

Pragmaattisen funktionaalisen ohjelmoinnin arviointi

Insinöörityö

Artu Pennanen 25.11.2024

Järjestys

0. L^AT_EX
1. Aihe
2. Funktioaalinen ohjelmointi
3. Kaikukammion räjäytys
4. Mallintaminen
5. Lopetus

0. L^AT_EX

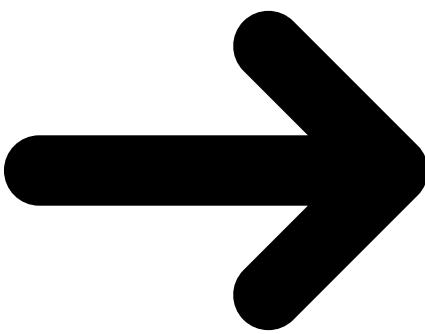
LATEX



LATEX

LATEX

.tex tekstitiedostoja



PDF

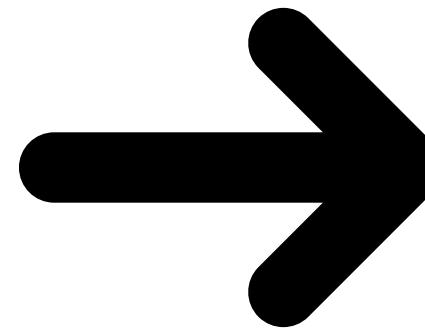
```
404
405 \subsection{Yleiset monadit}
406
407 \texttt{Promise, Maybe, Either, Result, State, IO, ja Lista}
  (Array). Nämä ovat esimerkkejä yleisesti käytetyistä
  monadeista. \citet{monad_wikipedia,
  bartosz_category_for_progamers_10}
408
```

3.3.1 Yleiset monadit

Promise, Maybe, Either, Result, State, IO, ja Lista (*Array*). Nämä ovat esimerkkejä yleisesti käytetyistä monadeista. [52; 53.]

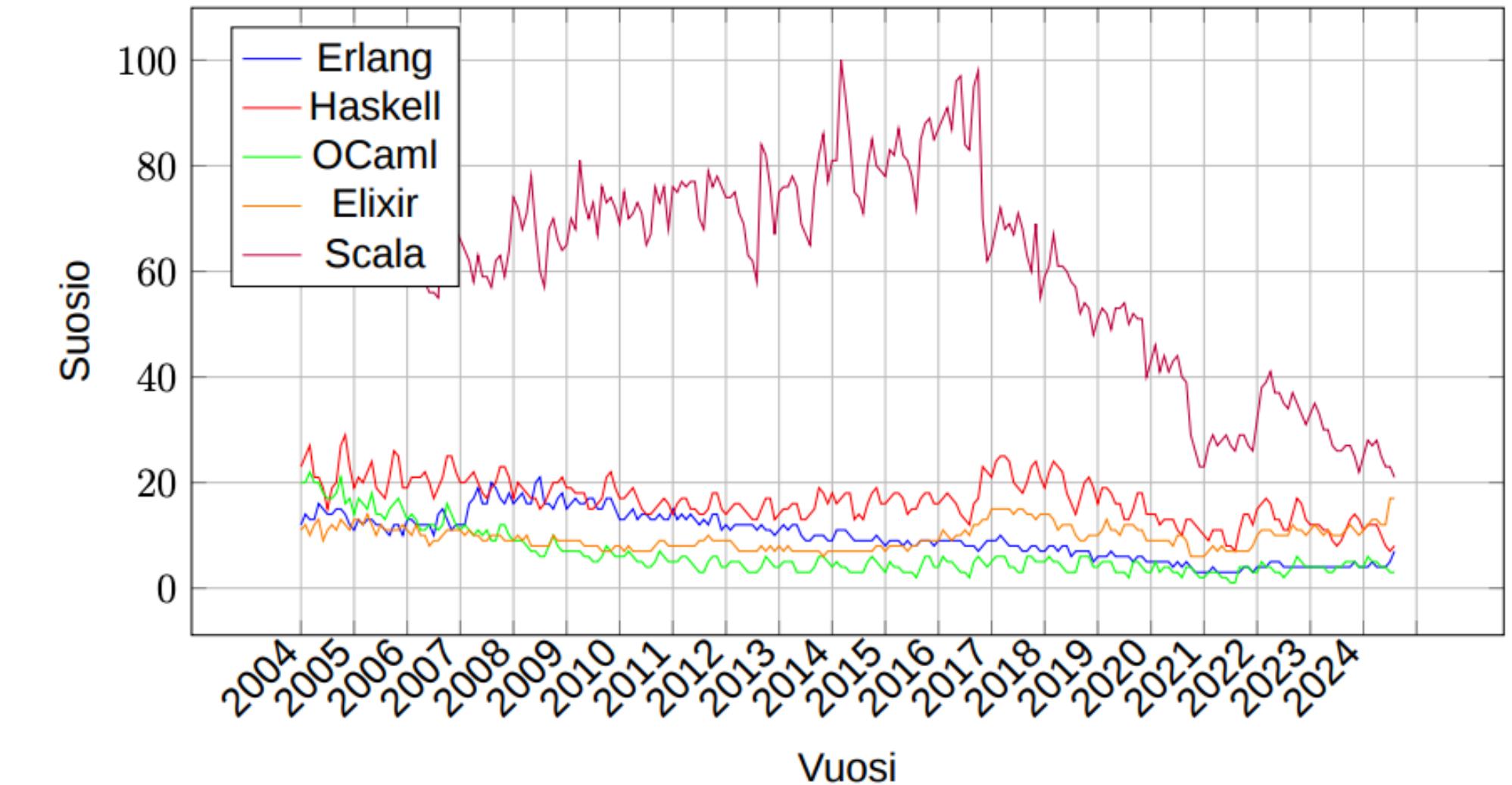
LATEX

.tex tekstitiedostoja



PDF

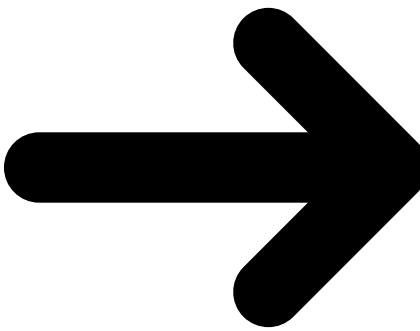
```
20
You, 4 weeks ago | 1 author (You)
21 ✓ \begin{figure}[htbp]
22     \pgfplotstableread[col sep=comma]{data/fp_lang_popularity.csv}\datatable
23     \pgfplotstablegetrowsof{\datatable}
24     \pgfmathtruncatemacro{\rows}{\pgfplotsretval-1}
25     \centering
26     \begin{tikzpicture}
27         \begin{axis}[
28             width=\linewidth, height=8cm, ylabel={Suosio}, xlabel={Vuosi}, xtick={0,12, ..., \rows},
29             xticklabel style={rotate=45, anchor=east}, xticklabels={2004, 2005, 2006, ..., 2024},
30             legend pos=north west, grid=major, cycle list name=color list
31         ]
32         \addplot [mark=none, color=blue] table [x expr=\coordindex, y=Erlang] {\datatable};
33         \addplot [mark=none, color=red] table [x expr=\coordindex, y=Haskell] {\datatable};
34         \addplot [mark=none, color=green] table [x expr=\coordindex, y=OCaml] {\datatable};
35         \addplot [mark=none, color=orange] table [x expr=\coordindex, y=Elixir] {\datatable};
36         \addplot [mark=none, color=purple] table [x expr=\coordindex, y=Scala] {\datatable};
37         \legend{Erlang, Haskell, OCaml, Elixir, Scala}
38     \end{axis}
39 \end{tikzpicture}
40 \caption{Funktionaalisten ohjelmointikielien Erlangin, Haskellin, OCamlin, Elixirin ja Scalan suosiokehitys vuosina 2004–2024. Kielet eivät ole olleet suuressa nousussa. \citep{fplanggoogletrend}}
41 \label{fig:fplangpopularity}
42 \end{figure}
```



Kuva 1: Funktionaalisten ohjelmointikielien Erlangin, Haskellin, OCamlin, Elixirin ja Scalan suosiokehitys vuosina 2004–2024. Kielet eivät ole olleet suuressa nousussa. [13.]

LATEX

.tex tekstitiedostoja



PDF

```
91 \glsdisp{composed_function}{Yhdistetty funktio} tarkoittaa yksinkertaisesti kahden, tai  
useamman, funktion yhdistämistä siten, että yhden funktion tulos syötetään seuraavalle.  
Esimerkiksi koodiesimerkin \ref{code:javascript_manual_composition} funktio $h$ on  
funktioiden $f$ ja $g$ yhdiste. Ajaessa funktio $h$, suoritetaan ensin $g$, jonka  
palautusarvo annetaan funktiolle $f$. Kaksi funktiota yhdistämällä on saatu yksi uusi  
funktio.  
  
92  
93 \begin{code}  
94 |   \begin{minted}[language=javascript]  
95 const f = x => 2 * x  
96  
97 const g = x => x + 3  
98  
99 const h = x => f(g(x))  
100 \end{minted}  
101 | \caption{JavaScript-esimerkki yhdistetystä funktiosta h ilman pipe- tai  
102 | compose-funktiota}  
103 | \label{code:javascript_manual_composition}  
104 \end{code}  
  
105 Funktionaalissa ohjelmointikielissä yhdistettyjä funktioita pystyy usein kirjoittamaan  
käyttäen kieleen sisäänrakennettuja operaattoreja, joilla funktioiden yhdistäminen on  
helppoa ja suoraan osana ohjelmointikieltä \cite{fsharpcomposition,haskellcomposition}.
```

Yhdistetty funktio tarkoittaa yksinkertaisesti kahden, tai useamman, funktion yhdistämistä siten, että yhden funktion tulos syötetään seuraavalle. Esimerkiksi koodiesimerkin 4 funktio `h` on funktioiden f ja g yhdiste. Ajaessa funktio `h`, suoritetaan ensin `g`, jonka palautusarvo annetaan funktiolle `f`. Kaksi funktiota yhdistämällä on saatu yksi uusi funktio.

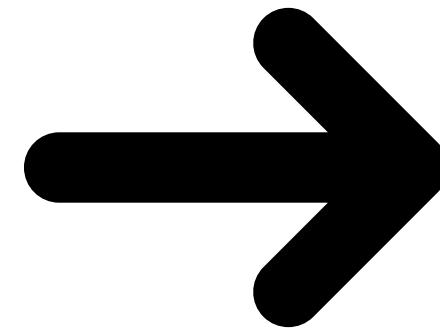
```
1 const f = x => 2 * x  
2  
3 const g = x => x + 3  
4  
5 const h = x => f(g(x))
```

Koodiesimerkki 4: JavaScript-esimerkki yhdistetystä funktiosta `h` ilman pipe- tai compose-funktiota

Funktionaalissa ohjelmointikielissä yhdistettyjä funktioita pystyy usein kirjoittamaan käyttäen kieleen sisäänrakennettuja operaattoreja, joilla funktioiden yhdis-

LATEX

lähteet .bib tiedostossa



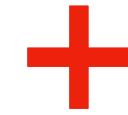
PDF

```
You, last month | 1 author (You)
710 ✓ @online{against_self_closing_tags,
711   url      = {https://jakearchibald.com/2023/
712     against-self-closing-tags-in-html/},
713   title    = {The case against self-closing tags in HTML},
714   author   = {Jake Archibald},
715   year     = 2023,
716   urldate  = {2024-09-14}
717 }
```

```
You, last month | 1 author (You)
718 ✓ @misc{viljofruits,
719   author = {Viljo Pennanen},
720   title  = {Hedelmät},
721   year   = 2024,
722   month  = 9,
723   day    = 15,
724   note   = {Kuvatiedostoon saatu oikeudet suoraan tekijältä.}
725 }
```

```
You, 4 weeks ago | 1 author (You)
727 ✓ @article{practicaltheory,
728   author = {Arthur G. Bedeian},
729   year   = {2016}.
```

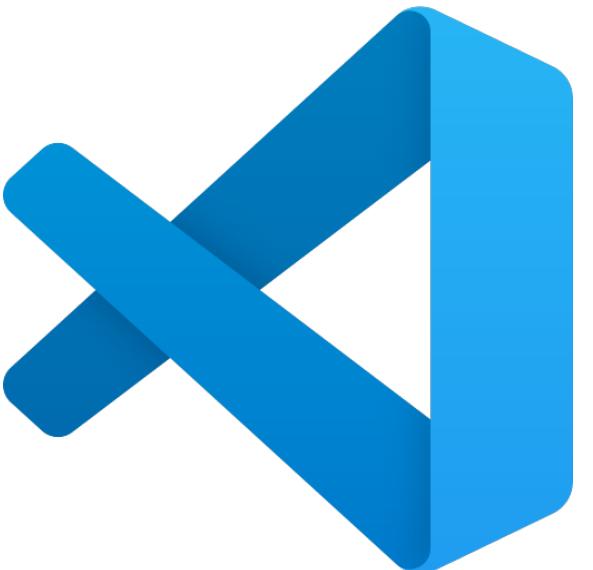
Hän suosii for-silmukoiden käyttämistä funktionaalisen `reduce`-funktion sijasta, ja perustelee valintaa luettavuudella [34; 57; 77]. Yleisesti Archibald kuitenkin kookee funktionaalisen ohjelmoinnin olevan tuonut positiivisia vaikuttavia JavaScriptiin [77]. Luettavuuden kannalta merkittävimpää tekijöitä ovat keskimääräinen riviin pituus, tunnisteiden määrä per rivi ja keskimääräinen sulkeiden määrä [78, s. 8].



Lähteet

- 1 TypeScript: JavaScript With Syntax For Types 2024. Microsoft. Verkkokauneisto. <<https://www.typescriptlang.org/>>. Luettu 09.09.2024.
- 2 Stack Overflow. 2008. What is a monad? Verkkokauneisto. <<https://stackoverflow.com/questions/44965/what-is-a-monad>>. Luettu 29.08.2024.
- 3 — 2017. Why are promises monads? Verkkokauneisto. <<https://stackoverflow.com/questions/45712106/why-are-promises-monads>>. Luettu 29.08.2024.
- 4 Miller, Paul et al. 2013. Incorporate monads and category theory. Promi-

LAT_EX



Overleaf

pennane / bachelors-thesis

Type / to search | + ⌂ ⌂ ⌂ ⌂ ⌂

Code Issues Pull requests Actions Projects Security Insights Settings

bachelors-thesis Public Pin Unwatch Fork Star

main Go to file + Code About

Evaluating Pragmatic Application of Functional Programming
www.theseus.fi/handle/10024/869576

Readme View license Activity 0 stars 1 watching 0 forks

Languages TeX 98.8% Shell 1.2%

File/Folder	Description	Time
pennane add alt texts later	✓	7afc1ff · 5 days ago
.github/workflows	revert new ci flow	2 months ago
.vscode	add not about biber blowing up	last month
chapters	add alt texts later	5 days ago
code	rm the	3 weeks ago
data	rm example files & move other files	3 months ago
figures	crazy monad graphic	last month
style	add alt texts later	5 days ago
.gitignore	motivation chapter	3 months ago
LICENSE.md	license as md	3 months ago
README.md	rm the	3 weeks ago
biblio.bib	cites & reword summary	last month
compile.sh	faster compilation	3 months ago
compile_all.sh	bring back compile_all.sh	last month
main.tex	add summary and fix stuff	last month

README License

1. Pragmaattisen funktionaalisen ohjelmoinnin arviointi

wtf??

$$\begin{array}{ccccc}
 \lim_{\mathcal{A}_{i+1}} X & \xrightarrow{s} & \lim_{\mathcal{A}'_{i+1}} X & & \\
 \downarrow u & \searrow a & \downarrow c & \nearrow \beta & \downarrow s' \\
 \lim_{\mathcal{A}_{i+1}} Y & \xleftarrow{d} & Q & & \lim_{\mathcal{A}'_{i+1}} Y \\
 \downarrow g & & \downarrow v & & \downarrow v' \\
 \prod_{(G\alpha \rightarrow \beta) \in T_{i+1}} X_\beta & \xrightarrow{t} & \prod_{\partial(\beta \downarrow \mathcal{D})} \lim_{\mathcal{A}_{i+1}} X & & \prod_{\partial(\beta \downarrow \mathcal{D})} \lim_{\mathcal{A}'_{i+1}} Y \\
 \downarrow \gamma & \searrow b & \downarrow e & \nearrow f & \downarrow t' \\
 \prod_{(G\alpha \rightarrow \beta) \in T_{i+1}} Y_\beta & \xrightarrow{u'} & R & \xrightarrow{f} & \prod_{\partial(\beta \downarrow \mathcal{D})} \lim_{\mathcal{A}'_{i+1}} Y
 \end{array}$$



1. Pragmaattisen funktionaalisen ohjelmoinnin arviointi

Pragmatistin mielestä kaikki hyödyllinen tai toimiva ei välittömästi totta, vaan totuus määräytyy sen mukaan, mikä pitkällä aikavälillä tuottaa eniten hyötyä.



- > Käytännönläheinen ei ole suora synonymi
- > Pyritään eniten hyödylliseen ratkaisuun
- > Teoreettisuus on **OK**, jos se tuottaa eniten hyötyä
- > Teoreettisuus on **Ei OK**, jos se tuottaa pelkkää hämminkiä
- > **Esim:** Voi itse päättää, onko pragmaattista käyttää kaikkeen samaa työkalua, ja säästää opiskelussa, vai käyttää aikaa uuden sopivamman työkalun löytämiseen, ja säästää toteuttamisessa.

1. Pragmaattisen **funktioaalisen** **ohjelmoinnin** arviointi

Funktioalinen ohjelointi on [ohjelointiparadigma](#), joka perustuu matemaattisten [funktioiden](#) käyttöön ja tarkemmin [lambdakalkyyliin](#). Puhtaasti funktionaalissa ohjelmissa ei ole lainkaan tilaa eikä siten myöskään sijoituslausesta tai silmukoita.



*Mitä on "ei-funktioalinen" ohjelointi? Jotain mikä ei edes toimi?
tms non-functional :D ?*

- Broidi, kun yritin selittää insinöörityön aihetta sille

1. Pragmaattisen funktionaalisen ohjelmoinnin **arvointi**

- Funktioaalinen ohjelointi on teoreettista.
- Asiakkaat ei ostaa teoriaa, vaan tuotteita.
- Ei ole pragmaattista työskennellä ympäristöstä välittämättä

Arvioinnin kohteena:

Miten funktioaalinen ohjelointi mahtuu mukaan, kun muut eivät ole "FP nautiskelijoita"

Eli miksi?

Teoreettinen tausta on kiinnostava

Etsimässä tapoja abstrahoida

Työpaikalla **FP** kiistanalainen

FP tunkeutuu ohjelointikieliin

Kursseilla lähinnä vain **OOP**

Halusin olla **FP**-sanansaattaja :D

(fp = functional programming = funktionaalinen ohjelointi)

(oop = object oriented programming = olio-ohjelointi)

Lopputulos:

Oppari oli seikkailu
kirjallisuuskatsaus / maisemakuvan arvointi /
näkemysten muodostus
Do's and don'ts

Esitys lähinnä pintaraapaisu. Turhin (valtaosa) teoreettisuus löytyy
raportista.

2. Funktioaalinen ohjelmointi

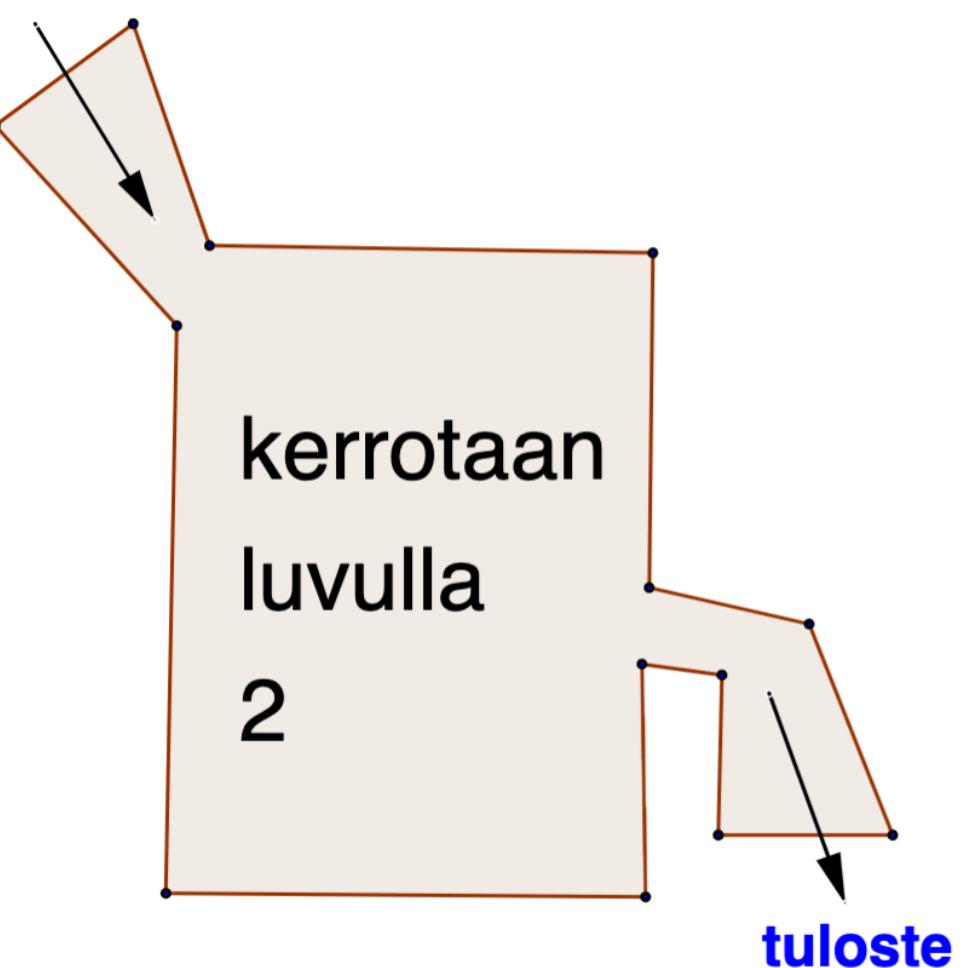
```
import * as R from 'ramda';

const horriblePipeline = R.pipe(
  R.map(R.pipe(R.split(''), R.sortBy(R.identity), R.join(''))),
  R.filter(R.pipe(R.length, R.lt(R.__, 5))),
  R.reduce(R.concat, []),
  R.map(R.pipe(R.multiply(2), R.subtract(R.__, 10))),
  R.uniqBy(R.identity),
  R.map(R.pipe(R.toUpperCase, R.reverse)),
  R.tap(console.log),
  R.slice(1, 4)
);
```

2. Funktionaalinen ohjelmointi

- > EI SIVUVAIKUTUKSIA, EI TILAA
- > VAIN FUNKTIOITA
- > KUIN MATEMATIIKKA!!!
(voi tuoda mukaan ripotellen)

syöte 3, 5, 10



Esimerkin kautta

Aluksi on kivaa; yhtäkkiä kamalaa!

(brace yourselves)

Esimerkin kautta imperatiivinen

```
1  const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
2  const result: number[] = [];
3
4  for (let i = 0; i < numbers.length; i++) {
5    if (numbers[i] % 2 === 0) {          // Check if the number is even
6      result.push(numbers[i] * 2);     // Double and add to result
7    }
8  }
9
10
```

Esimerkin kautta funktionaalinen

```
1  const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3  const result = numbers
4  |  .filter((n) => n % 2 === 0) // Keep even numbers
5  |  .map((n) => n * 2) // Double them
6
7  console.log(result) // [4, 8, 12, 16, 20]
8
```

Esimerkin kautta vielä funktionaalisempi tapa (function composition)

```
1  const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3  const doubleEvenNumbers = pipe(
4    filter(isEven),
5    map(times2)
6  )
7
8  const result = doubleEvenNumbers(numbers)
9
10 console.log(result) // [4, 8, 12, 16, 20]
11
```

Funktioaalinen ohjelmointi oikeasti

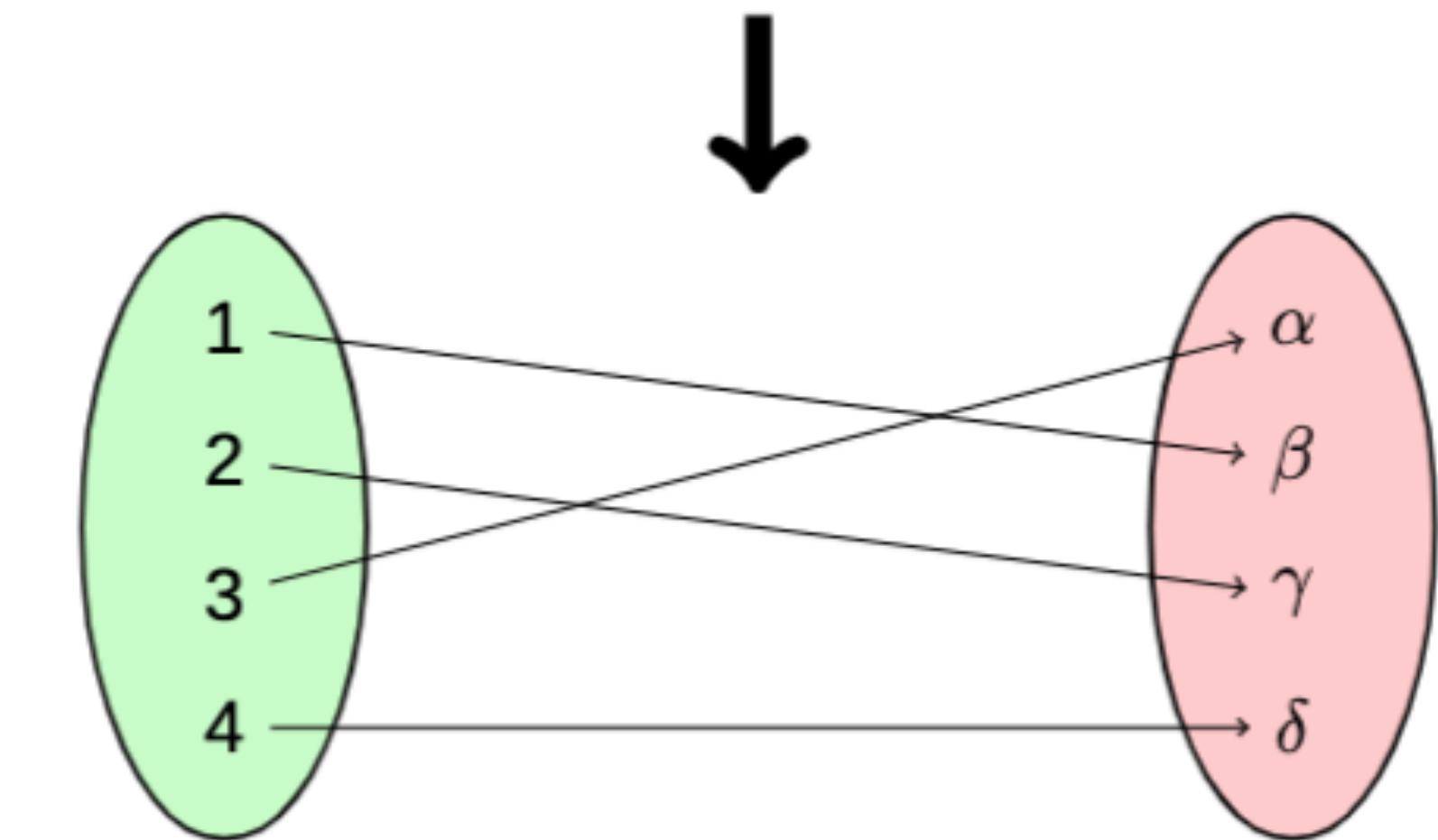
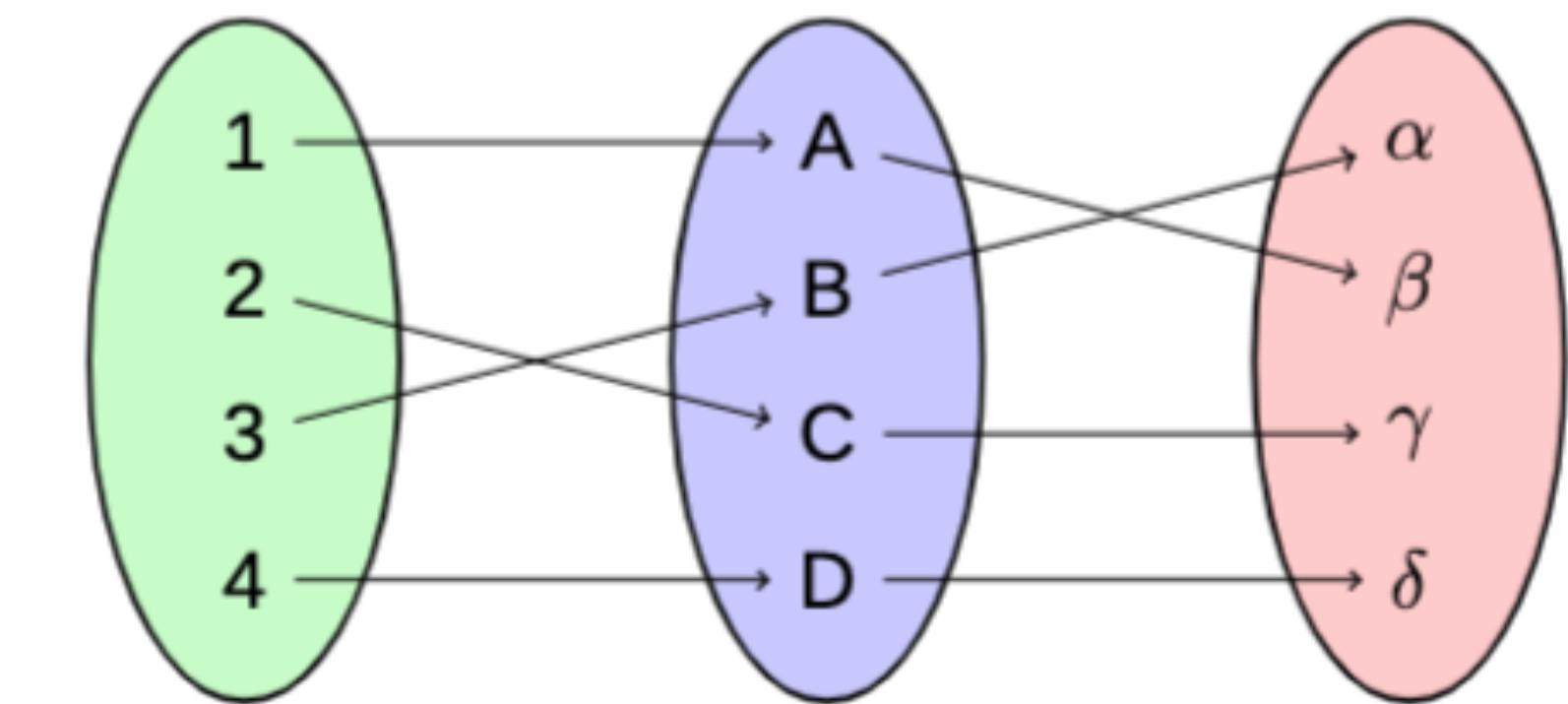
$$A = \{1, 2, 3, 4\} \quad B = \{A, B, C, D\} \quad C = \{\alpha, \beta, \gamma, \delta\}$$

> Funktiot ovat siirtymiä joukoista toisiin

```
// add2 :: number → number
const add2 = (x:number) => x +2

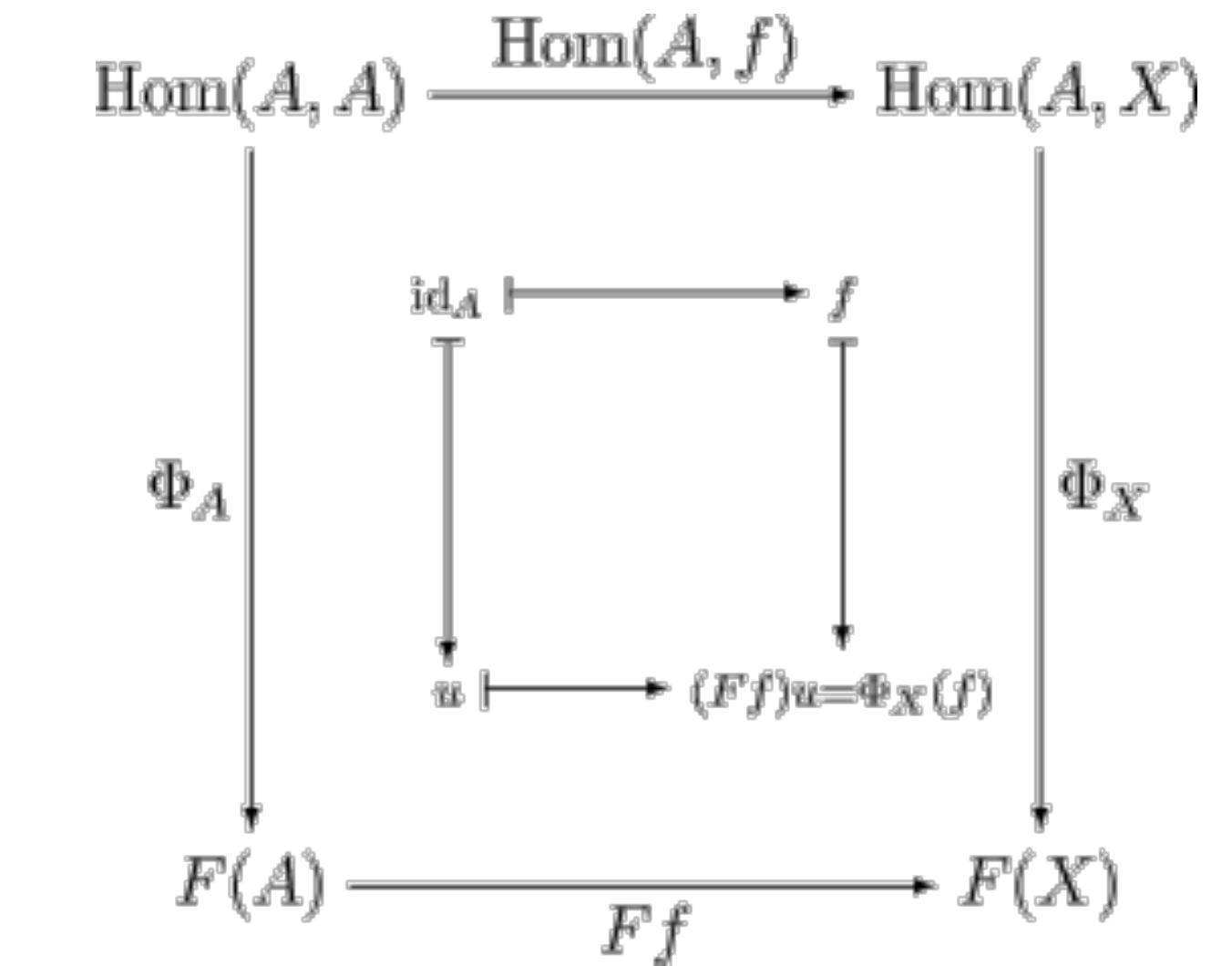
// parseNumber :: string → number / null
const parseNumber = (s: string) => parseFloat(s)

// brewCoffee :: (Water, CoffeeBeans) → Coffee
const brewCoffee = (a: Water, b: CoffeeBeans): Coffee => // brrr
```



Funktioaalinen ohjelmointi *ihan oikeasti*

- > Kategoriateoria
- > Joukot ja funktiot muodostavat kategorian.
Joukot ovat kategorian objektit, ja funktiot kategorian morfismit. Näin ajateltaessa ongelmia voidaan yleistää ja ratkaisuista tulee uudelleen sovellettavia.



?????????????

Monadi

- > On monoidi endofunktiorien kategoriassa (??? :DDD)
- > Asia, jolla voidaan ketjuttaa funktioita itsemäärittelemällä tavalla.
- > **Mahdollistaa sivuvaikutukset funktionaalisessa ohjelmoinnissa**

Raportissa monadi saa rakkautta <3

```
3 const resultMonad = Result.of(helloMaybe())
4   .bind(addExclamation)
5   .bind(toUpperCase)
6   .bind(reverse)
```

Monadi

> Monadit on oikeasti TOSI JEES! (JavaScriptin Promise on käytännössä monadi) (ihan perus Array on käytännössä monadi)

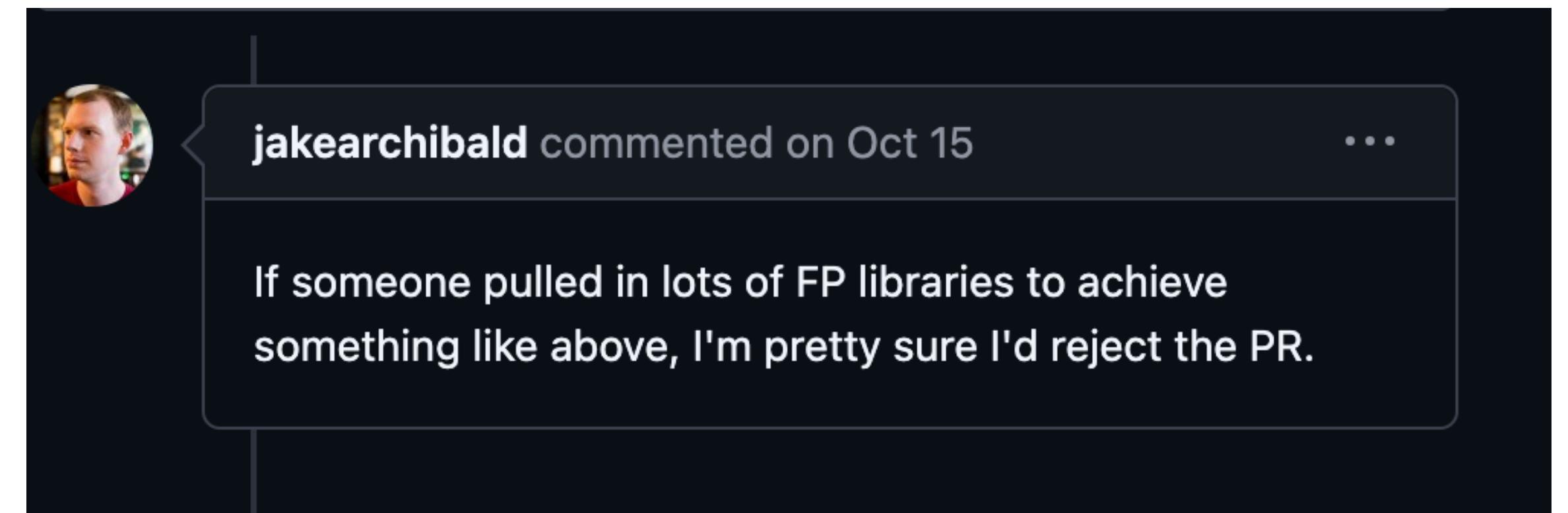
Niiden ongelma on määritelmän hähmäisyys.

-> (Sama teema toistuu funktioalisessa ohjelmoissa ylipäätään)

```
3 const resultMonad = Result.of(helloMaybe())
4   .bind(addExclamation)
5   .bind(toUpperCase)
6   .bind(reverse)

const promise = Promise.resolve(mightSayHello())
  .then(addExclamation)
  .then(toUpperCase)
  .then(reverse)
```

3. Kaikukammion räjäytys



Reality check

Haastattelu ja reflektointi

Jake Archibald



(Shopify, Google, Off the main thread -podcast, HTTP 203 podcast)

30+ vuotta alalla. Kirjoittanut JavaScriptiä kun Netscape Navigator oli asia.

Suostui vastaamaan minun kysymyksiin!

Hi, I was wondering do you still hate Array.reduce() ?
I'm writing a bachelors thesis about pragmatic usage
of fp in non fp languages, and would love to hear if
your feelings have changed at all in the last few years.

(I'm coming from the http 203 episode "is reduce()
bad".)

14. lokakuuta 2024 klo 1.03 ip.

I still hate it 😊

Reality check

Haastattelu ja reflektointi



jakearchibald commented on Oct 15

...

1. Is there any hope for FP in the mainstream?

I think FP has had a lot of positive influence on JS and will continue to do so. I'm watching <https://github.com/tc39/proposal-record-tuple> with interest.

I think where it (and many other coding patterns like OOP) fall down is when it's forced to be academically pure, even at the cost of readability & performance. `reduce` is a good example of this - many uses are orders of magnitude slower than a simple for loop, and the reduce version obfuscates the order of operations. The only benefit is that it ticks some FP purity box that someone is holding themselves to.

Reality check

Haastattelu ja reflektointi



jakearchibald commented on Oct 15

...

In the HTTP 203 episode, "Is `.reduce()` bad?", you mentioned three stages of a programmer's journey:
a. Avoiding complex ideas like currying and `reduce` because they seem intimidating. b. Using those ideas to feel smart. c. Writing code like in a beginner's book. Has your perspective shifted since then, or have you found more nuances to how you write code now?

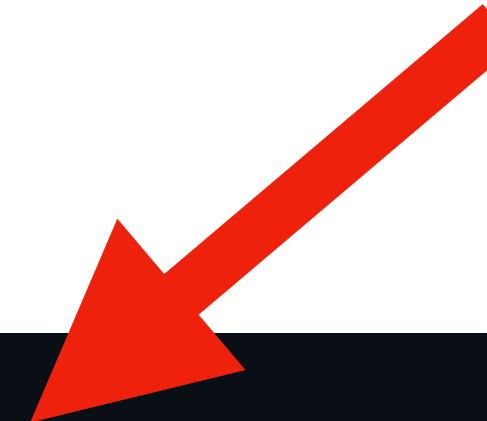
Has your perspective shifted since then, or have you found more nuances to how you write code now?

Not really. The best code is the code that solves the problem in the simplest and most efficient way possible. Sometimes you'll reduce readability to make some code perform better, and that should be done with care, as in is the performance improvement worth it given how this function will be used? I also realise that simplicity is subjective.

Reality check

Haastattelu ja reflektointi

Mun kyssäri



5. Implementing Maybe/Result as monads – too much?

What about turning types like Maybe and Result into full-fledged monads? For example:

```
const monad = Result.of(mightSayHello())
  .bind(addExclamation)
  .bind(toUpperCase)
  .bind(reverse)

const value = monad.value
// "!DLROW OLLEH" or Error("chili explosion")
```

Right, so your example translated to traditional JS would be:

```
try {
  const result = reverse(addExclamation(mightSayHello()).toUpperCase());
} catch (error) {
  console.log("it went wrong");
}
```



I don't think the 'Maybe' etc types improve anything here. The improvement would come with [a pipeline operator](#), but that's been stuck for years.

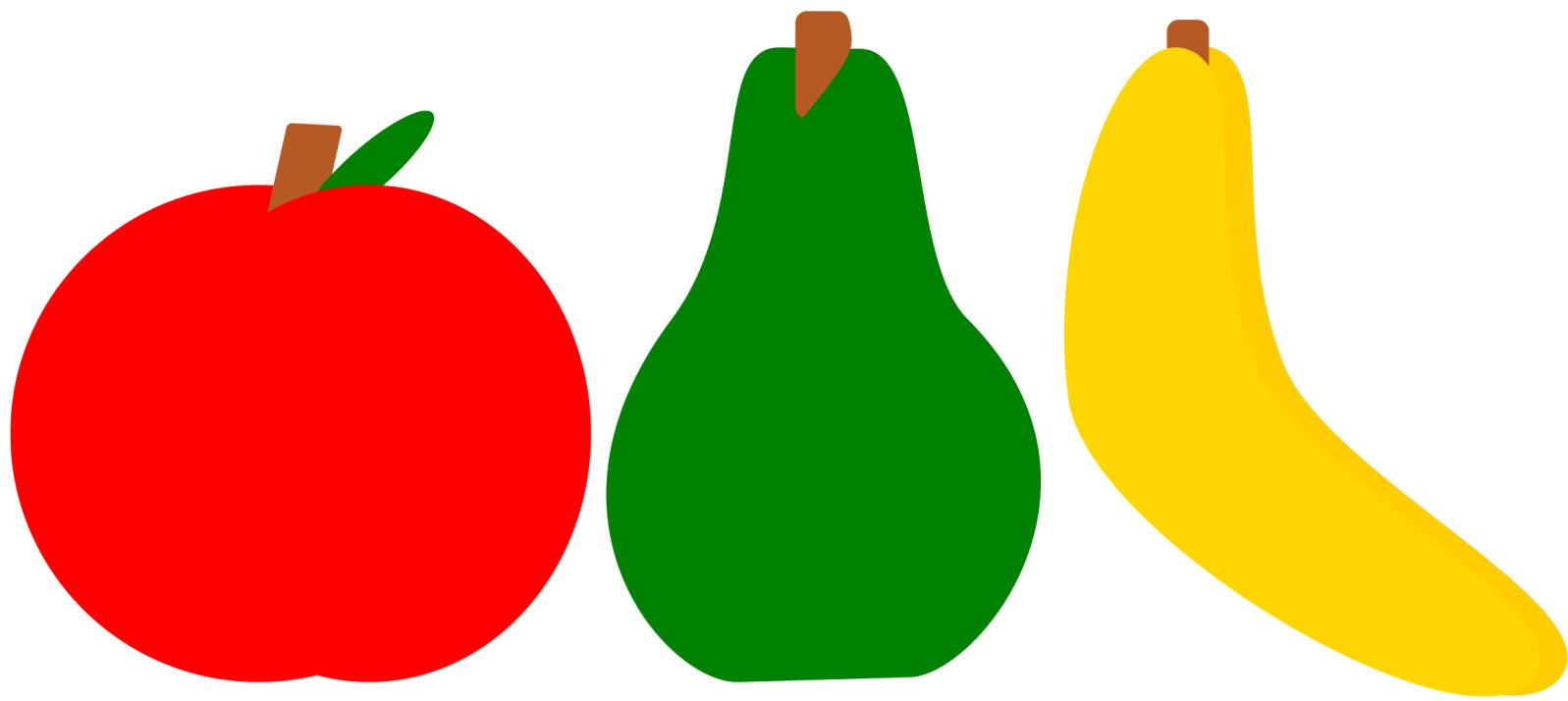
But I'd write it like this:

```
function someWellNamedFunction(): string | null {
  const result = mightSayHello();
  if (result === null) return;
  return reverse(result + '!').toUpperCase();
}
```

Here the order of operations is clear, and no one had to learn `Result.of`, binding, or monads. If someone pulled in lots of FP libraries to achieve something like above, I'm pretty sure I'd reject the PR.

4. Mallinnus

Funktioaalisen ohjelmoinnin periaatteita kevyesti



Mihin joukkoon nämä saattaisivat kuulua?

Ongelmien mallintaminen oikein ja koodi saamaan lopettamaan valehtelu

```
1 type Question = {  
2   question: string  
3   answers: Array<string>  
4   correctAnswerIndices: Array<number>  
5 }  
6  
7 type Quiz = {  
8   description: string  
9   questions: Array<Question>  
10 }  
11
```

Funktioalinen ohjelointi on joukkojen muuttamista toisiin.

Jos joukot on määritetty väärin, onko kyseessä enää edes funkcionaalinen ohjelointi?

Ongelmien mallintaminen oikein ja koodi saamaan lopettamaan valehtelu

```
1 type Question = {  
2   question: string  
3   answers: Array<string>  
4   correctAnswerIndices: Array<number>  
5 }  
6  
7 type Quiz = {  
8   description: string  
9   questions: Array<Question>  
10 }  
11
```

Virheet

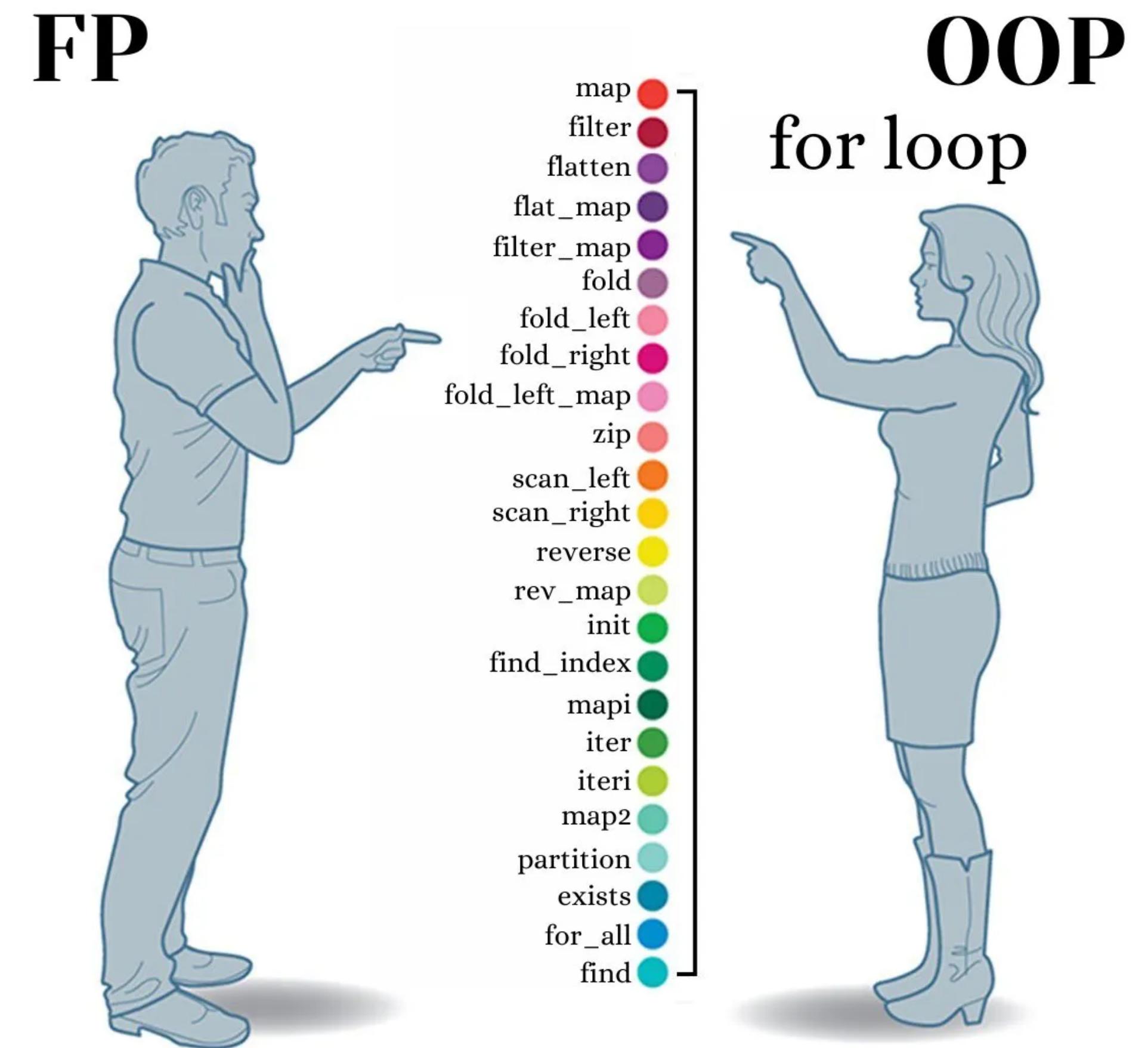
1. questions voi olla tyhjä
2. answers voi olla tyhjä
3. correctAnswerIndices voi olla tyhjä
4. correctAnswerIndices voi sisältää mitä vain numeroita

Ongelmien mallintaminen oikein ja koodi saamaan lopettamaan valehtelu

```
1 type Question = {  
2   question: string  
3   answers: Array<string>  
4   correctAnswerIndices: Array<number>  
5 }  
6  
7 type Quiz = {  
8   description: string  
9   questions: Array<Question>  
10 }  
11
```

```
type NonEmptyArray<T> = {  
  first: T  
  rest: Array<T>  
}  
  
type CorrectAnswer = { type: 'correct'; answer: string }  
type IncorrectAnswer = { type: 'incorrect'; answer: string }  
  
type Question = {  
  question: string  
  correctAnswers: NonEmptyArray<CorrectAnswer>  
  incorrectAnswers: Array<IncorrectAnswer>  
}
```

5. Lopetus



”It depends”

- + Funktionaalinen ohjelointi **poistaa testattavan koodin määrään**
- + Funktionaalinen ohjelointi **ei valehtele**
- + Funktionaalinen ohjelointi **on kivaa**

- Funktionaalinen ohjelointi **on teoreettista ja huuruista**
- Funktionaalinen ohjelointi **on taas uusi asia mikä pitää opetella**
- Funktionaaliisen ohjelmoinnin **määritelmä on häilyvä**

Reflektio

- Käsitys funktioalisen ohjelmoinnin laajuudesta kasvoi
- Confirmation bias maybe no longer
- En jatkossa tyrkytä funktioalista ohjelointia yhtä helposti ratkaisuksi ongelmiin
- Funktioalinen ohjelointi kukoistaa oikeasti funktioalisissa ohjelointikelissä
- Olen tyytyväinen lopputulokseen ja käytyn prosessiin
- Vaikea aihe, vaikea lähestymistapa, vaikeaa ja harmaata

Palaute

TIL et Promise on Monadi. Päivitän CVn

Joo. Iha hyvännäköst settii mut tarvii kyl chat gpt avuks ku lukee tota
et se voi selittää mitä siel puhutaa

Hyvää työtä @Artru 🙌 ton lukeminen saa mut vaan välttää ramdaa enemmän

Tomas 20:05

| Turing-vahva lambda-kalkyyli-pohja

Täst tulee mun motto

Joel 20:35

ollaaks me molemmat viettämäs iltaa Artun työn kanssa

Joel 20:41

TIL: turing vahvuus

meidän kakka koulussa opeteltiin ainoastaan turing täydellisyydestä

huge

No kannattaako?

- Kannattaa, mutta vasta vahvan tarkastelun jälkeen
- Pragmatismi ei suosi funktioaalista ohjelmointia

Kiitos mielenkiinnosta!

Pragmaattisen
funktionaalisen
ohjelmoinnin arvointi
Insinöörityö
Artu Pennanen 25.11.2024

