaan vastata vain rajallisesti, johtuen siitä, että jq on erittäin paljon laajempi työkalu kuin Jia. Toisaalta kuitenkin Jia sai enemmän kannatusta ja sen syntaksia pidettiin useissa esimerkeissä selkeämpänä vaihtoehtona. Nämä kyselyt ja evaluoinnit keskittyivät rajattuun osuuteen jq:n toiminnallisuuksia joten tuloksia ei voi yleistää.

8 Yhteenveto

Työn kehitettiin yksinkertainen ohjelmointikieli json-prosessointiin. Kielen ominaisuudet rajoittuivat kyselyihin ja siltäkin osin joitain alunperin suunnitelluista ominaisuuksista jäi toteuttamatta. Tutkielman teko venyi suunniteltua paljon pidempään ja joitain valmiita osuuksia jouduttiin kirjoittamaan uusiksi, sillä hahmoperusteinen ohjelmointi on saanut enemmän jalansijaa valtakielissä sitten tämän työn aloittamisen. Tämän tutkielman aiheen kannalta se on hyvä asia, sillä se todistaa että tässä tutkittuihin ominaisuuksiin ja hahmoperusteiseen ohjelmointiin paradigmana on kiinnostusta.

Evaluointikierrokset olivat hyviä ja auttoivat ymmärtämään mitkä asiat ovat hahmoperusteisesssa ohjelmoinnissa ovat hankalia ja mihin pitäisi kiinnittää huomiota. Osallistujamäärät jäivät rajallisiksi molemmissa evaluoinnissa ja varsinkin jälkimmäisen kohdalla näkyi työn venymisen aiheuttama aikataulupaine. Se ei kuitenkaan invalidoinut toistakaan evaluointikierrosta vaan siitäkin saatin arvokasta tietoa jatkokehityksen näkökulmasta.

Jia saatiin tämän työn puitteissa toimivaksi artefaktiksi, jota pystyttiin koestamaan. Tämä oli minimitavoite, joka olisi haluttu ylittää. Jiassa todettiin molemmissa kyselyissä potentiaalia, mutta myös ongelmia. Erityisen kiinnostavaksi asiaksi, jota ei päästy testaamaan jäi lyhennesyntaksin toteuttaminen ja testaaminen koehenkilöillä.

Kokonaisuudessaan työssä päästiin hyvin pureutumaan hahmoperusteisen ohjelmointiin ja sitä noudattavan ohjelmointikielen toteutukseen, mutta arviointeja ja evaluointisyklejä olisi tullut pitää enemmän ja useammin. Tämän toteutumisessa vastaan tuli Pro gradu -tutkielman laajuus tällaisessa aiheessa, joka vaatii merkittävän paljon suunnittelua ja ohjelmointia varsinaisen tutkimustyön lisäksi.

Lähteet

Binkley, Dave, Marcia Davis, Dawn Lawrie ja Christopher Morrell. 2009. “To CamelCase or under_score”. Teoksessa *2009 IEEE 17th International Conference on Program Comprehension,* 158–167. https://doi.org/10.1109/ICPC.2009.5090039.

Broberg, Niklas, Andreas Farre ja Josef Svenningsson. 2004. “Regular Expression Patterns”. Teoksessa *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming,* 67–78. ICFP ’04. Snow Bird, UT, USA: Association for Computing Machinery. ISBN: 1581-139055. https://doi.org/10.1145/1016850.1016863. https://doi.org/10.1145/1016850.1016863.

Clarke, Steven. 2001. “Evaluating a new programming language.” Teoksessa *PPIG,* 13:275–289. Citeseer.

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.

Egi, Satoshi, ja Yuichi Nishiwaki. 2018. “Non-linear Pattern Matching with Backtracking for Non-free Data Types”. Teoksessa *Programming Languages and Systems,* toimittanut Sukyoung Ryu, 3–23. Cham: Springer International Publishing. ISBN: 9783-030027681. https://doi.org/10.1007/978-3-030-02768-1_1.

———. 2020. “Functional Programming in Pattern-Match-Oriented Programming Style”. *The Art, Science, and Engineering of Programming* 4 (3). https://doi.org/10.22152/programming-journal.org/2020/4/7. http://dx.doi.org/10.22152/programming-journal.org/2020/4/7.

Hevner, Alan R., Salvatore T. March, Jinsoo Park ja Sudha Ram. 2004. “Design Science in Information Systems Research”. *MIS Quarterly* 28 (1): 75–105. https://doi.org/10.2307/25148625. http://www.jstor.org/stable/25148625.

Krishnamurthi, Shriram. 2008. “Teaching programming languages in a post-linnaean age”. *ACM Sigplan Notices* 43 (11): 81–83.

Luxton-Reilly, Andrew, ja Andrew Petersen. 2017. “The Compound Nature of Novice Programming Assessments”. Teoksessa *Proceedings of the Nineteenth Australasian Computing Education Conference,* 26–35. ACE ’17. Geelong, VIC, Australia: Association for Computing Machinery. ISBN: 9781450348232. https://doi.org/10.1145/3013499.3013500. https://doi.org/10.1145/3013499.3013500.

Marlow, Simon, ym. 2010. “Haskell 2010 language report”. *Available on: https://www. haskell. org/onlinereport/haskell2010.*

Nurseitov, Nurzhan, Michael Paulson, Randall Reynolds ja Clemente Izurieta. 2009. “Comparison of JSON and XML data interchange formats: A case study”. *22nd International Conference on Computer Applications in Industry and Engineering 2009, CAINE 2009* (tammikuu): 157–162.

Odersky, Martin, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman ja Matthias Zenger. 2004. *An overview of the Scala programming language.* Tekninen raportti.

Parnin, Chris, Christian Bird ja Emerson Murphy-Hill. 2011. “Java generics adoption: how new features are introduced, championed, or ignored”. Teoksessa *Proceedings of the 8th Working Conference on Mining Software Repositories,* 3–12.

Pearson, Karl. 1900. “X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling”. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50, numero 302 (heinäkuu): 157–175. https://doi.org/10.1080/14786440009463897. https://doi.org/10.1080/14786440009463897.

Rust, Roland T, Debora Viana Thompson ja Rebecca W Hamilton. 2006. “Defeating feature fatigue”. *Harvard business review* 84 (2): 37–47.

Severance, C. 2012. “Discovering JavaScript Object Notation” [kielellä eng]. *Computer* 45 (4): 6–8. https://doi.org/10.1109/MC.2012.132. https://jyu.finna.fi/PrimoRecord/pci.ieee_s6178118.

Simmonds, Devon M. 2012. “The programming paradigm evolution”. *Computer* 45 (06): 93–95.

Solodkyy, Yuriy, Gabriel Dos Reis ja Bjarne Stroustrup. 2013. “Open pattern matching for C++”. Teoksessa *Proceedings of the 12th international conference on Generative programming: concepts & experiences,* 33–42.

Soloway, E., ja K. Ehrlich. 1984. “Empirical Studies of Programming Knowledge”. *IEEE Transactions on Software Engineering* SE-10 (5): 595–609.

Stefik, Andreas, ja Ed Gellenbeck. 2011. “Empirical studies on programming language stimuli”. *Software Quality Journal* 19 (1): 65–99.

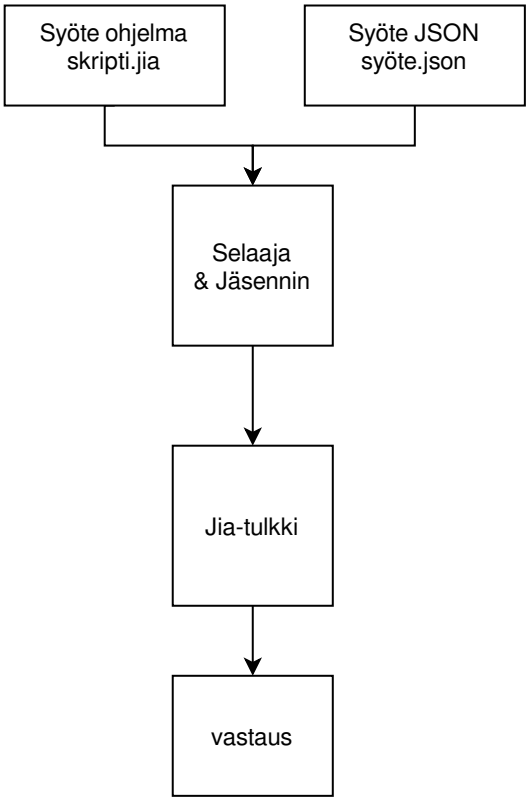
Stefik, Andreas, Stefan Hanenberg, Mark McKenney, Anneliese Andrews, Srinivas Kalyan Yellanki ja Susanna Siebert. 2014. “What is the foundation of evidence of human factors decisions in language design? an empirical study on programming language workshops”. Teoksessa *Proceedings of the 22nd international conference on program comprehension,* 223–231.

Stefik, Andreas, ja Susanna Siebert. 2013. “An Empirical Investigation into Programming Language Syntax”. *ACM Trans. Comput. Educ.* (New York, NY, USA) 13 (4). https://doi.org/10.1145/2534973. https://doi.org/10.1145/2534973.

Van Roy, Peter, ym. 2009. “Programming paradigms for dummies: What every programmer should know”. *New computational paradigms for computer music* 104:616–621.

Liitteet

A Artefaktin rakenne



B Ensimmäinen kysely

Kysely json-prosessointiin suunnitellun kielen syntaksista

Pakolliset kysymykset merkitty tähdellä (*)

Tämän kyselyn on tehnyt Janne Isoaho osana Jyväskylän yliopistolla tehtävää pro gradu -tutkielmaa. Tässä kyselyssä pyritään keräämään tietoa tutkielmassa kehitettävän JSON-kyselykielen parantamiseen. Kyselyyn vastaaminen vie vain 10-15 minuuttia. Ensimmäisellä ja toisella sivulla on valitse selkein-tyyppisiä kysymyksiä ja kolmannella sivulla on avoimia kysymyksiä.

Ensimmäisen ja toisen sivun kysymyksissä on esitetty kahdesta kolmeen vaihtoehtoista tapaa poimia tietoja json-rakenteesta. Tehtävänänne on vastata kumpi on mielestänne selkeämpi. Sivun alareunassa on vapaaehtoinen avoin palautekenttä.

Kyselyyn vastataan anonyymisti ja kyselyssä on yksi vapaaehtoinen kysymys tietyn teknologian tuntemisesta. Mitään muita tietoja tai henkilötietoja ei kerätä vastaajista. Kyselystä kerätty tieto koostettuna esitetään osana pro gradu -tutkielmaa, mutta tarkemmat vastaustiedostot poistetaan tutkimuksen päätyttyä.

Kaikissa tilanteissa syöte on seuraava. Syöte on toistettu jokaisen sivun yläreunaan, jotta siihen on helpompi palata.


```
{
   "firstName": "Matti",
   "lastName": "Meikalainen",
   "alive": true,
   "age": 45,
   "phoneNumbers": [
      {
         "type": "home",
         "number": "050 123 1234"
      },
      {
         "type": "work",
         "number": "050 321 4321"
      }
   ],
   "address": {
      "streetAddress": "Alkutie",
      "postalCode": "00100",
      "city": "Helsinki"
   },
   "socialMedia": [
      {
         "service": "twitter",
         "username": "@matti"
      },
      {
         "service": "instagram",
```

```
            "username": "@igmatti"
        },
        {
            "service": "linkedin",
            "username": "Matti Meikalainen"
        }
    ],
    "children": []
}
```


Ensimmäisen sivun kysymyksissä kaikki vaihtoehdot kuvaavat samaa toiminnallisuutta. Tehtävänänne on valita mielestänne selkein vaihtoehto. Valitkaa selkein sen perusteella mistä on mielestänne helpoin ja yksinkertaisin ymmärtää koodin toiminta.

1. Mikä on selkein? *

◯ '{firstName}' Input

◯ '.firstName' Input

◯ {firstName: x} <- Input
return x

2. Mikä on selkein? *

◯ '{"*Numbers":{number}}' Input

◯ 'to_entries[] | select(.key|endswith("Numbers")) | .value | .[] | .number' Input

◯ {"*Numbers": {number:X}} <- Input
return X

3. Mikä on selkein? *

◯ '{phoneNumbers:{number:x}}' Input

◯ '.phoneNumbers[] | .number' Input

◯ {phoneNumbers: {number:X}} <- Input
return X

4. Mikä on selkein? *

◯ '{socialMedia:[x..]}' Input

◯ '.socialMedia, | first' Input

◯ {socialMedia: [x..]} <- Input
return x

5. Mikä on selkein? *

◯ '{firstName: x, lastName: y}' Input

◯ '.firstName, .lastName' Input

◯ {firstName:x, lastName:y<- Input
return (x,y)

6. Mikä on selkein? *

◯ '{alive:true, socialMedia:{service:"linkedin", username: x}}' Input

◯ 'select(.alive==true) | .socialMedia[] | select(.service=="linkedin") | .username' Input

◯ {alive:true, socialMedia:{service:"linkedin", username: x}} <- Input
return x

7. Mikä on selkein? *

◯ '{address:{city:"Helsinki"}, isAlive:true, children:[], age: x}' Input

◯ 'select(.address.city=="Helsinki") | select(.isAlive==true) | select(.children==[]) | .age' Input

◯ {age: x, address:{city: "Helsinki"}, isAlive: true, children: []} <- Input
return x

8. Avoin palaute sivun yksi kysymyksistä

Tämän sivun kysymykset ovat muuten samanlaisia kun ensimmäisen sivun, mutta vastausvaihtoehtoja on vain 2 per kysymys. Valitse siis mielestäsi selkeämpi vaihtoehtoinen syntaksi.

Tämänkin sivun kysymyksissä input on sama ja sen näkee alla.

Input:

```
{
   "firstName": "Matti",
   "lastName": "Meikalainen",
   "alive": true,
   "age": 45,
   "phoneNumbers": [
      {
         "type": "home",
         "number": "050 123 1234"
      },
      {
         "type": "work",
         "number": "050 321 4321"
      }
   ],
   "address": {
      "streetAddress": "Alkutie",
      "postalCode": "00100",
      "city": "Helsinki"
   },
   "socialMedia": [
      {
         "service": "twitter",
         "username": "@matti"
      },
      {
         "service": "instagram",
         "username": "@igmatti"
      },
      {
         "service": "linkedin",
         "username": "Matti Meikalainen"
      }
   ],
   "children": []
}
```

9. Kumpi on selkeämpi *

◯ {firstName:x, lastName:y} <- Input | return (x,y)

◯ {firstName:x, lastName:y} <- Input
return (x,y)

10. Kumpi on selkeämpi *

◯ {phoneNumbers: {number: "050 123 1234", type:x}} <- Input
{socialMedia: {username:"@matti", service: y}} <- Input
return (x,y)

◯ {phoneNumbers: {number: "050 123 1234", type:x}} <- Input | {socialMedia: {username:"@matti", service: y}} |<- Input |return (x,y)

11. Kumpi on selkeämpi? *

◯ lastName Input

◯ {lastName:x} <- Input | return x

12. Kumpi on selkeämpi? *

◯ '{"address":{"city":"Helsinki"}, "isAlive":true, age}' Input

◯ {age: x, address:{city: "Helsinki"}, isAlive: true, children: []} <- Input | return x

13. Kumpi on selkeämpi? *

◯ {socialMedia:{service}} Input

◯ {socialMedia:{service:x}} <- Input
return x

14. Kumpi on selkeämpi? *

◯ {socialMedia:{service:"twitter", username:"*"}, address:{streetAddress:"Alkutie", postalCode}} Input

◯ {socialMedia:{service:"twitter", username}, address:{streetAddress:"Alkutie", postalCode:x}} <- Input
return x

15. Kumpi on selkeämpi *

◯ {firstName, lastName} Input

◯ {firstName:x, lastName: y} Input

16. Kumpi on selkeämpi *

◯ {"address":{"city":"Helsinki"}, "isAlive":true, "children":[], age} Input

◯ {"address":{"city":"Helsinki"}, "isAlive":true, "children":[], age: x} Input

17. Avoin palaute sivun kaksi kysymyksistä

Tämän sivun kysymyksissä on vastaavia otteita ohjelmakoodista mutta nyt tehtävänänne on kertoa mitä ne tekevät ja mikä on ns. output eli mitä kysely palauttaa.

Koodiotteita on useista eri syntakseista.

Input:

```
{
   "firstName": "Matti",
   "lastName": "Meikalainen",
   "alive": true,
   "age": 45,
   "phoneNumbers": [
      {
         "type": "home",
         "number": "050 123 1234"
      },
      {
         "type": "work",
         "number": "050 321 4321"
      }
   ],
   "address": {
      "streetAddress": "Alkutie",
      "postalCode": "00100",
      "city": "Helsinki"
   },
   "socialMedia": [
      {
         "service": "twitter",
         "username": "@matti"
      },
      {
         "service": "instagram",
         "username": "@igmatti"
      },
      {
```

```
            "service": "linkedin",
            "username": "Matti Meikalainen"
        }
    ],
    "children": []
}
```

18. Mitä seuraava ohjelmakoodi tekee ja mikä paluuarvo?

{firstName:x, lastName:y} <- Input
return (x,y) *

19. Mitä seuraava ohjelmakoodi tekee ja mikä paluuarvo?

{phoneNumbers: {number: "050 123 1234", type:x}} <- Input
{socialMedia: {username:"@matti", service: y}} <- Input
return (x,y) *

20. Mitä seuraava ohjelmakoodi tekee ja mikä paluuarvo?

{alive, age} Input *

21. Mitä seuraava ohjelmakoodi tekee ja mikä paluuarvo?

{socialMedia: {username, service:"twitter"}} Input *

22. Mitä seuraava ohjelmakoodi tekee ja mikä paluuarvo?

'.firstName, .lastName' Input *

23. Mitä seuraava ohjelmakoodi tekee ja mikä paluuarvo?

'.phoneNumbers |.[] | select(.number=="050 123 1234") | .type ' Input *

24. Mitä seuraava ohjelmakoodi tekee ja mikä paluuarvo?

{alive:x, age:y} <- Input | return (x,y) *

25. Mitä seuraava ohjelmakoodi tekee ja mikä paluuarvo?

{socialMedia: {username:x, service: "twitter"}} <- Input | return x *

Osa kyselyssä olleista ohjelmakoodin otteista oli jq:n (https://stedolan.github.io/jq/) syntaksia. Teidän ei

tarvinnut tietää tätä etukäteen. (Vapaaehtoinen) Arvioisitteko vielä jq osaamisenne.

26. Miten hyvin osaatte käyttää jq:ta?

- ◯ En tiedä mikä jq on
- ◯ 0 (Ei ollenkaan)
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5 (Erittäin hyvin)

27. Avoin palaute sivun kolme kysymyksistä

_____

_____

_____

_____

_____

C Toinen kysely

Ohjeet ja vastaukset

Tässä markdown-tiedostossa on ohjeet kyselyyn vastaamiseen, ja paikat varsinaisille vastauksille lopussa “Kysely”-osiossa.

Kyselyn tarkoitus ja Jia-ohjelmointi kieli

Kyselyn avulla pyritään kehittämään jia-ohjelmointikieltä. Kyselyssä ei mitata osallistujan taitoja.

Kyselyssä on 3 osiota, jokaisessa osiossa on oma esimerkki-json ja jokaiseen esimerkkiin liittyy kolme tehtävää näiden lisäksi lopuksi on muutama avoin kysymys. Esimerkit löytyvät VSCoden file explorerista nimillä `esimerkki1.json` , `esimerkki2.json` ja `esimerkki3.json` mutta niiden sisältö on kopioitu myös tähän dokumenttiin kysymyksien yhteyteen.

Tehtäviin vastataan kirjoittamalla pyydetty Jia-ohjelmakoodi sille annettuun koodiblokkiin. Avoimiin kysymyksiin vastataan vapaamuotoisesti, esimerkiksi ranskalaisin viivoin.

Jian perustoiminta

Jia on kyselykieli json-muotoisen datan käsittelyyn.

Jia-koodi muodostuu kahden tyyppisistä lauseista, 1. hahmontunnistuslauseista, esim:

```
{"nimi": retval} <- input
```

ja 2. palautuslaiseista, esim:

```
return retval
```

Yksinkertainen kokonainen ohjelma on siis näiden yhdistelmä:

```
{"nimi": retval} <- input
return retval
```

HUOM. hahmontunnistuslauseissa seuraava osuus:

```
<- input
```

tarkoittaa syötteenä annettavaa json-tiedoston sisältöä. Tämän kyselyn puitteissa se on käytännössä aina vakio, koska tehtävät on tarkoitus ratkaista yhdellä hahmontunnistus-lauseella.

Tässä ohjelmassa haetaan json-tiedostosta avaimen `nimi` arvo, joka sijoitetaan “**retval**”-nimiseen muuttujaan ja joka sitten palautetaan ohjelman tuloksena return-lauseessa. Tehtävien yhteydessä tulee on vielä lisäesimerkkejä. Seuraavaksi huomioita kyselyyn vastaamisesta ja koodin testaamisesta.

Tehtäviin vastaaminen

- Vastaa tehtäviin kopioimalla koodi oikeaan kohtaan, esimerkiksi Esimerkki1:n ensimmäinen tehtävä alaotsikon `E1. teht 1.` alle.
- Esimerkeissä ei ole käytetty ääkkösiä vaan ä ja ö-kirjaimet on korvattu a:lla ja o:lla

Koodin testaaminen

Koodia voi testata painamalla `F5`. Se ajaa jia-koodin tiedostossa `koodiTestaukset.jia` käyttämällä syötteenä `jsonTestaukset.json`-tiedostoa.

Esimerkki workflow:sta voisi olla, että ensin kopioi halutun esimerkin sisällön`jsonTestaukset.json`-tiedoston sisällöksi ja sitten alkaa käsin kirjoittamaan koodia `koodiTestaukset.jia`-tiedostoon. Olen tiedostojen alkusisältö on seuraava esimerkki-json ja sitä vastaava jia.

Kysely

Esimerkki1.

```
{
    "etunimi": "Pate",
    "sukunimi": "Kivi",
    "ammatti": "Postimies",
    "lemmikki": {
        "laji": "kissa",
        "nimi": "Jessi"
    },
    "siviilisaaty": "naimisissa",
    "puoliso": {
        "nimi":"Saara Kivi"
    }
}
```

Kuten aikaisemmin näytettiin jia:ssa pystyy hakemaan tiettyjä tietoja kentän nimen perusteella. Jia-koodissa tulee mallintaa sen verran hahmoa, kuin haluaa tietoja saada. Esimerkiksi tässä tapauksessa jo halutaan saada nimi olisi Jia-koodi seuraavanlainen:

```
{"etunimi" : x} <- input
return x
```

Jolloin jia palauttaa seuraavasti:

```
"AST BEGIN"
Right (Iinput [IinputLine (PsJsonPattern (JsonPattern
(JsonElement (JvJsonObject (JoMembers (JsonMembersSingle
```

```
(JsonMember (JsonString "etunimi") (JsonElement
(JvPatternVar (PatternVar "x")))))))))) (Rside (PatternVar "input")),
IinputLineReturn (ReturnStmt (RetValList [RetVal (RetOpNone,PatternVar "x")]))])
"AST END"
"----"
"RESULT PRINT"
"Pate"
```

Tässä ei tarvi välittää tiedosta ennen `RESULT PRINT` -osuutta (muut tiedot liittyvät debuggaukseen josta ei tässä tarvitse välittää). `RESULT PRINT`:ssä nähdään varsinainen palautus eli "`Pate`"

Jos puolestaan halutaan palauttaa jotain syvemmältä json-rakenteesta pitää mallintaa syvenevä rakenne haluttuun pisteeseen asti.

Esimerkiksi jos halutaan selvittää `lemmikki`-elementin `laji`-elementin arvo:

```
{"lemmikki" : {"laji":x}} <- input
return x
```

Jolloin Jia palauttaa (debug-tiedot poistettu):

```
..
"RESULT PRINT"
"kissa"
```

Esimerkki1 kysymykset & vastaukset

E1. teht 1.

Kirjoita jia-koodi, joka palauttaa `sukunimi`-elementin arvon

Vastaus:

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E1T1
```

E1. teht 2.

Kirjoita jia-koodi, joka palauttaa `lemmikki`-elementin allaolevan `nimi`-elementin arvon

Vastaus:

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E1T2
```

E1. teht 3.

Kirjoita jia-koodi, joka palauttaa `puoliso`-elementin allaolevan `nimi`-elementin arvon

Vastaus:

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E1T3
```

Esimerkki2.

```
{
    "etunimi": "Ash",
    "sukunimi": "Ketchum",
    "ammatti": "Pokemon-kouluttaja",
    "pokemon": {
        "laji": "Pikachu",
        "vari": "keltainen",
        "kuvio": "raidallinen",
        "tyyppi": "sahko",
        "heikkous": "maa"
    }
}
```

Jiassa on mahdollista myös antaa ehtoja sille, missä tilanteessa tietoja palautetaan. Tosimaailman esimerkkinä voitaisiin käyttää esimerkiksi tuotekatalogia josta halutaan palauttaa tietyn tyyppiset tai arvoiset tuotteet.

Tässä esimerkissä kuitenkin käsitellään hieman eri domainia ja halutaan palauttaa `Ash`-nimisen kouluttajan `pokemonin` tiedoista sen `tyyppi`. Jiassa tällainen kysely toteutaan seuraavasti:

```
{"etunimi": "Ash", "pokemon":{"tyyppi":y}} <- input
return y
```

Jolloin jia-palauttaa

```
...
"RESULT PRINT"
"sahko"
```

Esimerkki2 kysymykset ja vastaukset

HUOM! Kaikki kyselyt eivät välttämättä palauta mitään HUOM2! Tässä vaiheessa kannattaa kopioida esimerkki2.json `jsonTestaukset.json`-sisällöksi! ### E2. teht 1. **Kirjoita** Jia-koodi, joka palauttaa kouluttajan, jonka `sukunimi` on "Ketchum" `pokemonin`-elementin `laji` sisältö.

Vastaus:

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E2T1
```

E2. teht 2.

Kirjoita Jia-koodi, joka palauttaa `pokemonin`, jonka `laji` on "Pikachu" `vari`:n.

Vastaus:

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E2T2
```

E2. teht 3.

Kirjoita Jia-koodi, joka palauttaa niiden `pokemon`:ien `tyyppi`:n joiden `heikkous` on “jaa”.

Vastaus:

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E2T3
```

Esimerkki 3

```
{
    "nimi": "Muumipeikko",
    "sukupuoli": "poika",
    "ika": 10,
    "perhe": {
        "isa": "Muumipappa",
        "aiti": "Muumimamma"
    },
    "tuttavat":{
        "tyttoystava": "Niiskuneiti",
        "ystava": "Nuuskamuikkunen",
        "tuttu": "Nipsu",
        "kaveri": "Pikku Myy"
    }
}
```

HUOM! sekä että ehtoja että palautusarvoja pystyy määrittelemään useita!

Esimerkki3 kysymykset & vastaukset

HUOM2! Tässä vaiheessa kannattaa kopioida esimerkki3.json `jsonTestaukset.json`-sisällöksi!

Kirjoita Jia-koodi, joka palauttaa `nimi`-elementin kun `perhe`-elementin arvo `isa` on “Muumipappa” ja `tuttavat`-lementin arvo `tuttu` on “Nipsu”

Vastaus: ### E3. teht 1.

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E3T1
```

E3. teht 2.

Kirjoita Jia-koodi, joka palauttaa `tuttavat`-elementin alla olevan `tuttu`-arvon, jos `tyttoystava` on “Niiskuneiti” `kaveri` on “Pikku Myy” ja ylemmän tason `sukupuoli` on “poika”

Vastaus:

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E3T2
```

E3. teht 3.

Kirjoita Jia-koodi joka palauttaa `nimi`-elementin arvon kun `ika` on 10. *Huom. tässä ika ei ole merkkijono vaan numero.*

Vastaus:

```
COPY PASTE OMA JIA-KOODI TÄMÄN TILALLE E3T3
```

Loppukysymykset

Vastaa vapaamuotoisesti tai esimerkiksi ranskalaisilla viivoilla ajatuksia Jian syntaksista ja toiminnasta. Halutessasi voit käyttää seuraavia kysymyksiä pohjana

Mikä oli vaikeaa tai epäintuitiivista?

Mikä oli hyvää?

Muita kommentteja?