Programação Funcional

λfλx.f(f(fx)) motivos para aprender

Sobre Mim

- Arquiteto de Software na TOTVS Goiânia
- Equipe de Framework, Engenharia e Inovação
- Centenas de milhares de linhas de códigos escritas
- +1000 Litros de café bebidos
- -10 anos de expectativa de vida
- Java paga meu salário mas queria mesmo era estar programando em Haskell

Haskell


Son: Dad, why is my sister named Rose?
Dad: Because your mom loves roses!
Son: Thanks daddy.
Dad: No problem, Haskell.

“Existem duas maneiras de se construir o desenho de um software: Uma maneira é fazê-lo de um modo tão simples onde obviamente não se há deficiências, a outra é fazê-lo de um modo tão complicado onde não se há deficiências óbvias. A primeira maneira é muito mais difícil.”

Charles Hoare (Prêmio Turing 1980)

“Nossa habilidade de se decompor problemas em pequenas partes depende diretamente da nossa capacidade de se juntar tudo isso novamente.” John Hughes (Why functional programming matters)

O que é?

- É um paradigma de programação.
- É um modo de se programar utilizando **funções puras, evitando estados compartilhados, mutabilidade e efeitos adversos.**
- É declarativa ao invés de imperativa.
- É linda s2

Origem

- Alonzo Church
- Orientador de Doutorado do Alan Turing
- λ-Calculus
- Tese de Church-Turing
- É antiga pra caralho
- Uma das primeiras implementações foi o LISP.

λ-Calculus

- Base matemática das linguagens funcionais
- Sistema matemático formal
- A “Linguagem de programação” mais simples
- Modelo universal de computação
- Se “roda” em uma máquina de turing, pode ser representado em cálculo lambda.

λ-Calculus

Variáveis

Expressão

$$\lambda x \lambda y \, . \, x+y$$

Aplicação

λ-Calculus

$$\lambda f \lambda x.f(f(fx)) = 3$$

Motivo λfλx = fx

Quase **SEM**

EFEITOS

ADVERSOS

=0

Sem efeitos adversos

Sem efeitos adversos

- Funções Puras
- Transparência Referencial
- Imutabilidade
- Mônadas (são apenas monóides na categoria dos endofunctores)

Funções Puras

- São funções que dadas as entradas, ela SEMPRE retornará o MESMO resultado.
- Não produz efeitos adversos OBSERVÁVEIS.

Funções Puras

```
def f(x: Int, y: Int) = x + y
```

Funções Puras

```
def f(x: Int, y: Int) = x + y
```

Pura

Funções Puras

```
def seq(i: Int) = {
  var soma = 0
  0 to i foreach((i) => soma = soma + i)
  soma
}
```

Transparência Referencial

- Uma função é dada como transparente quando a sua aplicação independe do contexto no qual ela é executada.

Transparência Referencial

```
def quadrado(x:Int) = x * x
```

Transparência Referencial

```
var num = 2
def dividir(x: Int) = {
  num = num/x
  num
}
```

Imutabilidade

- Em linguagens puramente funcionais, os tipos são naturalmente IMUTÁVEIS.
- Evita acoplamento temporal
- Facilita paralelismo

Imutabilidade

```
Email email = new SimpleEmail();
email.setFrom("ygor.santana@totvs.com.br",
"Ygor Castor”);
email.addTo("fgsl2017@google.com");
email.setSubject("Olá FGSL!");
....
email.send();
```

Imutabilidade

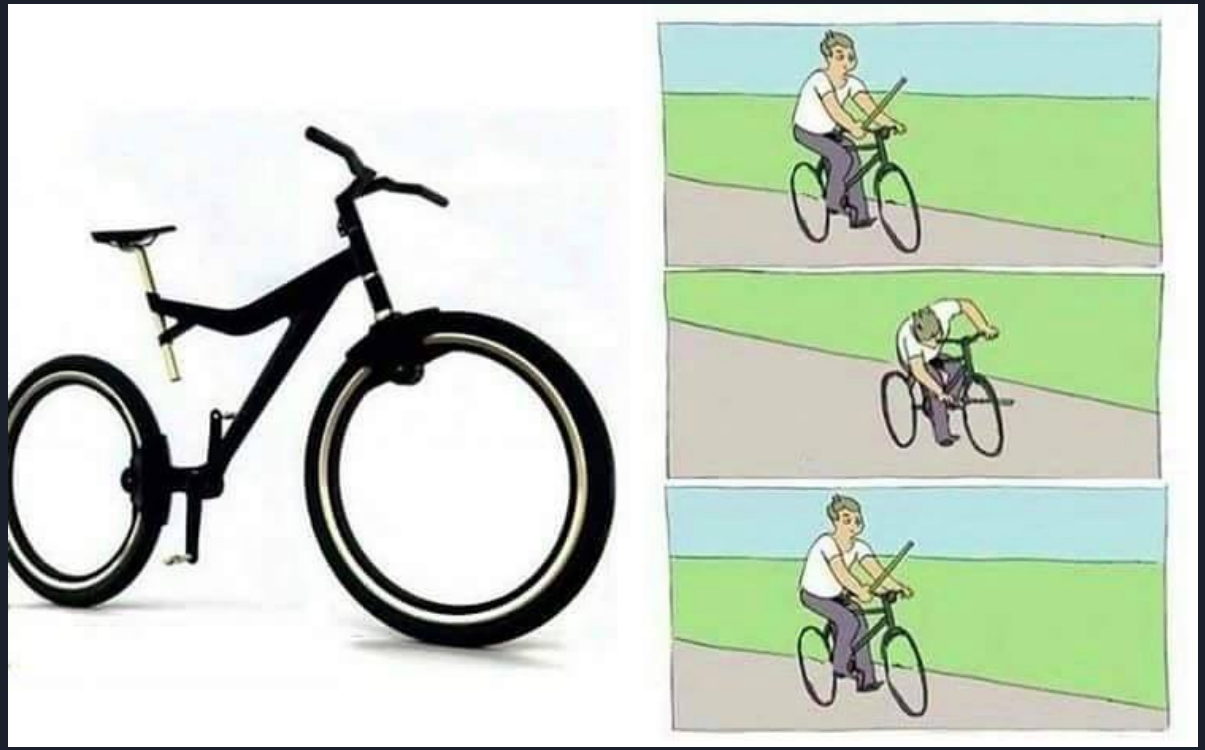
```
MailSender sender = new MailSender(new ConfigSmtp("smtp", 465,
"user", "pwd")
);

Message msg = new Message(Remetente.of("Ygor Castor",
ygor.santana@totvs.com.br),
Destinatarios.of("fgsl2017@gmail.com"), "Olá FGSL!");

sender.send(msg);
```

Programação Imperativa

Programação Funcional



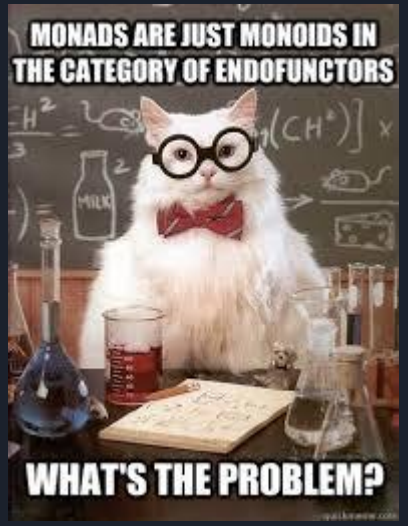
Como assim, “quase”?

- Programas completamente sem efeitos adversos são praticamente impossíveis de serem feitos.
- I/O e Aleatoriedade sempre estarão presentes no seu código.


FUI TAPEADO
GERADORMEMES.COM

Mônadas ao resgate

- Mônadas são amplificadoras de tipos.
- São “Construtores Computacionais”
- Definem comportamentos
- Lembram vagamente a “Command Pattern” do OO
- Modelam Efeitos Adversos



Mônadas

```haskell
instance Monad Maybe where
    return x = Just x
    Nothing >>= f = Nothing
    Just x >>= f  = f x
    fail _ = Nothing
```

Mônadas

```
safeLog :: (Floating a, Ord a) => a -> Maybe
a
safeLog x
    | x > 0      = Just (log x)
    | otherwise = Nothing
```

Mônadas

Motivo λfλx = f(fx)

É naturalmente paralelizável

Paralelismo Natural

- Não existem estados compartilhados
- Se as funções são puras, Beta reduções podem ocorrer em qualquer ordem (Propriedade de Church-Rosser)
- Execução Previsível
- Otimização

Motivo λfλx = f(f(fx)

~~Virou modinha~~

Princípios estão sendo aplicados nas linguagens mais usadas

Adoção

- O paradigma funcional está ganhando muito espaço no momento.
- Linguagens mainstream adotando princípios: Java , C#, C++, Ruby, Python, ...
- Linguagens tecnicamente funcionais: Javascript
- Linguagens Funcionais: Erlang, Elixir, LISP, Clojure, Scala
- Linguagens Puramente Funcionais: Idris, Haskell, ELM

Adoção - Uso no mundo real

- NuBank: Serviços em Clojure, uso de Clojurescript
- Google: Haskell nos filtros de Spam
- Facebook: Erlang no Chat, Haskell nos filtros de Spam
- Discord: Elixir para controlar os chats
- Pivotal: Erlang no RabbitMQ
- Mozilla: A Servo engine é escrita em Rust
- Twitter: Scala

Adoção - Exemplo - XMONAD

- https://github.com/xmonad/xmonad

Como aprender?

- https://www.edx.org/course/introduction-functional-programming-delftx-fp101x-0 - Introdução a programação funcional em haskell, com o pica das galáxias do Eric Meijer.
- https://www.coursera.org/learn/progfun1
- https://www.edx.org/course/paradigms-computer-programming-louvainx-louv1-1x-2#.VKWU9aYVIpk
- https://mitpress.mit.edu/sicp/full-text/book/book.html <- MELHOR LIVRO
- https://www.amazon.com/Concepts-Techniques-Models-Computer-Programming/dp/0262220695 <- Van Roy é o pica

Dúvidas?


Mutable State
Weak Typing
Side Effects
Mutable State
Weak Typing
Side Effects
@System.out.println( )

Contato

- E-Mail: ygorcastor@gmail.com
- LinkedIn: https://www.linkedin.com/in/ygorcastor/