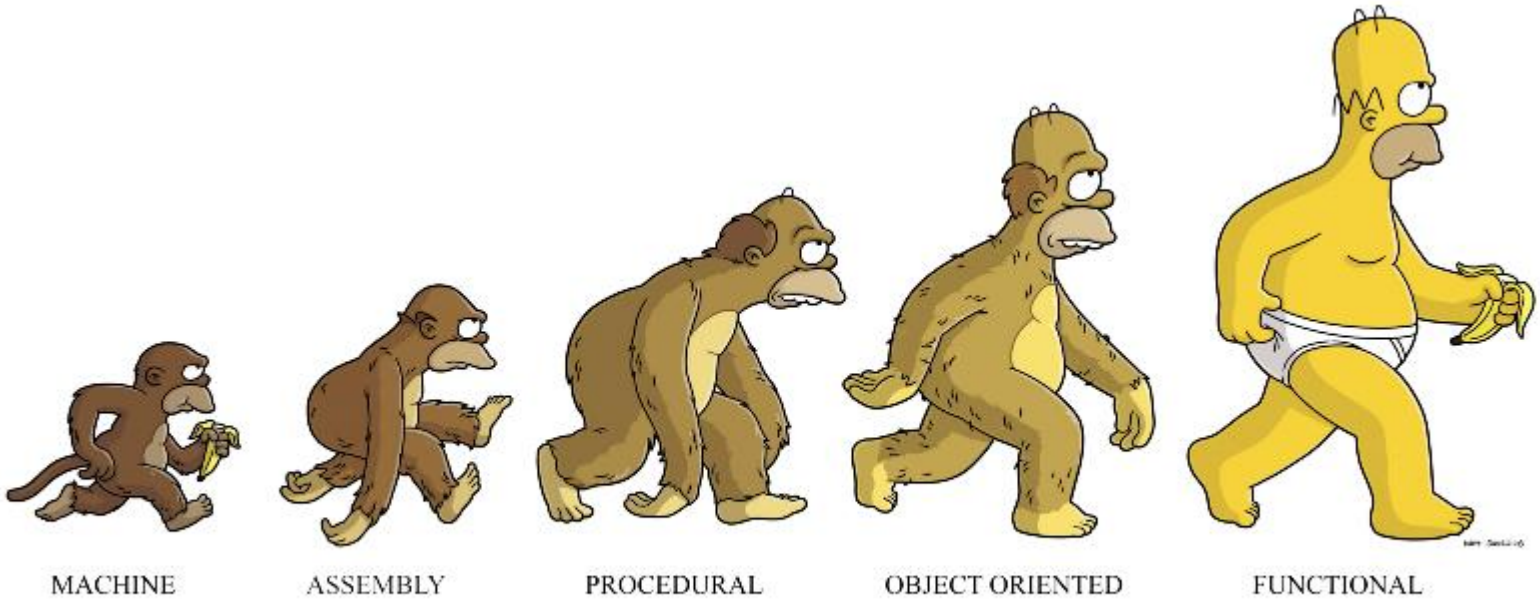
# Funktionaalisten kielten oppimisesta ja valinnasta

Tampere 24.1.2018

# Kehitystä?



# Tarve funktionaaliselle ohjelmoinnille - motivaatio

- Rinnakkaisen suorittamisen lisääntyminen
  - Immutability –vaatimus
- Oliokieliä ei käytetä niin kuin joskus ajateltiin
  - Luokat lähinnä datan väliaikaiseen kuvaamiseen
  - Ei varsinaista uudelleenkäyttöä perimällä
  - Käytännössä palvelimen kirjoittaminen on POJOjen kirjoittamista ja datan kuljettaminen niiden kautta selaimeen ja tietovarastoon (ORM)
  - Tyypillinen spring/hibernate softa
    - Set, get
- Laiska evaluointi säästää resursseja

# Funktonaalisuus kiteytettynä muuttujien kannalta

```
a = 0
def increment():
    global a
    a += 1
```

Mutable data

```
def increment(a):
    return a + 1
```

Immutable data

# Java 9

- Lambda-lausekkeet
  - Välitetään funktio parametrina
    - Korkeamman asteen funktiot
  - Palautetaan funktio funktiosta
- Immutable stream
  - Sivuvaikutusten välttämiseksi
- Optionals
  - Null pointterien välttämiseksi
- Map, filter, reduce

# Funktionaalisen ohjelmoinnin hyödyt

- Paljon vähemmän koodia
  - Esim. 50%
- Nopeuttaa myös uuden tekijän “sisäänajoa”
  - Koska kokonaiskoodimäärä on pienempi
- Kehittäminen nopeampaa
- Yksinkertaisemmat ratkaisut, joita on helpompi ymmärtää
  - Luokkahierarkiat ja viestinvälitys synnyttävät paljon ajonaikaisia yhteyksiä olioiden välille, joiden perässä pysyminen vaikeaa
- Moniperintä ja korket luokkahierarkiat
  - Vaikea hallita -> usein päädytään kompositioon

# Milloin fp ei ole hyvä vaihtoehto

- Ohjelman toiminta on lähinnä **tilan muuttamista**
  - Ohjelman osien välinen **kommunikointi** tapahtuu **tilan muutosten** avulla
- Matalan tason ohjelmointi
  - Tilan muuttamista

# Mihin se sopii?

- Rinnakkainen ohjelmointi
  - Tilattomuus
  - 4 CPU:ta rinnan
- DSL
  - Helppoja kirjoittaa esim. Clojurella
- Matemaattispainotteiset algoritmit
  - Koodia on helpompi lukea kuin vastaaavaa proseduraalista koodia
- Kun halutaan välttää bugeja
  - Optional, Maybe
  - Vähemmän koodia
  - Sofistikoituneempi virheenkäsittely (ei pureta pinoa)

# Ruby, Python, Java…

- Näistä löytyy funktionaalisia piirteitä – kaikista
- Miksei siis käyttäisi **map, reduce, filter ja lambdat ja Optional**?
- Problem solved – käytetään vaikka Java 9 versiota ja sen funktionaalisia piirteitä!

# Java, Python, Scala…

- Eivät kuitenkaan ole **olemukseltaan** funktionaalisia kieliä
  - Voi tehdä myös imperatiivisia ratkaisuja
- Funktionaalisella kielellä ohjelmoitaessa **tapa ajatella ongelman ratkaisusta** eroaa merkittävästi imperatiivista tavasta ratkaista ongelma
- Ongelman kuvaus ajattomasti
  - Imperatiivinen tyyli korostaa ajan kulumista ja suorituksen vaiheita
  - Funktionaalinen tyyli korostaa ongelman kuvausta

# Mikä kieli oppimisen kannalta paras?

- Kysy ensin mikä on oppimisen **tavoite?**
  - Oppia käyttämään map ja reduce funktioita?
    - Ja muita korkeamman kertaluvun funktioita?
  - Oppia ”jotain” funktionaalista?
  - Vältellä muuttuvaa dataa, varautua rinnakkaisuuteen/samanaikaisuuteen
    - Immutability, pure functions
  - Oppia uusi tapa ajatella ongelman ratkaisua?
    - Oppia uusi paradigma
  - Oppia käyttämään esim. ClojureScriptin re-frame/reagent kirjastoja?
    - Oppia eräs funktionaaliseen ohjelmointiin läheisesti liittyvä ratkaisu tiettyyn domain spesifiin ongelmaan – selaimen DOM puun tilan hallinta
  - Käyttää fp-ominaisuuksia sisältävää kieltä heti käytännön toteutuksissa (esim. verkkopalvelin + selain)
    - Esim. Scala, Clojure, F#
  - Jokin muu, mikä?

# Tavoite - **paradigman oppiminen**

- Erinomainen kieli **Haskell**
  - Pakottaa funktionaaliseen ajatteluun
    - Mikä on paljon muutakin kuin map ja reduce
  - IO-operaatiot eristetty ”puhtaasta” koodista IO -monadeilla
    - Esim. satunnaislukujen luominen, file IO, jne.
    - IO-operaatioita kömpelöjä käyttää
  - Dokumentaatiossa toivomisen varaa
  - Huikea tyyppijärjestelmä
  - Käytetään jossain määrin ”oikeastikin”
- Myös ML (Meta Language)

# Tavoite – **fp-ominaisuuksien hyödyntäminen**

- Korkeamman kertaluvun funktiot
  - Ruby, Java 8->, Python, JavaScript, C++ ->11, Elixir, Erlang, Go, Elixir, D, Dart, Scala, Swift, Smalltalk, PHP, R, Clojure, F#, OCaml, C#, Haskell..
- Siis lähes kaikilla kielillä voi ohjelmoida korkeamman kertaluvun funktioilla
  - ”**operointi** funktionaalisesti”
  - Tyyli imperatiivinen

# Välimalli - tavoite – epäpuhtaan koodin vähentäminen

- Välimallin valinta
  - https://www.youtube.com/watch?v=MXiIV_ecS88
- Clojure, Scala…
  - Käytetään ”oikeasti”
  - Koodi voi olla hyvinkin ”puhdasta” (pure)
    - Pohdinta, mikä pitää toteuttaa puhtaasti ja missä voi olla epäpuhtauksia
  - Myös selaimessa scala.js ja ClojureScript
- Käytännöllisin ratkaisu arkitarpeisiin
  - Tarvitaanko olioita?
    - Refaktorointitarpeet (funktioiden polymorfismi (clj), luokat->oliot->perintä(scala)

# Scala

- Martin Odersky, 2004
  - École Polytechnique Fédérale de Lausanne
  - https://www.scala-lang.org/documentation/learn.html
  - “Mutability kuuluu kuvioon, mutta sitä voi myös vältellä”
  - Hyvällä tavalla pragmaattinen
- Oliokieli, missä funktionaalinen
  - Shapeless etc. “pure functional” kirjastot
  - Lens (mm. kopiointi mutaation sijaan)
- Helppo “kokeilla” ScalaFiddle
  - Tamperelaista tekoa, Otto Chrons.
- JVM-pohjainen, käytetään oikeassa elämässä
  - CLR-versiota ei päivitetä
- Scala.js -> JavaScript

# Clojure

- Rick Hickey, 2007
- Helppo laatia DSL makrojen avulla
  - Helppo tehdä kokonaan “uusi” kieli
  - https://www.braveclojure.com/, pragmaattinen kirjanen
- Käytetään oikeassa elämässä
  - Työkalut keskeneräisiä ja vaiheessa
- Osaamistoive aika monessa työpaikkailmoituksessa
  - Mutta kuitenkaan itse clojure-**kehittäjiä** ei etsitä vrt. scala
- JVM-pohjainen, myös CLR ja JavaScript
  - ClojureScript

# F#

- Don Syme, 2005
- Pyörii CLR:n päällä
- Yhteensopiva C#:n kassa
  - C# sis. Myös fp-ominaisuuksia
  - LINQ
- Walmart investoi vuonna 2016 3.3 mrd taalaa start-upiin, jonka teknologia on F# pohjainen (jet.com)
  - Kurottava Amazonin etumatkaa umpeen
- Oliot mukana vrt. Scala
- Piirteitä Haskellista, ML:stä ja LINQ:sta

# Funktionaaliset käsitteet

- Kategoriateoria
  - Kategoriateoria näkyy Haskellissa
  - Asian kuvaa erinomaisesti Bartoz Milewski
  - https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/
- Monadi, funktori, applicative –funktori
  - javallakin voi ohjelmoida “monadisesti”
  - Aika hankalaa
    - Esim. https://www.slideshare.net/mariofusco/monadic-java
  - Ja monta muuta…
- Funktionaaliseen ohjelmointiin on paras **funktionaalinen** ohjelmointikieli

# Fp-kieli ensimmäisenä kielenä

- Yläasteen opettajilla erittäin hyviä kokemuksia **Racketista**
  - Hyvin clojuremainen kieli
  - Scheme –tyyppinen
- Racket **BSL** (Beginning Student Language)
  - Rajallinen määrä funktioita
- Kokemuksen mukaan parempi oppia **fp ensin** – helpottaa muiden paradigmojen oppimista myöhemmin
  - **Ei toimi toisin päin!!!**

# Lampun vaihtaminen

Kuinka monta Haskell-ohjelmoijaa tarvitaan vaihtamaan lamppu?

Haskell-ohjelmoijat eivät vaihda lamppua, vaan korvaavat sen. Samalla korvataan myös talo, jossa lamppu on.

# Pari opiskeluvinkkiä

- REPL
  - Vuorovaikutus komentotulkin kanssa
  - Funktioiden testaaminen
- Vanhan unohtaminen
  - Varsinkin, jos haluaa oppia **tavan ajatella** vs. käyttää funktionaalisia ominaisuuksia

# Reality check

- Clojure
  - Ei ole “production ready”
  - Hypeä enemmän, pihviä vähemmän
  - Lisp-maine ei auta
  - Hyvä “porttihuume” funktionaaliseen ohjelmointiin, mutta jos on aikaa opiskella vain yksi kieli, yleiskäyttäisempi ja valmiimpi voi olla parempi, esim. Scala
- Haskell
  - Erinomainen, jos löytyy hyvä käyttötapaus, oma lukunsa
  - Ohjelmien todistaminen helpompaa
  - Matemaattiset sovellukset
  - Teoreettinen
  - Heikot työkalut
- F#
  - Microsoftin työkalut, CLR, C# yhteensopiva
  - Käytetään oikeasti, esim. case Walmart
  - Finanssimaailman sovellukset – pitää laskea oikein
  - Azure functions tuki maaliskussa 2017 (vrt. Amazon Lambda, serverless)
- Scala
  - Lupaavin 2000-luvun uusista kielistä
  - Tukee olio-ohjelmointia
  - Vahvat backend kirjastot (mm. Play, Akka)
  - Vahvat fp-kirjastot (scalaz, cats etc.)
  - Pragmaattinen

# Linkit

- Kategoriateoria (jämäkkää teoriaa)
  - https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/
  - Videosarja: https://www.youtube.com/watch?v=I8LbkfSSR58
- Functional design patterns (yleistajuinen)
  - https://www.youtube.com/watch?v=srQt1NAHYC0
- Odersky fp:n ja olio-ohjelmoinnin eroista ja kuinka Scala vastaa tarpeeseen – modulaarisuus (yleistajuinen)
  - https://www.youtube.com/watch?v=K_g_xUtydpg

# Linkit

- Haskell-kirja (kohtuullisen helppotajuinen)
  - http://learnyouahaskell.com/
- “Why functional programming matters” (teoreettinen)
  - http://www.cse.chalmers.se/~rjmh/Papers/whyfp.pdf
- Oderskyn Scala-kurssit Courserassa
  - https://www.coursera.org/specializations/scala
- Clojure-kirja (helppotajuinen)
  - https://www.braveclojure.com/
- ScalaFiddle (kokeile Scalaa)
  - https://scalafiddle.io/
  - Huom! Tässä myös scala.js vinkit, selaimen ja backendin voi molemmat laatia Scalalla

# Linkit

- F# (kokeile F#)
  - http://www.tryfsharp.org/Learn
- Racket (opettajan materiaali harjoituksineen)
  - http://racket.koodiaapinen.fi/

# Koulutukset ja legacy-koodin kirjoittaminen uudelleen

- Räätälöidyt toimialaan sovitetut koulutukset funktionaalisesta ohjelmoinnista tai ohjelmistokomponenttien uudelleensuunnittelu funktionaalisesti
- Funktionaalisen ohjelmoinnin soveltuvuuskonsultointi
  - Sopiiko valitun toimialan ongelman ratkaisuun ja millä edellytyksillä
- juuso.vuorinen@ideallearning.fi