




Introdução ao Paradigma Funcional com Scala

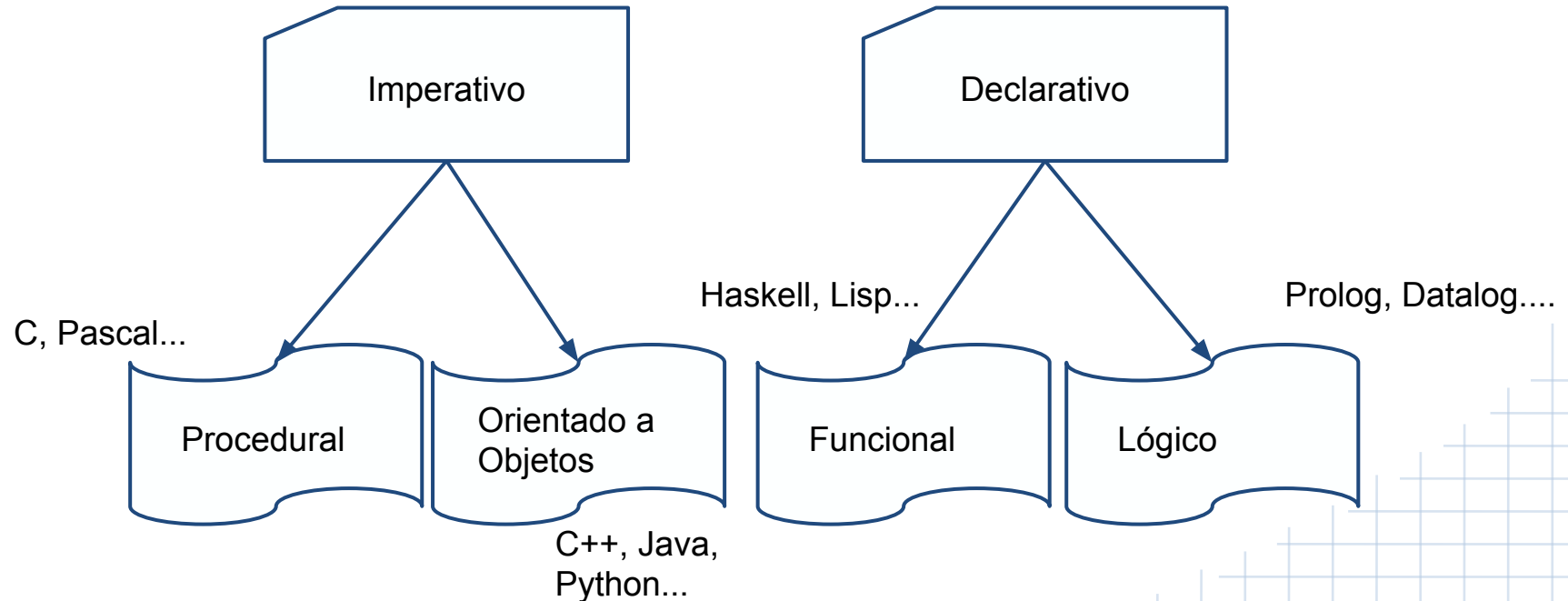
Gustavo Fernandes dos Santos
gfdsantos@inf.ufpel.edu.br



Paradigma Funcional

- É um paradigma de programação que trata a computação como avaliações de expressões onde é evitado a mudança de estados durante a execução do programa e utiliza dados imutáveis.

Paradigma Funcional?



Linguagens funcionais

- Linguagens puras (Não possuem o conceito de memória)
 - Haskell
 - Miranda
- Linguagens impuras (Possuem o conceito de memória)
 - Scala
 - Erlang
 - F#
 - OCaml
 - JavaScript

Características

- **Funções de Primeira Classe e de Alta Ordem**
 - **Primeira Classe:** permite a atribuição de funções a valores ou armazenamento em estruturas de dados
 - **Alta Ordem:** permite receber e retornar uma função de outra função
- **Funções puras**
- **Recursão**
- **Avaliação rigorosa e preguiçosa:** Forma com que os argumentos são avaliados numa expressão.
 - **Rigorosa:** Todos argumentos são processados no momento da avaliação
 - **Preguiçosa:** Os argumentos não são processados durante a sua avaliação
- **Sistema de tipos**

Por que Funcional?

“Because the life is too short for imperative programming”

John Hughes

Mãos na massa

```
def qs(l: List[Int]): List[Int] = l match {  
  case Nil => Nil  
  case h::t => qs(t filter {_ < h}) ::: h :: qs(t filter {_ >= h})  
}
```

Mãos na massa

```
def t(n:Int) = (0 to 10) map { _*n } filter { _<50 }
```


Mãos na massa

```
def t(n:Int) = (0 to 10) map { _*n } filter { _<50 }
```



Sentido da computação

Exemplo 1

```
int *t(int n) {  
    int vet[11];  
    for (i = 0; i <= 10; i++) {  
        vet[i] = i*n;  
    }  
    return *vet;  
}
```

Exemplo 1

```
def t(n: Int) = (0 to 10) map { (x) => x * n }
```

```
let t n = [0 .. 10] |> List.map (fun x => x * n)
```

```
t n = map (\x -> x * n) [0 .. 10]
```

Por que Funcional?

```
def t(n: Int) = (0 to 10) map { (x) => x * n }
```

```
int *t(int n) {  
    int vet[11];  
    for (i = 0; i <= 10; i++) {  
        vet[i] = i*n;  
    }  
    return *vet;  
}
```

Por que Funcional?

E se eu quiser retornar somente os números pares da tabuada de qualquer número? Ou um intervalo? Ou os valores maiores que um certo valor?

Exemplo 2

```
int *tabuada(int n) {  
    int vet[11];  
  
    for (i = 0; i <= 10; i++) {  
        if ( (i*n)%2 == 0 )  
            vet[i] = i*n;  
    }  
    return *vet;  
}
```

Exemplo 2

```
def tabuada(n: Int) = (0 to 10) map {_*n} filter {_%2 == 0}
```

```
let tabuada n = [0..10] |>  
    List.map(fun x -> x * n) |>  
    List.filter (fun x -> x % 2 = 0)
```

```
tabuada n = filter (\p -> mod p 2 == 0) (map (\x -> x * n) [0..  
10])
```

Por que Funcional?

- Sem:
 - Variáveis
 - Estruturas de repetição
 - Ponteiros
 - Mutabilidade
- Com:
 - Funções de alta ordem
 - Recursão
 - Avaliação preguiçosa

Por que Funcional?

- **SEM VARIÁVEIS?**
- **SEM ESTRUTURAS DE REPETIÇÃO?**



Funcional + OO?

É possível?

Funcional + OO?



Scala

Scalable Language

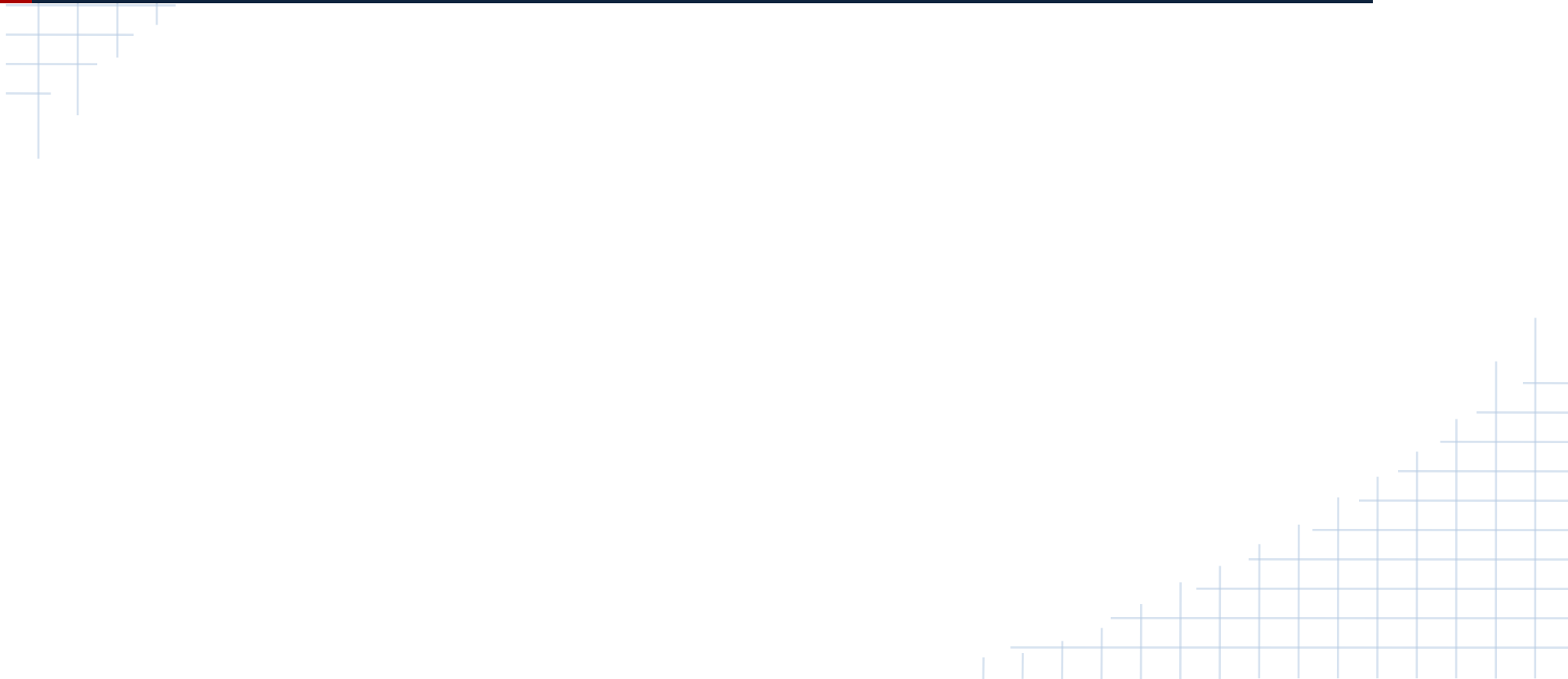
Martin Odersky

Escola Politécnica Federal de Lausanne (EPFL)

www.scala-lang.org



Quem usa Scala?



Quem usa Scala?

theguardian

foursquare®

LinkedIn

tumblr.

twitter™

NETFLIX

SONY



Por que?

The Scala logo consists of a red icon on the left, which is a stylized representation of the letter 'S' formed by three horizontal bars of increasing length, and the word "Scala" in a bold, black, sans-serif font to its right.The akka logo features a blue icon on the left, which is a stylized, rounded shape resembling a mountain or a drop, and the word "akka" in a bold, dark blue, sans-serif font to its right.

+

The play logo consists of a green icon on the left, which is a stylized, rounded shape resembling a play button, and the word "play" in a bold, dark green, sans-serif font to its right.The Java logo features a red icon on the left, which is a stylized representation of a coffee cup with steam rising from it, and the word "Java" in a bold, red, sans-serif font to its right.

Java?

- Interoperabilidade

Java!



Java!

```
import java.util.Scanner
```

```
object Programa {  
    def main(args: Array[String]) {  
        val scn = new Scanner(System.in)  
        val linha = scn.nextLine  
        println(linha)  
    }  
}
```

Começando

`val, var`

Começando

def

Começando

map, filter, reduce

Começando

```
def soma1(x: Int, y: Int) = x + y
```

```
def soma2(x: Int)(y: Int) = x + y
```

```
(0 to 5) map { soma2(1) }
```

```
('a' to 'g') filter { c => c != 'e' }
```

```
(0 to 5) reduce { soma1 }
```

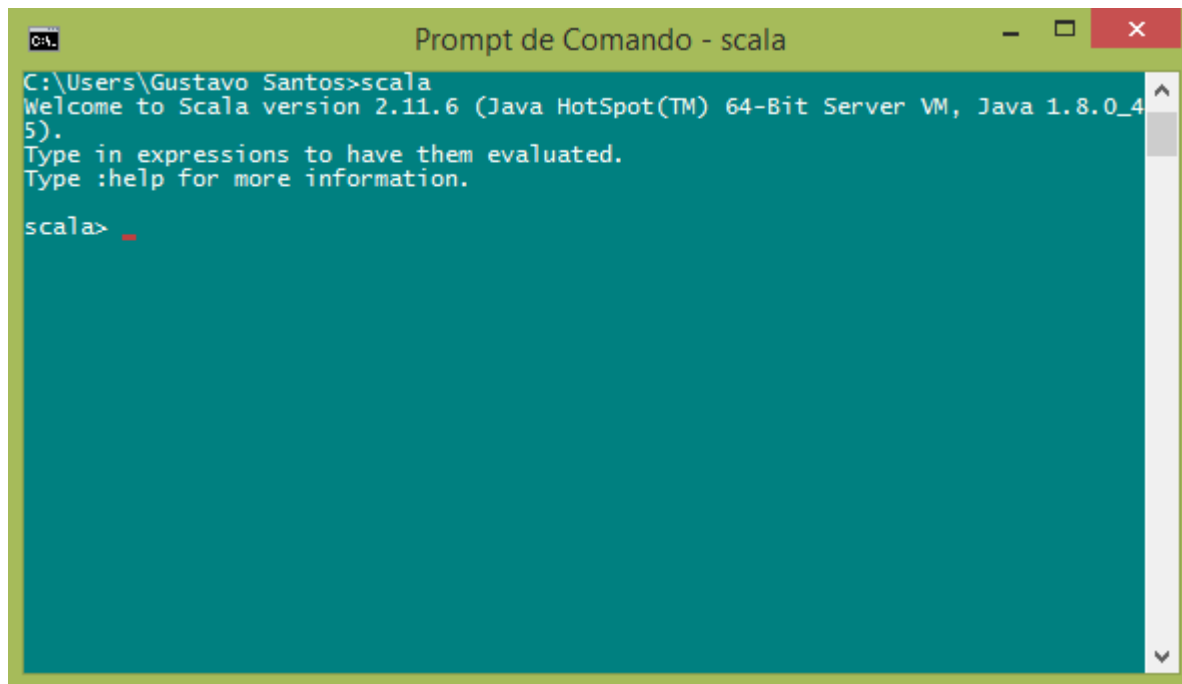
Começando

$$1 + 2 = (1) . + (2)$$

Características do Scala

- Scala REPL (Read Evaluate Print Loop)
- Otimização de chamadas recursivas em cauda
- Funções de alta ordem e de primeira classe
- Açúcares sintáticos (Syntax Sugar)
- Casamento de Padrões (Pattern Matching)
- Interoperabilidade
- Suporte completo ao paradigma funcional
- Suporte completo ao paradigma orientado a objetos
- Linguagem híbrida (Multiparadigma)
- ...

Scala REPL

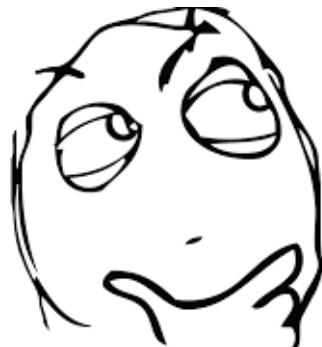


```
C:\Users\Gustavo Santos>scala
Welcome to Scala version 2.11.6 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_45).
Type in expressions to have them evaluated.
Type :help for more information.

scala> _
```

Otimização de chamadas recursivas em cauda

- Somente funções “recursivas em cauda” podem ser otimizadas
- Usa-se a anotação `tailrec`
- Evita o “Stackoverflow”



Otimização de chamadas recursivas em cauda

- Somente funções “recursivas em cauda” podem ser otimizadas
 - Usa-se a anotação `tailrec`
 - Evita o “Stackoverflow”
-
- **O Fatorial**

Otimização de chamadas recursivas em cauda

- Somente funções “recursivas em cauda” podem ser otimizadas
- Usa-se a anotação `tailrec`
- Evita o “Stackoverflow”

- **O Fatorial**

```
def fatorial(n: Int): Int = if (n == 1) 1 else n*fatorial(n-1)
```

Otimização de chamadas recursivas em cauda

- Somente funções “recursivas em cauda” podem ser otimizadas
- Usa-se a anotação `tailrec`
- Evita o “Stackoverflow”

- **O Fatorial**

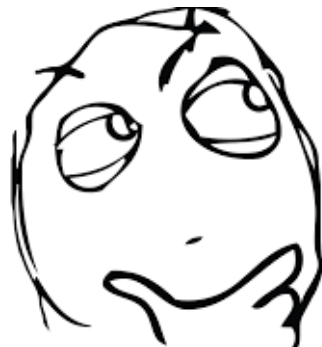
```
def fatorial(n: Int): Int = if (n == 1) 1 else n*fatorial(n-1)
```

```
@annotation.tailrec
```

```
def fat(n: Int, acum: Int = 1): Int = if (n == 1) acum else fat(n-1, n*acum)
```

Funções de alta ordem

- Linguagens de programação com suporte a funções de alta ordem, encaram uma função da mesma forma como uma linguagem imperativa, como C++, encara uma variável.
- Funções podem ser passadas como argumentos para outras funções
- Funções podem ser retornadas de outras funções
- Funções podem ser compostas
- Funções podem ser atribuídas
- ...



Funções de alta ordem

- Um exemplo de funções de alta ordem pode ser visto no Cálculo, onde o operador d/dx retorna a derivada de uma função f .

$$d/dx(x^2) = 2x$$

Funções de alta ordem

C++

```
...  
function<int, int> soma = [](x, y) { return (x + y); };  
...
```

Scala

```
val soma = (x: Int, y: Int) => x + y
```


Funções de alta ordem

```
val f = (x: Int) => 2 * x
```

```
def g(f: Int => Int, x: Int) = f(x)
```

```
println(g(f, 3))
```

Funções de alta ordem

```
val f = (x: Int) => 2 * x
```

```
def m[A,B](f: (A) => B, l: List[A]): List[B] = l match {  
  case Nil => Nil  
  case h::t => f(h)::m(f, t)  
}
```

Pattern Matching

(Switch com super poderes)

- Verifica um dado padrão em um conjunto de dados

```
def maiorque(x: Int, y: Int) = {  
  (x, y) match {  
    case (x, y) if x > y => true  
    case (x, y) if x < y => false  
    case _ => false  
  }  
}
```

Pattern Matching

- Verifica um dado padrão em um conjunto de dados

```
def maiorQue(x: Int)(y: Int) = {  
  (x, y) match {  
    case (x, y) => if (x > y) true else false  
    case _ => false  
  }  
}
```

Pattern Matching

- Verifica um dado padrão em um conjunto de dados

```
def somaLista(l: List[Int], soma: Int = 0): Int = l match {  
  case Nil => soma  
  case head::tail => somaLista(tail, head + soma)  
}
```

Pattern Matching

- Verifica um dado padrão em um conjunto de dados

```
def aplicaLista(f: Int => Int, l: List[Int]): List[Int] = l match {  
  case Nil => Nil  
  case h::t => f(h)::aplicaLista(f, t)  
}
```

Pattern Matching

- Verifica um dado padrão em um conjunto de dados

```
def map2[A, B]( l: List[A])(f: (A) => B): List[B] = l match {  
  case Nil => Nil  
  case h::t => f(h)::map2(t)(f)  
}
```

Experimente:

```
map2(('a' to 'e').toList) { (x: Char) => (x + 1).toChar }
```

Avaliação preguiçosa

- Técnica que atrasa a computação de uma expressão enquanto o seu resultado não for requisitado

```
scala> lazy val naturais = Stream.from(0)
```

```
scala> val lista10 = naturais take 11
```

```
lista10: scala.collection.immutable.Stream[Int] = Stream(0, ?)
```

```
scala> lista10.force
```

```
res15: scala.collection.immutable.Stream[Int] = Stream(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```


For Comprehension

```
def tabuada(n: Int) = for (i <- 0 to 10) yield i*n
```

```
let tabuada n = [for i in 0 .. 10 -> i*n]
```

```
let tabuada n = [n*x | x <- [0..10]]
```

For Comprehension

```
def tabuada(n: Int) =  
  (for (i <- 0 to 10) yield i*n) filter { (x) => x%2==0 }
```

```
let tabuada n =  
  [for i in 0 .. 10 do if (i*n)%2 = 0 then yield i*n]
```

```
let tabuada n = filter (\x -> mod x 2 == 0) [n*x | x <- [0..10]]
```

Vamos atuar!



Quer aprender mais?

- Coursera
- Livros
- Youtube
- Documentação

Nem tudo é perfeito

- A linguagem ainda é recente
- Muitas maneiras de realizar uma mesma computação
- Compilador ainda é lento e gera muito “lixo”
- SBT (Scala Build Tool) ainda não tem uma versão estável
- IDE (Eclipse) com poucos recursos comparada ao IntelliJ e Visual Studio
- A cada versão, um pacote é removido ou substituído por outro

Para quem Scala é indicado?

- Para quem não conhece o paradigma funcional
- Para quem usa Java e quer conhecer uma nova linguagem que roda na JVM e seja compatível com seus programas ou seus frameworks
- Para quem precisa de uma linguagem que seja facilmente distribuída
- Para entusiastas

Alternativas

- Haskell
- Erlang
- F#
- OCaml
- Clojure
- Swift*
- D, Lisp, ML...

*<https://leverich.github.io/swiftislikescala/>

Opinião

- Scala é uma boa linguagem para começar a estudar o paradigma funcional
- Scala pode te ajudar a desenvolver alguns métodos em seu projeto usando a JVM
- O uso de Scala pode ser problemático em projetos grandes, que envolvam muitos colaboradores

Referências

- Programming in Scala 2nd ed. - M. Odersky
- Scala Cookbook - A. Alexander
- Programming Scala - D. Wampler & A. Payne
- Princípios da Programação Funcional em Scala - M. Odersky (Coursera)
- Scala on Android - G. Couprie
- Programming F# 3.0 2nd ed. - C. Smith
- Learn You a Haskell for Great Good - M. Lipovaca
- Learn You Some Erlang for Great Good - F. Hébert
- Programação Funcional com a Linguagem Haskell - A. Du Bois
- Paradigma Funcional, Caso de estudo: Haskell - S. Costa
- Java in a Nutshell 5th ed. - D. Flanagan



Alguma Pergunta?

Gustavo Fernandes dos Santos
`gfdsantos@inf.ufpel.edu.br`

