

Programação Funcional com Go

Possibilidade ou forçação de barra?

Marco Ollivier - setembro 2023 - GopherCon Brasil

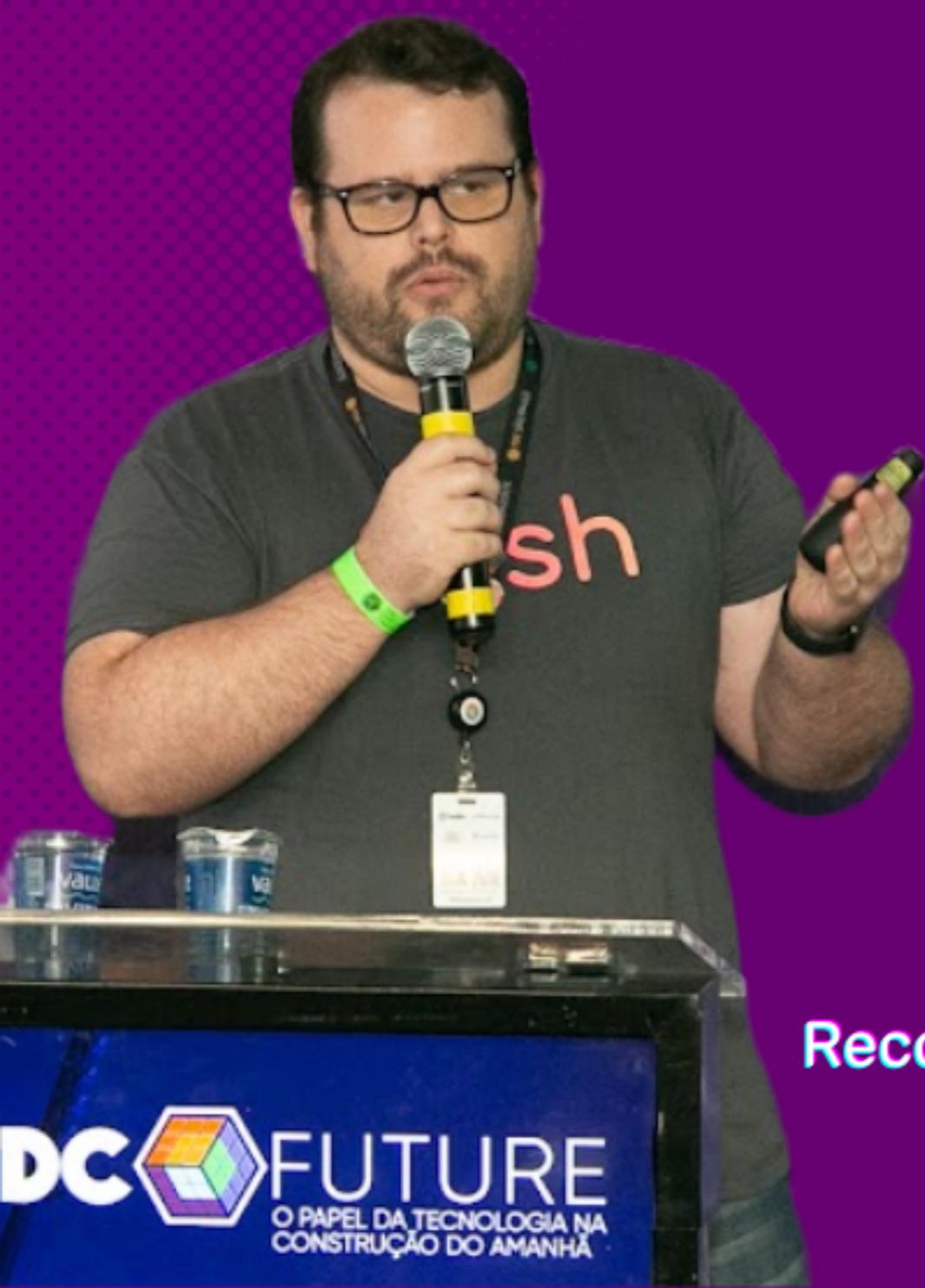
Go e o Paradigma Funcional?
É possível?

Sim

Valeu, valeu :)



Oi, eu sou
Marco Ollivier



Formação



Jornada
Profissional

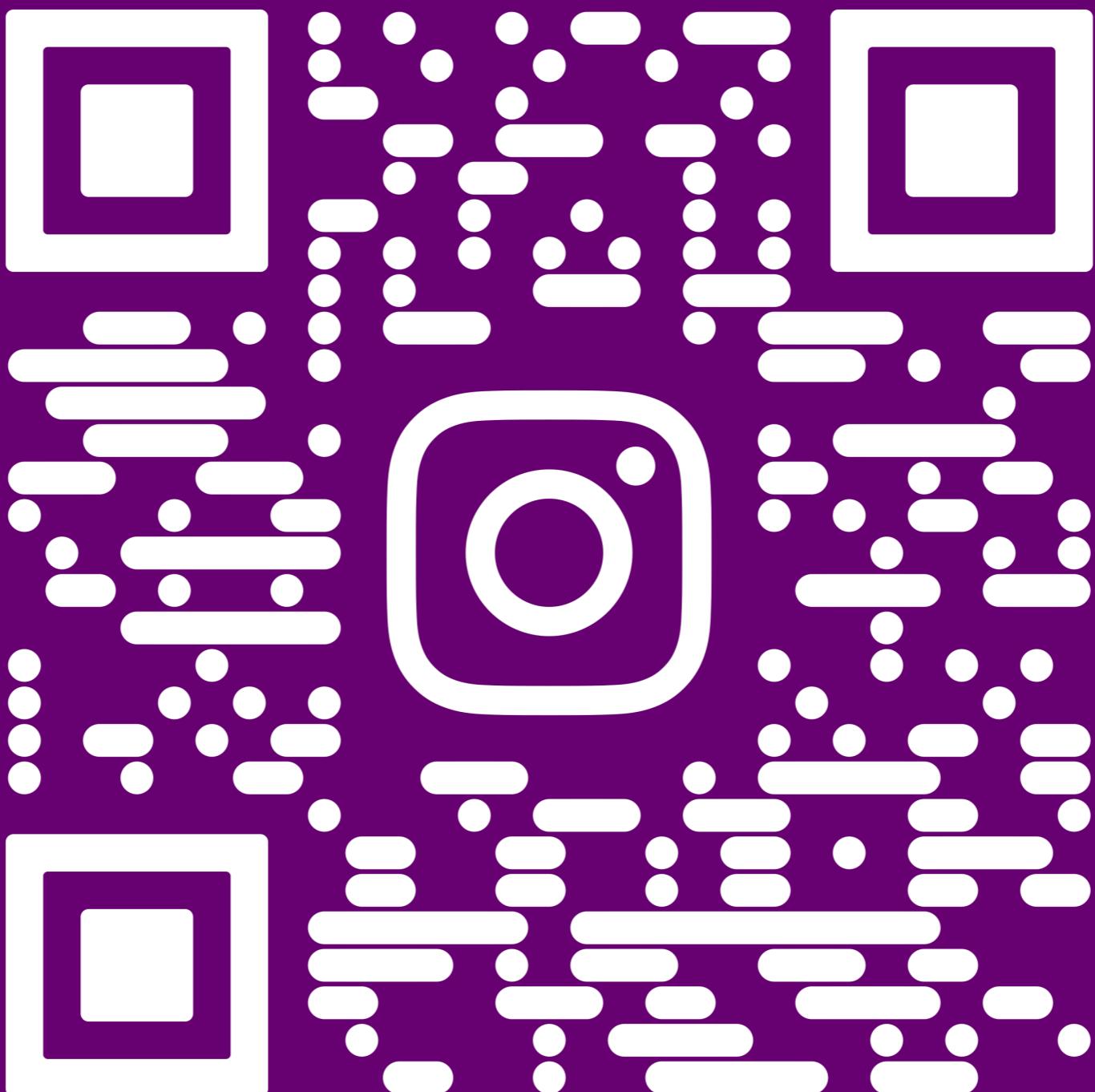


Voluntariado
e palestras



Reconhecimentos





essemarco.top

Agenda

- Alinhando os patos
- O paradigma
- Aplicando conceitos
- Exemplos e opções de implementação
- Conclusão

Alinhar os patos

Vou tentar não ter viés

**Funcional tá na moda, mas não é
por isso que estamos aqui**

Certo?

Certo?

**Funcional não vai resolver todos
os seus problemas**

Funcional é apenas mais um
paradigma...

ou seja... uma forma de escrever
código

Quando você toma uma escolha...

**Automaticamente está abrindo
mão de alguma outra**

Depende... trade-offs...

blablabla frases de que fazem
você parecer Sr.

Mentira... é porque eu gosto e
ponto



THE FUNCTIONAL WAY IS THE RIGHT WAY

Mentira outra vez

Chega de piadinha... o assunto é
sério

Momento exposed

Cadê o Elton?

DEUS PERDOA





eltonminetto.dev @eminetto · 3 de dez de 2022

...

Tuitei que acho programação funcional overrated. Esperando as ofensas para todas as minhas gerações passadas e futuras começarem... 😊

13

12

34

...

↑

Doar



Marco Ollivier @marcopollivier · 3 de dez de 2022

...

Em resposta a [@eminetto](#)

É exatamente o que o [@p_balduino](#) disse... é só um jeito de pensar. Admito que pra mim funcionou muito bem. Meio que me ajudou a ver algumas coisas de forma diferente e melhorar meu código.

1

12

5

...

↑

Doar



Marco Ollivier
@marcopollivier

...

Em resposta a [@marcopollivier](#) [@eminetto](#) e [@p_balduino](#)

Mas talvez tenha sido muito mais pelo efeito de mudar a forma de ver o problema do que o paradigma em si

3:53 PM · 3 de dez de 2022

-Você é de guarda rancor ?

- Eu não...



Pessoas que
falam mal de
funcional

2020

Primeira vez com
Funcional
profissionalmente

nu

Hexagonal +
Clojure +
Funcional

2022

Possibilidade de
refatorar e rever
paradigmas

flash

JS/TS/NodeJS +
Clean Arch + Time
querendo
aprofundar no
funcional

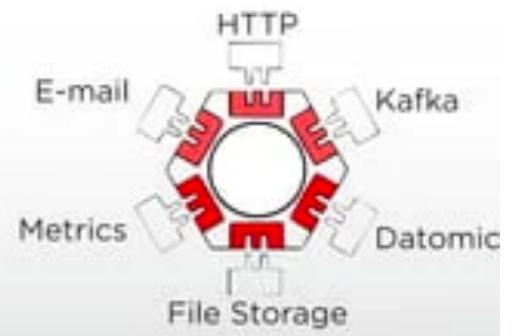
Arquitetura bem definida e RESPEITADA

Hexagonal + Diplomat

Adaptadores (Diplomat)

Ports and Adapters

- Adaptadores são os conectores das portas
- Contém as funções que tratam as chamadas HTTP, mensagens do Kafka, etc
- **Adaptam o wire schema para o schema interno**
- Chamam e são chamados pelas funções controllers
- **Testada com** versões falsas dos componentes, ou **mocks**



Escopos bem separados

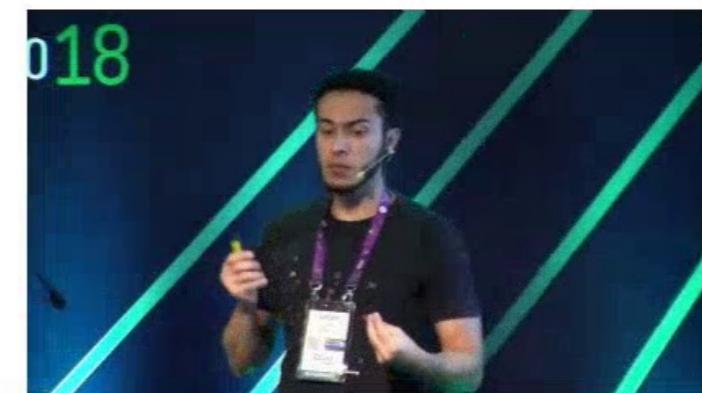
**Testes bem escritos e apenas para
o que realmente é necessário**

**Respeito a conceitos
fundamentais da funcional**

Ex. Funções puras

Nu... primeiro contato com funcional

Tecnologias escolhidas
Imutabilidade



Gravado no

QCon
SÃO PAULO

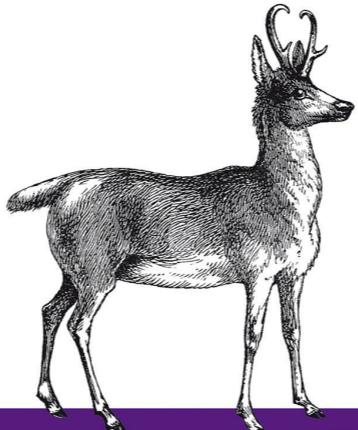
Trazido a você por

InfoQ
BRASIL

O paradigma

**“...supus que ainda usaria (...) OO
como abordagem principal (...)
Estava errado” — Dean Wampler**

Ferramentas para Melhor Concorrência, Abstração e Agilidade



Programação Funcional

para Desenvolvedores Java

O'REILLY®
novatec

Dean Wampler

Paradigma de programação

$$f(x) = x^2 + 3$$

Base em conceitos matemáticos
Evita mudança de estados

Evita mudança de estado... isso não deveria acontecer



```
package main

import "fmt"

var global int8 = 2

func main() {
    global = 127
    fmt.Println(global)
}
```

$$f(x) = x^2 + 3$$

$$f(2) = 2^2 + 3 \rightarrow 7$$

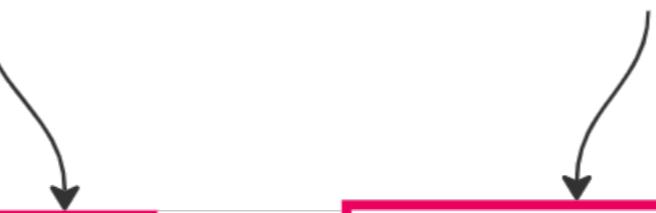
Contrato



$$f(x) = x^2 + 3$$

Entrada

Processamento



$$f(2) = 2^2 + 3 \rightarrow 7$$

Argumento / Parametro

Saída / Retorno

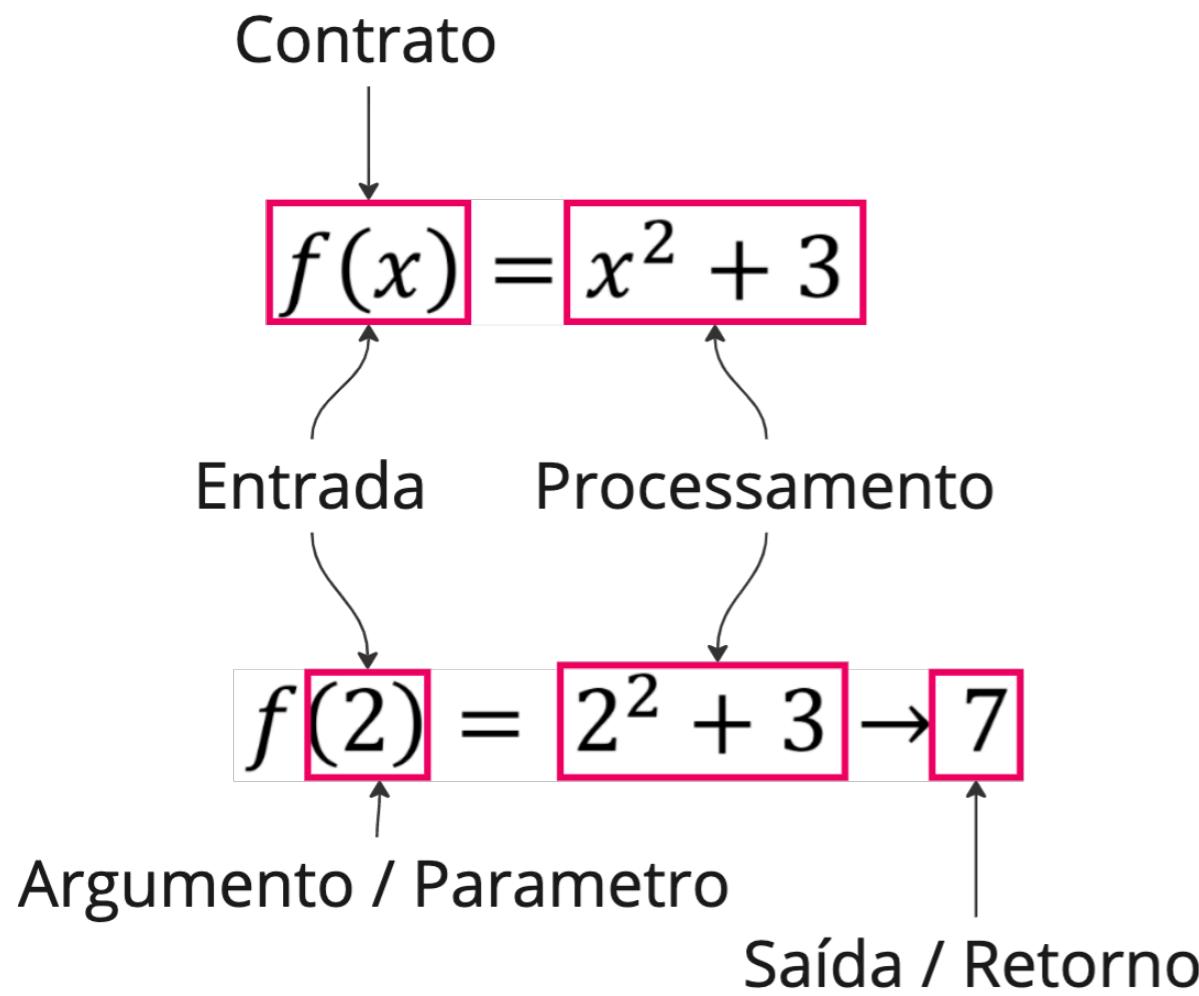


```
package main

import "fmt"

// f(x) = (x*x) + 3
// f(2) = (2*2) + 3
func f(x int) int {
    return (x * x) + 3
}

func main() {
    resultado := f(2)
    fmt.Println(resultado)
}
```



```

package main

import "fmt"

// f(x) = (x*x) + 3
// f(2) = (2*2) + 3
func f(x int) int {
    return (x * x) + 3
}

func main() {
    resultado := f(2)
    fmt.Println(resultado)
}

```

Ter funções me faz funcional?

**Ter funções me faz funcional?
Não exatamente**

Ter funções me faz funcional?

Como usar a função, sim

Ter funções me faz funcional?

A linguagem tem que prover as
ferramentas

Aplicando conceitos

Imutabilidade e funções puras

Imutabilidade



```
package main

import "fmt"

func elevaAoQuadradoESoma3(valor int) int {
    return (valor * valor) + 3
}

func main() {
    resultado := elevaAoQuadradoESoma3(2)
    fmt.Println(resultado)
}
```

Imutabilidade

“...Permitir valores mutáveis é o que torna a programação multithread tão difícil...”

Imutabilidade

“...Se múltiplas threads podem modificar o valor compartilhado, você precisa sincronizar o acesso a esse valor...”

Concorrência

Pra mim, talvez, seja a principal
vantagem

Concorrência (OO)

Prevê passagem de estados

Concorrência (OO)

As informações passadas podem
mudar
Gets, sets, etc

Concorrência (OO)

Recorrer a estruturas específicas

Concorrência (Func)

**Não existe mudança de estado
constante**
Os riscos são atenuados

Coleções

Qual o problema desse código na imutabilidade?



```
func OrdenaSliceInPlace(slice []int) {  
    sort.Ints(slice)  
}
```

Coleções



```
func OrdenaSlice(sliceOriginal []int) []int {  
    copia := make([]int, len(sliceOriginal))  
    copy(copia, sliceOriginal)  
    sort.Ints(copia)  
    return copia  
}  
  
func OrdenaSliceInPlace(slice []int) {  
    sort.Ints(slice)  
}
```

Coleções



```
func BenchmarkOrdeneSlice(b *testing.B) {
    slice := GeraSliceAleatorio(5)
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        OrdeneSlice(slice)
    }
}

func BenchmarkOrdeneSliceInPlace(b *testing.B) {
    slice := GeraSliceAleatorio(5)
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        OrdeneSliceInPlace(slice)
    }
}
```

Coleções

		Tempo por operação		Bytes alocados por operação	
goos: darwin goarch: arm64 pkg: gophercon Benchmark0rdenaSlice-8 Benchmark0rdenaSliceInPlace-8 PASS ok	gophercon	3.543s	5	12932932 22774755	100.4 ns/op 66.11 ns/op 72 B/op 24 B/op 2 allocs/op 1 allocs/op
goos: darwin goarch: arm64 pkg: gophercon Benchmark0rdenaSlice-8 Benchmark0rdenaSliceInPlace-8 PASS		1k	49584 634639	21542 ns/op 1898 ns/op 4120 B/op 24 B/op 2 allocs/op 1 allocs/op	
goos: darwin goarch: arm64 pkg: gophercon Benchmark0rdenaSlice-8 Benchmark0rdenaSliceInPlace-8 PASS ok	gophercon	3.439s	10k	961 33256	1225165 ns/op 36133 ns/op 81944 B/op 24 B/op 2 allocs/op 1 allocs/op

Funções puras

Para as mesmas entradas, ele sempre
retorna a mesma saída



```
package main

import "fmt"

var global int8 = 2

func main() {
    global = 127
    fmt.Println(global)
}
```

Funções puras

Sem efeitos colaterais

Funções puras

Quando falamos de efeitos colaterais, vamos falar de:

BD, Mensagens etc

Funções puras



```
( s/defn can-cancel? :- s/Bool
  [card-request :- models.card-request/CardRequest]
  ( and ( not ( created? card-request ))
    ( not ( canceled? card-request ))))
```



```
( s/defn ^:private update-card-request-to-created!
  [card-request :- models.card-request/CardRequest]
  ( update! card-request as-of datomic))
```

Perdão pelo código
Clojure....

Nada a ver isso ae

Funções puras

A saída do processamento dos valores
enviados para uma função sempre
retornará a mesma coisa

Funções puras



```
func elevaAoQuadradoESoma3(valor int) int {  
    return (valor * valor) + 3  
    fmt.Println(resultado)  
}
```

Funções de primeira classe e classe superior

Funções de primeira classe

Capacidade de tratar
funções como valores

Funções de primeira classe

Atribuídas a variáveis



```
func add(a, b int) int {  
    return a + b  
}  
  
func main() {  
    // atribuindo a função add a uma variável  
    var myFunc func(int, int) int = add  
}
```

Funções de primeira classe

Passadas como parâmetro para outras funções



```
func add(a, b int) int {
    return a + b
}

func compute(f func(int, int) int, a, b int) int {
    return f(a, b)
}

func main() {
    // atribuindo a função add a uma variável
    var myFunc func(int, int) int = add

    // passando a função add como argumento para outra função
    result := compute(myFunc, 2, 3)

    // exibindo o resultado
    fmt.Println(result)
}
```



```
package main

import "fmt"

func add(a, b int) int {
    return a + b
}

func times(a, b int) int {
    return a * b
}

func compute(f func(int, int) int, a, b int) int {
    return f(a, b)
}

func main() {
    // atribuindo a função add a uma variável
    var myAddFunc func(int, int) int = add
    var myTimesFunc func(int, int) int = times

    // passando a função add como argumento para outra função
    resultAdd := compute(myAddFunc, 2, 3)
    resultTimes := compute(myTimesFunc, 2, 3)

    // exibindo o resultado
    fmt.Println(resultAdd)
    fmt.Println(resultTimes)
}
```

Funções de primeira classe

Retornadas como valores de outras funções



```
func exemploRetornoComoFuncao() func(int, int) int {  
    return times  
}
```

Funções de ordem superior

Funções que tomam outras funções como argumento

Funções de ordem superior

E/ou

Retornam funções



```
package main

import "fmt"

func add(a, b int) int {
    return a + b
}

func times(a, b int) int {
    return a * b
}

func compute(f func(int, int) int, a, b int) int {
    return f(a, b)
}

func main() {
    // atribuindo a função add a uma variável
    var myAddFunc func(int, int) int = add
    var myTimesFunc func(int, int) int = times

    // passando a função add como argumento para outra função
    resultAdd := compute(myAddFunc, 2, 3)
    resultTimes := compute(myTimesFunc, 2, 3)

    // exibindo o resultado
    fmt.Println(resultAdd)
    fmt.Println(resultTimes)
}
```

```
func compute(f func(int, int) int, a, b int) int {  
    return f(a, b)  
}
```

Monads e o Either

Monads

É um tipo especial de container

Essa definição possui funções que permitem transformar esses valores

Monads

Não é tão visto em Go

Mas...

Either



```
const getAge = (user) => {  
  ...  
  
  return user.birthDate.isValid()  
    ? Either.of(now.diff(user.birthDate, 'years'))  
    : left('Not valid Birthdate')  
}
```

JS

Either



```
const getAge = (user) => {  
    ...  
  
    return user.birthDate.isValid()  
        ? Either.of(now.diff(user.birthDate, 'years'))  
        : left('Not valid Birthdate')  
}
```

Não são a mesma coisa, mas...



```
func divide(x, y float64) (float64, error) {  
    if y == 0 {  
        return 0, errors.New("division by zero")  
    }  
    return x / y, nil  
}  
  
result, err := divide(10, 0)  
if err != nil {  
    fmt.Println("Error:", err)  
} else {  
    fmt.Println("Result:", result)  
}
```

Either

Maaaaaaaaas...

Either

Pode ser problemático pra function composition

Function Composition



```
func addOne(x int) int {
    return x + 1
}

func double(x int) int {
    return x * 2
}

func compose(f func(int) int, g func(int) int) func(int) int {
    return func(x int) int {
        return f(g(x))
    }
}

func main() {
    // compondo as funções addOne e double
    addOneThenDouble := compose(double, addOne)

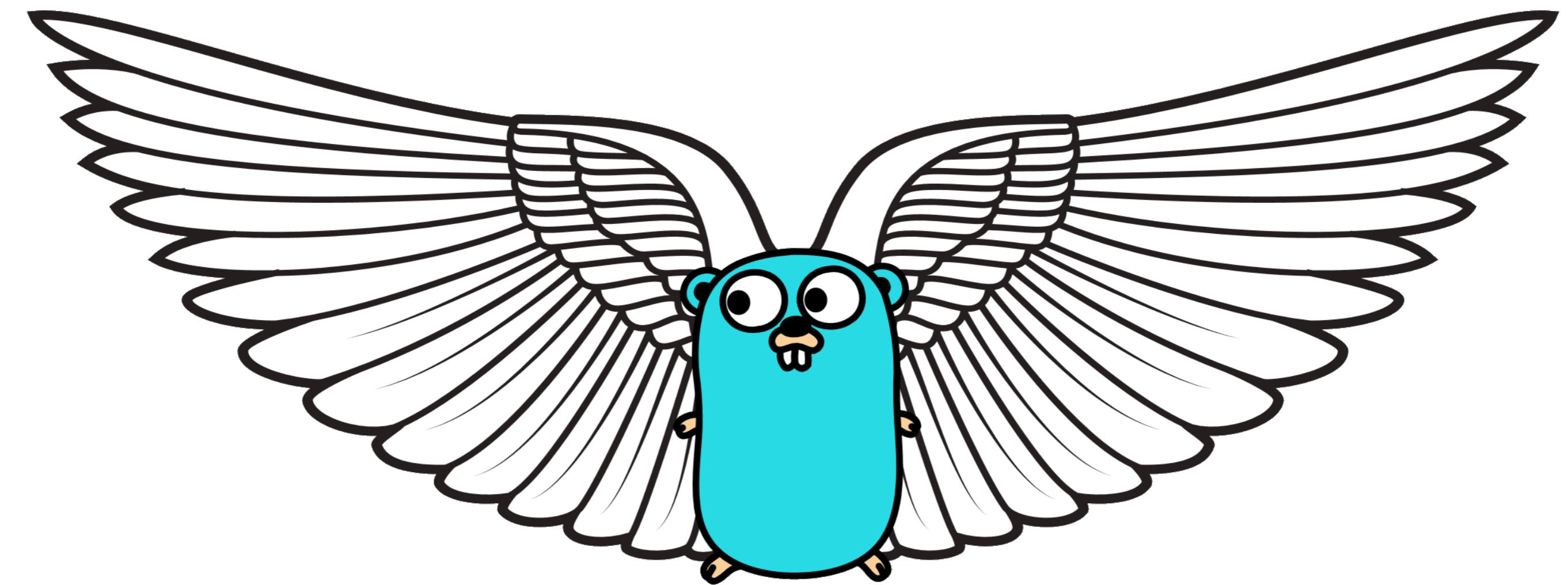
    // usando a nova função criada
    result := addOneThenDouble(3)
    fmt.Println(result) // exibe 8
}
```

O retorno de uma função é o primeiro parâmetro da
próxima

Uma função completamente nova é criada

```
// usando a nova função criada
result := addOneThenDouble(3)
fmt.Println(result) // exibe 8
```

fp-go



Marco Ollivier

Programação Funcional com TS

Possibilidade ou forçação de barra?

flash

fp-ts

Search fp-ts

Docs API Reference GitHub

Introduction

Learning Resources

Ecosystem

Modules

Guides



Typed functional programming in TypeScript

`fp-ts` provides developers with popular patterns and reliable abstractions from typed functional languages in TypeScript.

Disclaimer. Teaching functional programming is out of scope of this project, so the documentation assumes you already know what FP is.

Either

Implementa o Either para resolver o problema de composição de funções

Conclusão

Natty or not?



Minha percepção

Go não é nativamente funcional

Mas é viável usar e aproveitar dos benefícios

Minha percepção

Escrevo menos

Código mais limpo

Testar fica mais fácil

Mais fácil de encontrar as coisas

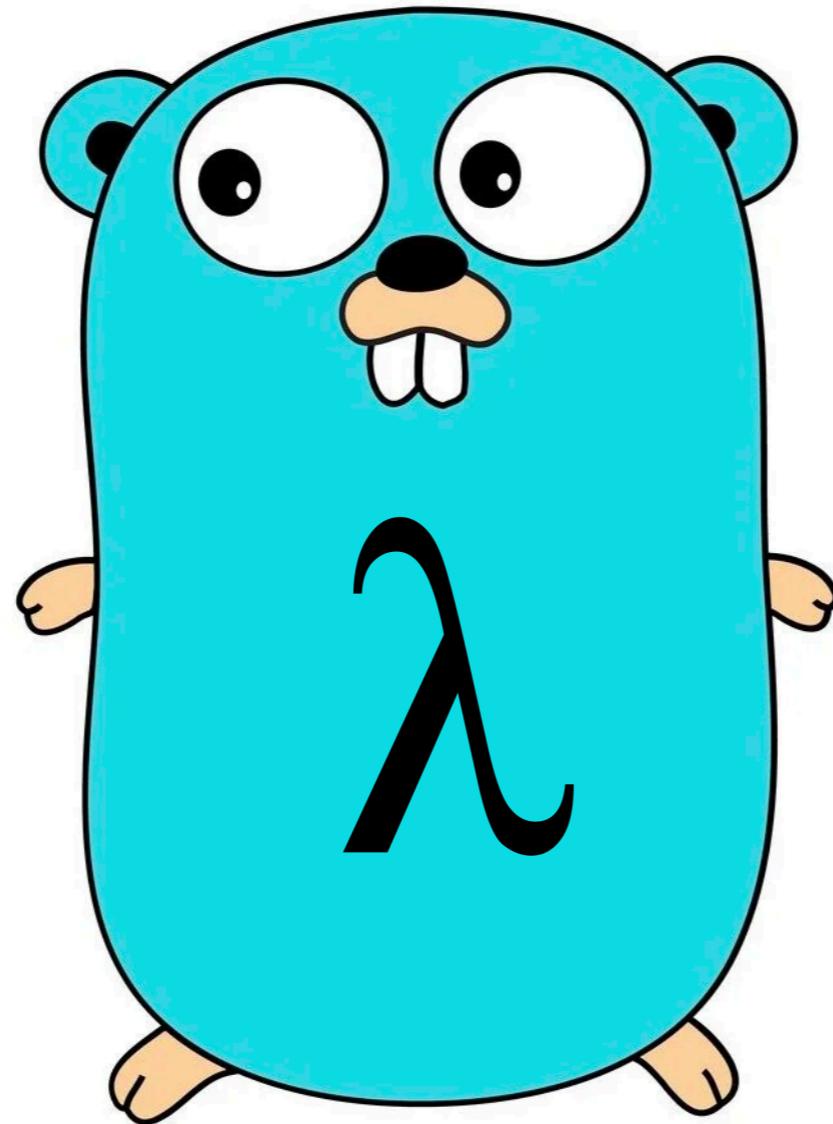
Conclusões

Funcional é cool, funcional é moderno, funcional é pop



Conclusões

É viável SIM usar funcional com Go



Próximos passos

Vejam outros conceitos

Point free

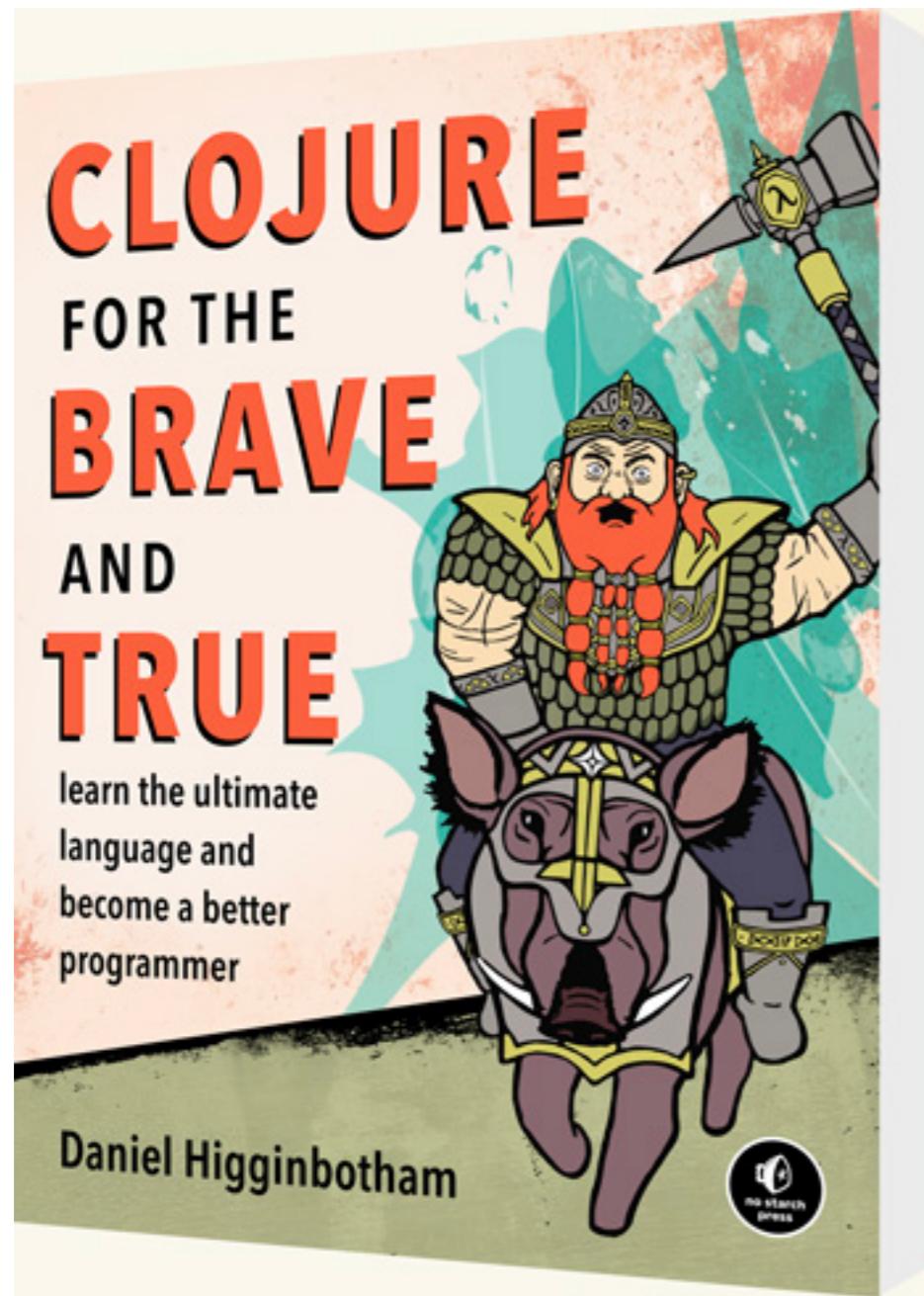
Functors

Maybe

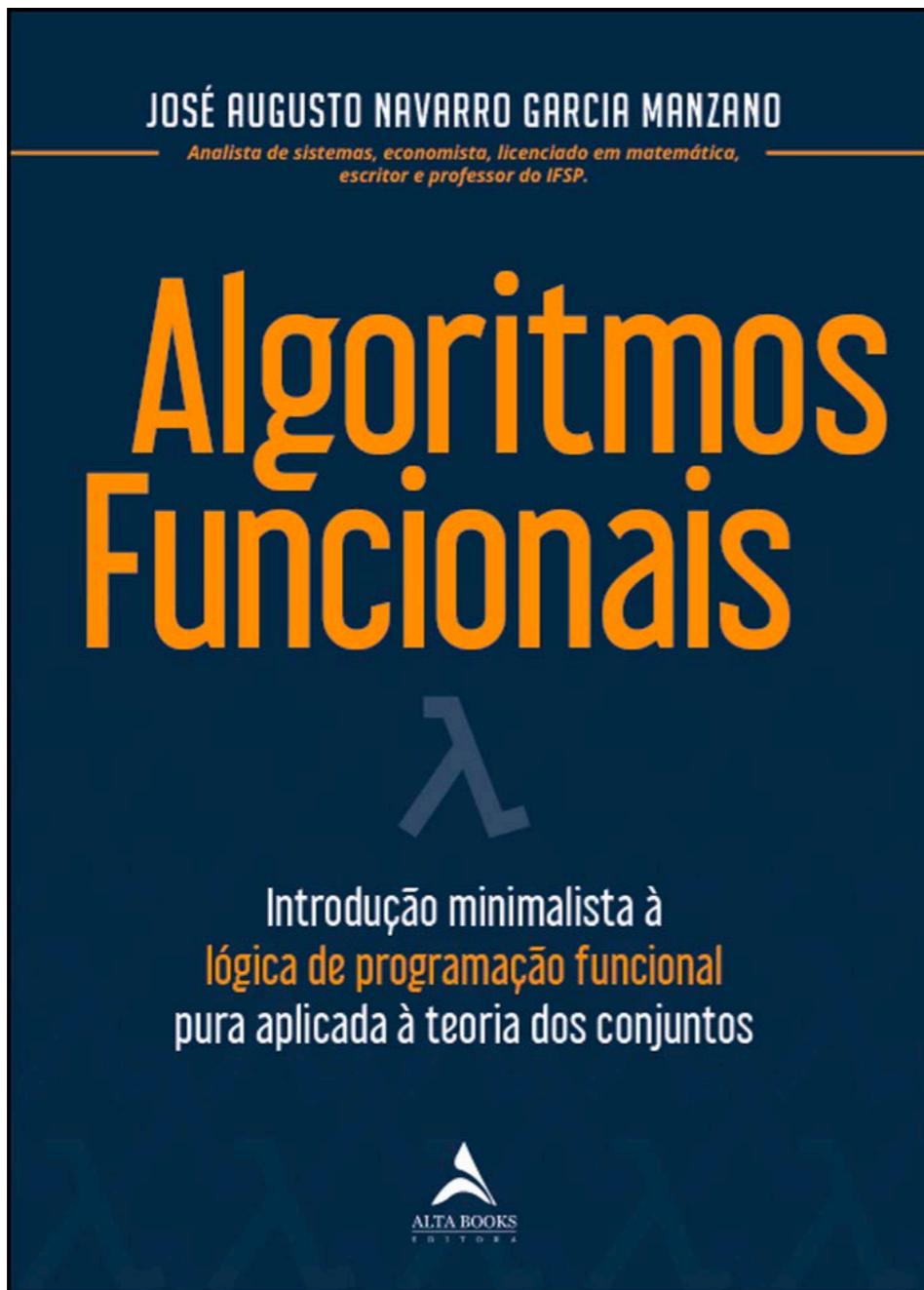
Future

Etc

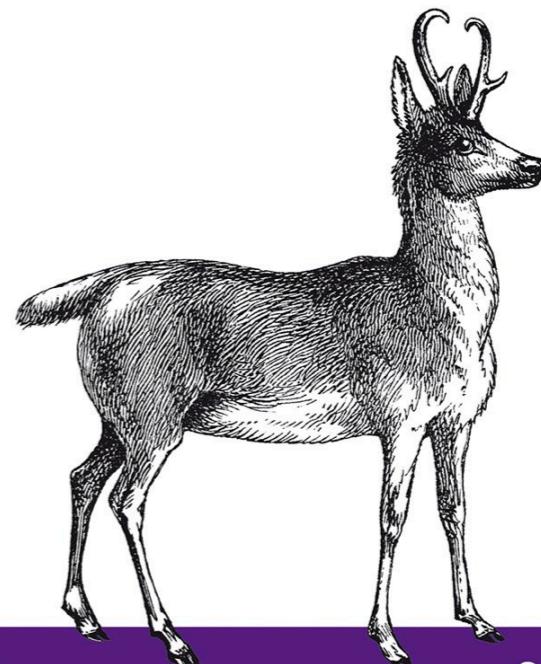
Leiam livros



Leiam livros



Ferramentas para Melhor Concorrência, Abstração e Agilidade



Programação Funcional

para Desenvolvedores Java

O'REILLY®
novatec

Dean Wampler



Obrigado =)