

# Ohjelmointikielet ja -paradigmat 2017

Sakari Snäll 507267  
Antti Vainikka 510575

May 2, 2017

## 1 Johdanto

Toteutimme harjoitustyön kolmannen tehtävän. Valitsimme toteutettavaksi peliksi pokerin koska järkevän toteutuksen tekeminen tutustuttaisi meidät kielen tapaan luoda struktuureita, lukea syötettä standardista syöteputkesta ja joutuisimme toteuttamaan pelijatkumon funktionaalisesti. Alkuperäisen tietämyksemme perusteella pelin ohjelmoiminen on funktionaalisesti vaikeampaa verrattuna imperatiiviseen tapaan ohjelmoida koska peleissä on tila, jota pitää muuttaa usein. Useat pelienohjelmointiin liittyvät kirjat käyttävät olio-ohjelmoinnin paradigmaa, sillä pelin osat ovat yleensä helpompia hahmottaa kun ne jaetaan yksittäisiin olioihin, joilla on jokin tietty funktio. Ohjelmointityö antaa meille myös tuntumaa miksi funktionaalinen paradigma ei ole suosittu ainakaan pelien ohjelmoinnissa.

## 2 Funktionaalinen paradigma

Funktionaalinen paradigma on deklarativisten ohjelmointiparadigmojen luokkaan kuuluva ohjelmointiparadigma. Funktionaalisen paradigman ominaisuuksia ovat esimerkiksi muuttumattomat tietorakenteet ja laiska laskenta. Laiskalla laskennalla tarkoitetaan sitä, että mitään ei lasketa ennen kuin on pakko ja mitään ei lasketa kahdesti.

## 2.1 Matemaattinen lähtökohta

Funktionaalisessa ohjelmoinnissa pyritään kuvaamaan ongelma matemaattisesta näkökulmasta käyttäen funktioita ja lausekkeita. Funktiot pyritään pitämään tarkkaan määriteltynä ja sivuvaikutuksettomina, eli sellaisina etteivät ne ole riippuvaisia globaalista datasta ja tuottavat aina saman lopputuloksen samoilla lähtöarvoilla.

Funktionaalisen ohjelmoinnin matemaattisuus ilmenee myös siten, että funktioiden suoritusjärjestys ohjataan rekursioiden ja ehtojen avulla, kun taas imperatiivisessa ohjelmoinnissa ohjaus on peräkkäin ja iteratiivisesti. Funktionaalisilla ohjelmilla ei ole varsinaisesti tilaa kuten proseduraalisissa ohjelmissa, vaan niiden kontrollivuo muodostuu suoraan lausekkeista, funktiokutsuista ja niiden välisistä riippuvuuksista.

## 2.2 Funktionaaliset ohjelmointikielet

On vaikea määrittää onko ohjelmointikieli funktionaalinen vai ei. Useimmissa ohjelmointikielissä on olemassa mahdollisuus luoda anonyymeja funktioita ja sen pohjalta on vedetty johtopäätös, että kieli olisi funktionaalinen. Esimerkiksi Java8 kutsutaan funktionaaliseksi koska siihen lisättiin lyhenne paikallaan luodulle objektille, joka toteuttaa jonkin rajapinnan. Funktionaaliset ohjelmointikielet voidaan jakaa kuitenkin puhtaasti funktionaalsiin ja epäpuhtaisiin. Puhtaus funktionaalisissa ohjelmointikielissä viittaa siihen voidaan kielessä toteuttaa sivuvaikutuksia. Täysin funktionaalisia ohjelmointikieliä on esimerkiksi Haskell ja Clear. ML pohjaiset kielet ovat funktionaalisia mutta esimerkiksi standardissa ML:ssä ja Ocamlissa voidaan asettaa muuttujalle arvo. Jos halutaan olla tarkkoja niin myös Haskellissa tapahtuu sivuvaikutuksia koska esimerkiksi tekstin tulostaminen sisältää sivuvaikutuksen. Täydellisesti sivuvaikutuksellisessa ohjelmassa ei kuitenkaan ole järkeä koska tarpeeksi tehokas kääntäjä voisi tuottaa ohjelman, joka ei suorita mitään koska sivuvaikutuksia ei tehdä.

```
(let loop ((n 0))
  (cond
    ((= n 10))
    (#t
     .
     (loop (+ n 1)))))
```

```
for(int n = 0; n < 10; n++) {  
    ...  
}
```

## 2.3 Sulkeumat objektien sijaan

Ohjelmointikielissä joissa funktioon voidaan sisällyttää ympäristö(closure), pystytään triviaalisti toteuttamaan objektisysteemi. Yksi mahdollinen tapa toteuttaa objektisysteemi olisi tehdä objektin luova funktio, joka palauttaisi funktion, joka ottaa listan parametreja. Listan ensimmäinen parametri voisi olla metodi ja loput metodin vaatimia parametreja.

## 2.4 Mahdolliset vahvuudet

Funktionaalisen ohjelmoinnin suosio on ollut suuressa kasvussa, sillä prosessoritymien määrä on kasvanut ja funktionaalisella ohjelmoinnilla ohjelmakoodia on helpompi rinnakkaistaa ja tehostaa. Kun funktioilla ei ole sivuvaikutuksia, on niiden suoritusjärjestys vapaa ja kääntäjä saa optimoitua samanlaiset funktiokutsut yhdeksi. Rinnakkaisuus on mutkattomampaa kuin imperatiivisessa ohjelmoinnissa, sillä funktioilla ei ole yhteistä globaalia dataa, joka voisi aiheuttaa ongelmia.

## 2.5 Mahdolliset heikkoudet

Funktionaalisessa ohjelmoinnissa yritetään välttää globaalia dataa, minkä takia ohjelmoidessa täytyy miettiä tarkemmin suoritusjärjestystä ja putkittaa kutsuja. Tosimaailman asioiden hahmottaminen voi olla vaikeampaa olioiden puuttumisen takia, mutta matemaattisten ongelmien ratkaisu on virtaviivaisempaa. Funktionaaliset ohjelmointikielet ovat usein raskaampia suorittaa kuin imperatiiviset, mikä johtuu monissa tapauksissa mutatoitumattomista muuttujista ja niiden tehottomuudesta nykytietokoneissa.

# 3 Lisp

Lispejä pidetään yleensä funktionaalisina mutta ne ovat todellisuudessa moniparadigmaisia. Common Lisp on tunnetuista Lispeistä tulkintamme mukaan

vähiten funktionaalinen koska sen standardi ei edellytä häntärekursion optimointia. Common Lispissä on myös useita sivuvaikutuksellisia funktioita. Schemessä on myös sivuvaikutuksellisia funktioita mutta ainakin standardin määrittävät sivuvaikutukselliset funktiot päättyvät aina huutomerkkiin. Schemen standardi myös määrittää, että toteutuksen täytyy tukea häntärekursion optimointia. Schemen keskustelufoorumeilla myös kokemuksen perusteella suositaan enemmän rekursion käyttöä, toisin kuin Common Lispin ohjelmoijat, jotka tuntuvat hyödyntävän enemmän imperatiivisia silmukka-rakennetta.

### 3.1 Clojure

Clojure on ohjelmointikieli, joka on saanut paljon vaikutteita Common Lispistä. Clojure ei vältämättä tue häntärekursion optimointia. Clojure kuitenkin eroaa Schemestä ja Common Lispistä siinä, että Clojuressa perus datarakenteet ovat muuttumattomia.

Muut Lisp-ohjelmoijat ovat usein kriittisiä Clojuren suhteen koska siinä käytetään kaikkia eri tyyppisiä sulkeita pelkkien tavallisten sulkeiden sijaan.

### 3.2 Funktionaalinen Clojure

## 4 Harjoitustyö

Tehtävänannossa mainittiin, että ohjelmakoodin tulisi olla idiomaattista valitulle ohjelmointikielelle. Emme ole täysin varmoja onko ohjelmakoodi täysin idiomaaattista mutta koska työn oli tarkoitus olla harjoitus funktionaaliseen paradigmaan, toteutimme koodin niin, että siinä ei muuteta muuttujia olenkaan. Valitsimme Clojuren ohjelmointikieleksi koska ML:n kaltaisissa ohjelmointikielissä on liian vaikea syntaksi.

### 4.1 Funktionaalisuuden hyödyt ja haitat

Mielestämme ohjelmakoodin toteuttaminen funktionaalisesti oli vaikeampaa ja hitaampaa, kuin jos olisimme käyttäneet tavanomaisempaa paradigmaa. Jo pelkästään silmukkarakenteiden tekeminen vaati paljon enemmän miettimistä.

Peltilaa ei ylläpidetä globaaleissa muuttujissa vaan se kulkee funktioiden parametreinä. Tämän takia funktioiden rakentaminen oli hieman puhtaampaa koska mikään muu funktio ei voinut muuttaa tilaa joka oli saatu parametrinä.