Linguagens de Programação

Programação Funcional

Carlos Bazilio
carlosbazilio@id.uff.br
https://carlosbazilio.github.io

Motivação
Haskell

The program written in Java	The program written in Haskell

`final int LIMIT=50;`
`int[] a = new int[LIMIT];`
`int[] b = new int[LIMIT - 5];`
`for (int i=0;i < LIMIT;i++) {`
`  a[i] = (i+1)*2;`
`  if (i >=5) b[i - 5] = a[i];`
`}`	

Motivação
Haskell

The program written in Java	The program written in Haskell

`final int LIMIT=50;`
`int[] a = new int[LIMIT];`
`int[] b = new int[LIMIT - 5];`
`for (int i=0;i < LIMIT;i++) {`
`  a[i] = (i+1)*2;`
`  if (i >=5) b[i - 5] = a[i];`
`}`
	`let a = [2,4..100]`
	`let b = drop 5 a`

Motivação – Java 8

```
List <Pessoa>pessoas = new ArrayList<Pessoa>();
pessoas.add(new Pessoa("Fulano", "Silva", 40, 75.0));
pessoas.add(new Pessoa("Ciclano", "Silva", 25, 120));
pessoas.add(new Pessoa("Beltrano", "Silva", 70, 55));

Iterator <Pessoa>it = pessoas.iterator();
while (it.hasNext()) {
        System.out.println(it.next());
}

for (Pessoa p : pessoas)
        System.out.println(p);

pessoas.forEach(System.out::println);
```

Motivação

- Java 8
- ECMAScript 6, ELM
- PHP 5.3, HACK
- C# 2.0, F#
- C++11
- Objective C (blocks)
- Scala, Clojure, Kotlin
- Erlang
- R, Julia, ...

Impulso para o Curso

- Palestra do Professor Clóvis de Barros

https://www.youtube.com/watch?v=9Z6TSuJQ2oE

Funções Exemplo

maximo(a,b) = a, se a > b

b, c.c.

$$H_{100} = \sum_{i=1}^{100} \frac{1}{i}$$

Recursão

- Técnica básica em programação funcional
  - f(x) = g(x) ? f(x-1)
  - f(mínimo) = g(mínimo)
- Exemplo (fatorial):
- 4! = 4*3! = 4*3*2! = 4*3*2*1! = 4*3*2*1*0! = 4*3*2*1*1 = 24
- n! = n*(n-1)! = n*(n-1)*(n-2)! = … = n*(n-1)*(n-2)*..*1
- fat 0 = 1
- fat n = n * fat (n-1)

Recursão

- Técnica básica em programação funcional
  - f(x) = g(x) ? f(x-1)
  - f(mínimo) = g(mínimo)
- Exemplo (fatorial):
- 4! = 4*3! = 4*3*2! = 4*3*2*1! = 4*3*2*1*0! = 4*3*2*1*1 = 24
- n! = n*(n-1)! = n*(n-1)*(n-2)! = … = n*(n-1)*(n-2)*..*1
- fat 0 = 1
- fat n = n * fat (n-1)

Recursão

- Como calcular uma combinação em Haskell?

comb (n, p) = n! / p! * (n – p)!

```
comb n p = fat n / (fat p) * (fat (n - p))
```

Funções Exemplo

maximo(a,b) = a, se a > b

b, c.c.

$$H_{100} = \sum_{i=1}^{100} \frac{1}{i}$$

Paradigma Funcional

- Como era de se esperar, funções são um recurso importantíssimo em linguagens funcionais
- Tecnicamente falando, funções são valores de 1a classe em linguagens de programação funcionais
- O que isto quer dizer?

Funções como Valores de 1a. Classe
Atribuição a Variáveis

a = f()

g = f

b = g()

Funções como Valores de 1a. Classe
Passagem de Parâmetros

a = f()

b = g(f)

Funções como Valores de 1a. Classe
Passagem de Parâmetros

a = f()

b = g(f)

$$H_{100} = \sum_{i=1}^{100} \frac{1}{i}$$

Funções como Valores de 1a. Classe
Retorno de Funções

a = f()

b = a()

Composição

f(g(x))

Imutabilidade

f(a) == f(b) sempre que a == b

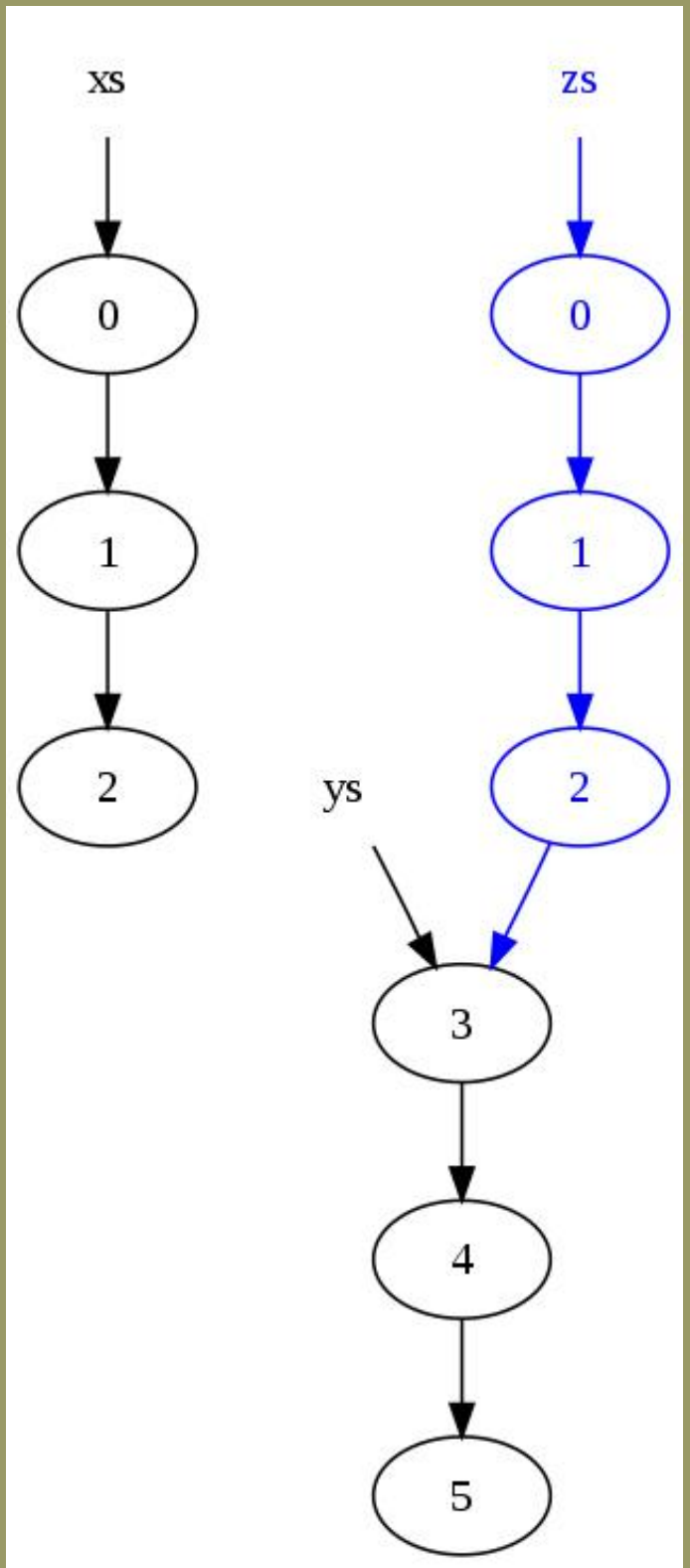
Atribuição

```
x = 0;
x = x + 1;
```

```
x = 0;
y = x + 1;
```

Persistent Data Structure

xs = [0, 1, 2]
ys = [3, 4, 5]
zs = xs ++ ys



Funções Puras

- Funções que atendem 2 requisitos:
  - Seus dados de entrada são necessariamente passados por parâmetro e de forma imutável;
  - Produzem seu resultado sem alterar qualquer valor fora da função (sem efeito colateral).
- Vantagens:
  - Manutenção mais simples
  - Permite otimização
  - Simplifica implementação de programas concorrentes

Características de uma Linguagem Funcional

- Numa linguagem funcional, tudo, até mesmo o próprio programa, é uma função
- Quando falamos função, pensamos no conceito matemático deste termo:

$$f(a, b, \ldots) : Dom \rightarrow CDom$$

  - Uma função f mapeia valores de Dom em Cdom
  - f pode mapear cada valor de Dom a, no máximo, 1 único valor de CDom
  - Se aplicamos f seguidamente para os mesmos valores de entrada (valores para a, b, …), obtemos sempre o mesmo valor resultante (função pura)

Características de uma Linguagem Funcional

- Pelo fato de tudo ser função, algumas características interessantes no uso de funções são facilmente descritas nestas linguagens:
  - Passagem de funções por parâmetro
  - Retorno de funções por outras funções
  - Composição de funções
  - Funções anônimas
  - Chamadas parciais de funções

Exemplo Scheme

- Originária da linguagem LISP, que é precurssora da maioria das linguagens funcionais
- Expressões nesta linguagem são escritas de forma parentizada e pré-fixada

(fnç arg1 arg2 … argn)

- Principal distribuição: DrScheme

Exemplos Scheme

```
(define fatorial
  (lambda (n)
    (if (= n 0)
        1
        (* n (fatorial (- n 1))))))
```

Exemplos
Scala

- Linguagem que mistura os paradigmas funcional e orientado a objetos
- Ganhou notoriedade recente por ter sido adotada pelo Twitter em detrimento da linguagem Ruby (questões de desempenho)

Exemplos Scala

```
def loopWhile(def cond: => Boolean)(def body: => Unit):
  Unit = {
  if (cond) {
    body;
    loopWhile(cond)(body);
  }
}

var i = 10;
loopWhile (i > 0) {
  Console.println(i);
  i = i - 1
}
```

Define método loopWhile

Sintaxe comum !!!

Exemplos Haskell

```
fatorial :: Int -> Int
fatorial 0 = 1
fatorial n = fatorial (n-1) * n
```

OU

```
fatorial n
    | n == 0 = 1
    | otherwise = fatorial (n-1) * n
```

OU

```
fatorial n = if n == 0 then 1
                       else fatorial (n-1) * n
```

Haskell - Alguns Links

- Site principal: http://www.haskell.org/haskellwiki/Haskell
- WikiBook: http://en.wikibooks.org/wiki/Haskell
- Tutoriais:
- http://learnyouahaskell.com/https://www.schoolofhaskell.com/ http://haskell.tailorfontela.com.br/https://www.haskell.org/tutorial/index.html
- Bibliotecas: https://downloads.haskell.org/~ghc/latest/docs/html/libraries/

Haskell - Alguns Links

- Pesquisa:
- https://www.haskell.org/hoogle/
- Cursos
- http://www.seas.upenn.edu/~cis194/fall16/
- Softwares em Haskell:
- http://elm-lang.org/
- https://www.yesodweb.com/

Listas

- Tipo coleção comum em linguagens funcionais
- São representadas por valores entre “[“ e “]”
  - ['a', 'b', 'c'] ou [1, 2, 3]
- Para varrer uma lista utilizamos o operador “:”
  - ['a', 'b', 'c'] equivale a 'a' : ['b', 'c']
  - ['a', 'b', 'c'] equivale a cab : Resto ; *onde cab = 'a' e Resto = ['b', 'c']*
- [] representa uma lista vazia

Exercícios

- Crie uma função que calcule a soma de uma lista de números

Casamento de Padrões

```
soma [] = 0

soma (cab:resto) = cab + soma resto

soma (cab1:cab2:resto) = cab1 + cab2 + soma resto


tamanho [] = 0

tamanho (_:resto) = 1 + tamanho resto
```

Exercícios

- Crie uma função que dada uma lista de números retorne a lista dos ímpares (função odd)
- Faça o mesmo para uma lista de pares (função even)

Exercícios

- Como podemos representar uma matriz?
- Crie uma função para, dada uma matriz, um número de linha e outro de coluna, retorne o valor existente nesta posição
- Crie uma função para, dada 2 listas ordenadas, fazer um *merge* desta, ou seja, gerar uma nova lista ordenada resultante da junção destas 2

Listas
Funções Pré-definidas

- head, tail, last, init, length, null, reverse, take, drop, maximum, minimum, sum, product, elem (pertence infixado), ++, ...
- Faixas: [1 .. 20], [2, 4 .. 20], [20, 19 .. 1]
- Compreensão de listas:
  - [x*2 | x <- [1..10]]
  - [x*2 | x <- [1..10], x*2 >= 12]
  - [x | x <- [50..100], x `mod` 7 == 3]

Exercícios

- Refaça a função que dada uma lista de números retorne a lista dos ímpares (função odd) usando compreensão de listas

Exemplos Haskell

```
[1 ..] -- Lista infinita (looping)

[x * x | x ← [1 ..]] -- Lista infinita (looping)

take 100 [x * x | x ← [1 ..]]
```

Exemplos Haskell

```
s? []     = []
s? (x:xs) = s? [y | y<-xs, y<x ] ++
            [x                 ] ++
            s? [y | y<-xs, y>=x]
```

- O que faz a função s?

Haskell

- Linguagem funcional que mais se aproxima de definições matemáticas
- É fortemente tipada e de tipagem estática
- Possui um sistema de tipos bastante rebuscado, fazendo inferência de tipos quando estes não são fornecidos

Haskell
Tipos Pré-Definidos

Tipo	Valores
Int	-100, 0, 1, 50, ...
Integer	-3333333, 3, 32408034852039504395204, ...
Float/Double	-3.2223433, 0.0, 3.0, ...
Char	'a', 'z', 'A', 'Z', '&', ...
String	“Bazilio”, “Linguagens”, ...
Bool	True, False

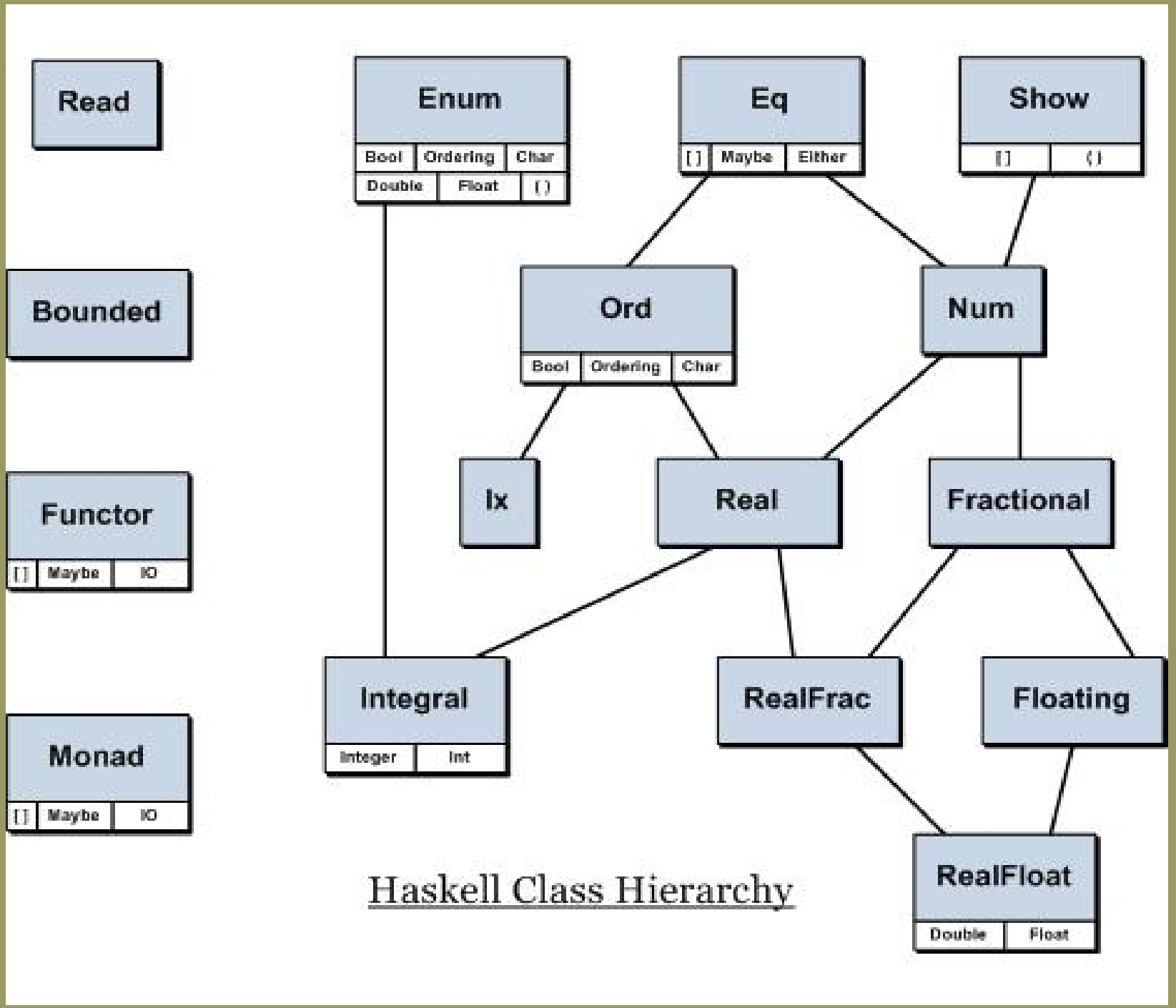
Exercícios

- Use a função seno para calcular o seno de um número, Por exemplo, seno de Pi / 2

```
> sin (pi / 2)
```

- Utilize o comando :t para verificar o tipo da função seno

```
> :t sin
```


Read
Bounded
Functor
[] Maybe IO
Monad
[] Maybe IO
Enum
Bool Ordering Char
Double Float ()
Eq
[] Maybe Either
Show
[] ()
Ord
Bool Ordering Char
Num
Ix
Real
Fractional
Integral
Integer Int
RealFrac
Floating
RealFloat
Double Float
Haskell Class Hierarchy

Exercícios

- Use a função length para calcular o tamanho de uma lista

```
> length ['a' .. 'w']
```

- Utilize o comando :t para verificar o tipo da função length

```
> :t length
```

Funções Genéricas / Polimórficas

- Assinatura da função maximo:

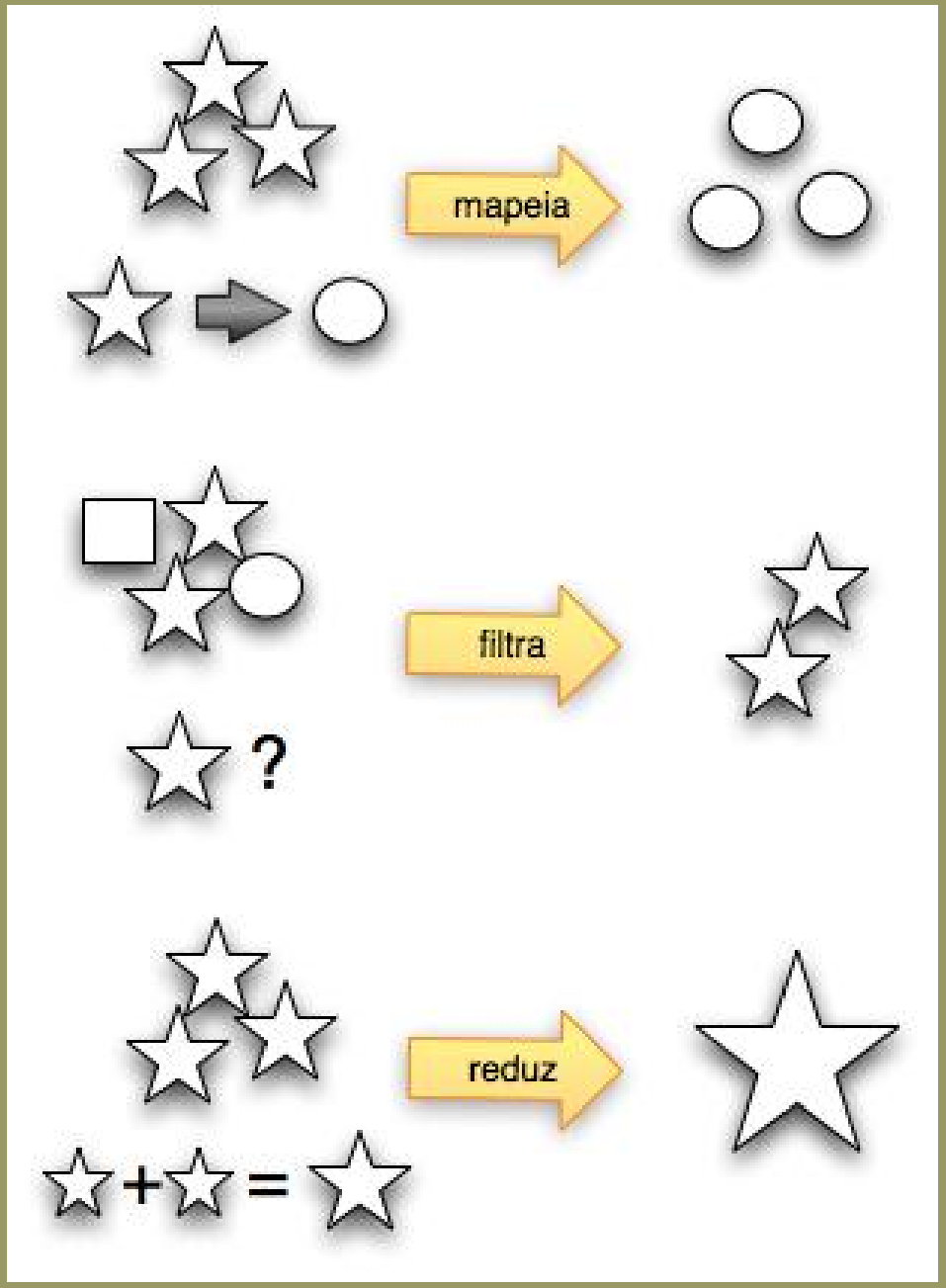
```
maximo :: Ord a => a -> a -> a
```

- Ord é a classe (conjunto) dos tipos ordenáveis em Haskell
- a representa o tipo genérico
- a -> a -> a indica que a função recebe um primeiro valor, recebe um segundo e retorna um terceiro, todos do mesmo tipo

Exercícios

- Utilize o comando :t no prompt do Haskell para verificar o tipo das funções já criadas
- Chame a função maximo para valores que não sejam apenas números inteiros

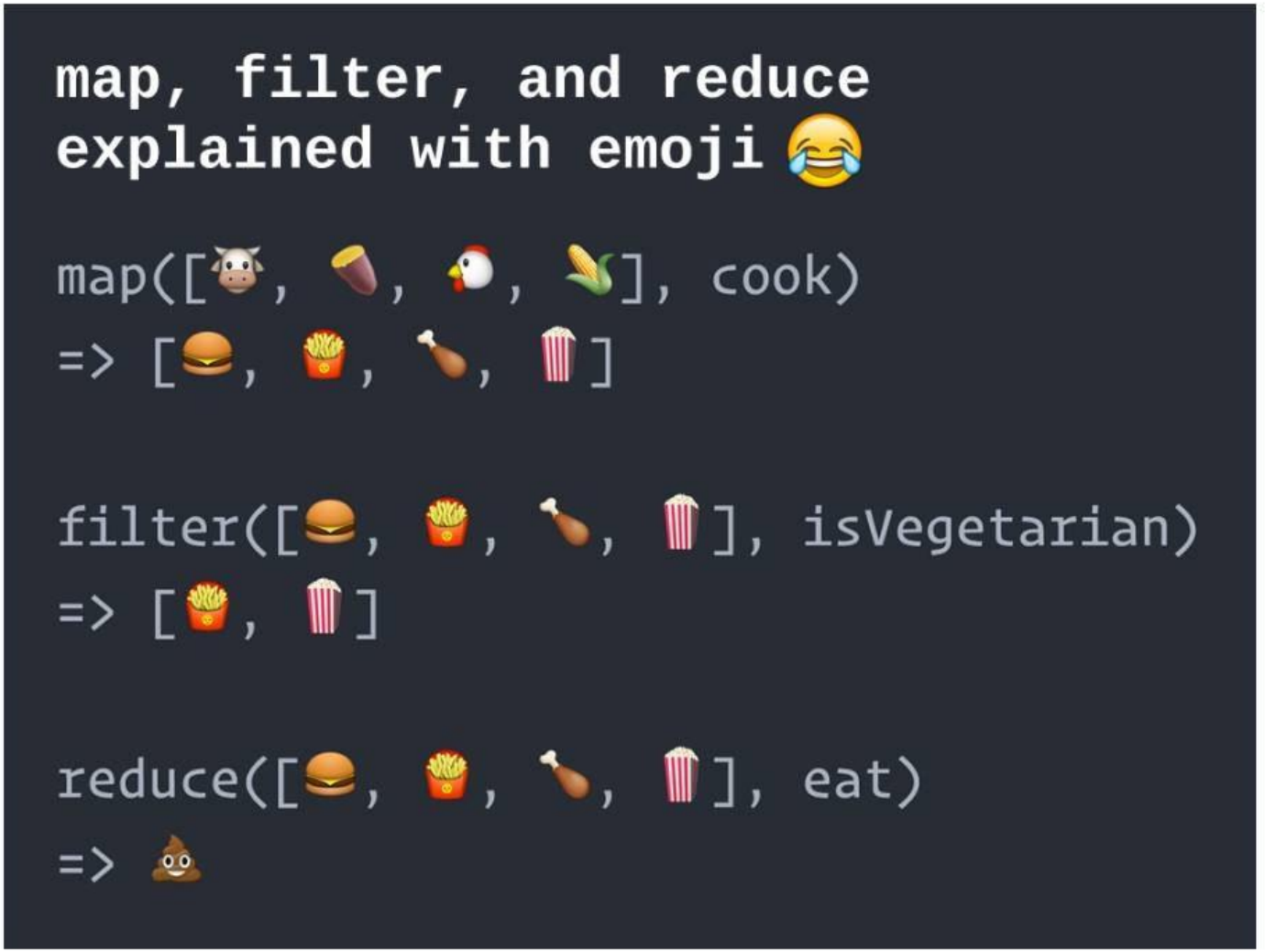
Funções de Alta Ordem



Funções de Alta Ordem

Map, filter, and reduce explained using emoji

by JOEY DEVILLA on JUNE 23, 2016



Exercícios

- Generalize as funções filtra par e filtra ímpar feitas anteriormente, ou seja, crie uma única função que possa realizar os 2 tipos de filtragem

Funções de Alta Ordem

- Funções que usualmente recebem outras funções como parâmetro para realização de suas tarefas
- Exemplos:
  - Mapeia: recebe um conjunto de valores e uma função e retorna o conjunto com a função aplicada a cada valor deste
  - Filtra: recebe um conjunto e uma função teste e retorna um subconjunto que satisfaça a função
  - Acumula: recebe um conjunto e uma função e retorna o resultado da aplicação da função a todos os elementos do conjunto

Funções de Alta Ordem Haskell

- Em Haskell, as seguintes funções pré-definidas estão disponíveis: map, filter, foldl, foldr
- Abaixo seguem alguns exemplos de uso:

```
> let exemplo = ["a b c", "d e", "f g h I"]

> map length exemplo

[5,3,7]

> filter (\x -> x /= ' ') "a b c"

"abc"
```

Funções de Alta Ordem Haskell

(… Continuação ...):

```
> let exemplo = ["a b c", "d e", "f g h I"]

> map (filter (\x -> x /= ' ')) exemplo

["abc","de","fghi"]

> map length (map (filter (\x -> x /= ' ')) exemplo)

[3,2,4]

> foldl (+) 0 (map length (map (filter (\x -> x /= ' ')) exemplo))

9
```

Exercícios

- Crie uma função que receba uma lista de strings e retorne uma lista de tamanhos de strings
- Combine esta com a função de soma de números para somar os tamanhos das listas
- Utilize a função foldl para realizar esta mesma soma

Exercícios

- O que a função f abaixo computa?

```
igual pal x = pal == x

f [] palavra = 0

f l palavra = length (filter (igual palavra) (words l))
```

Currying e Funções Parciais

- Mecanismo que permite maior flexibilidade na utilização das funções
- Considere a assinatura da função máximo:

```
maximo :: Ord a => a -> a -> a
```

- Essa definição indica que a função trata cada argumento 1 a 1
- Experimente fazer a seguinte chamada. O que vale x?

```
> let x = maximo 10
```

Currying e Funções Parciais

- Observe a função ocorrencias abaixo:

```
igual pal x = pal == x

ocorrencias [] palavra = 0

ocorrencias l palavra = length (filter (igual
  palavra) (words l))
```

- Observe o uso da função igual dentro de ocorrencias!

Função Lambda / Anônima

- Ainda observando a função ocorrencias:

```
igual pal x = pal == x

ocorrencias [] palavra = 0

ocorrencias l palavra = length (filter (igual
  palavra) (words l))
```

- A função igual não parece simples demais !?
- Que tal a criarmos apenas no momento da chamada?

```
(\x -> x == palavra)
```

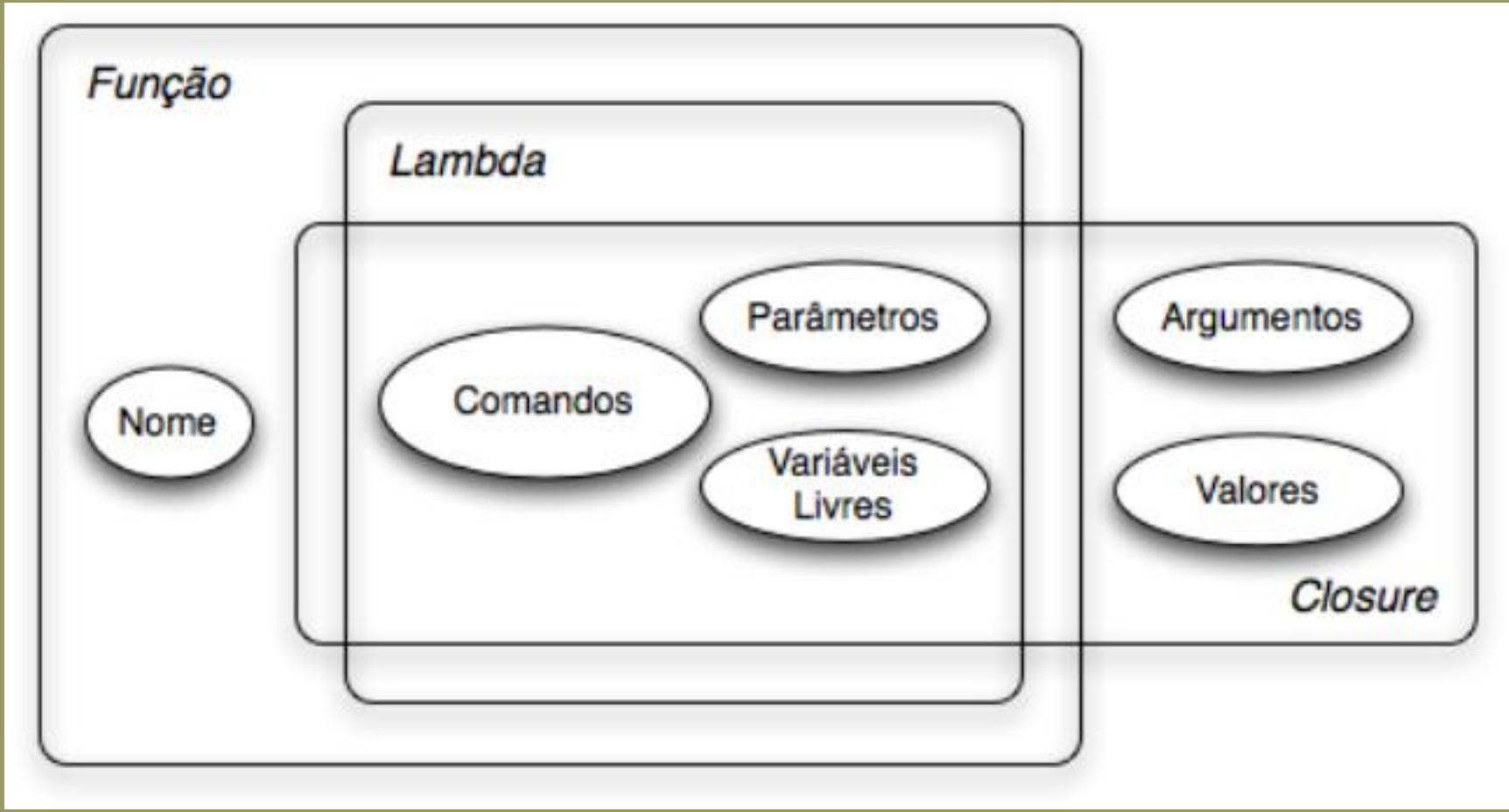
Funções Lambda

```
void map (int vet [100], int f (int)) {
    for (int i = 0; i < 99; i++)
        vet[i] = f(vet[i]);
}

int inc (int x) {
    return x++;
}

main() {
    int vetor [100] = {1, 2, 3, …, 99};
    map (vetor, inc);
    map (vetor, x++); // Versão Lambda
}
```

Funções e afins



Closure

“An object is data with functions. A closure is a function with data.” — John D. Cook

Tuplas

- Tuplas são um recurso em Haskell que permite a combinação de valores de tipos diferentes, como um tipo registro.
  - (1, “Fulano”, 9999, “Rua A”)
- Como listas são monotipadas, tuplas são um mecanismo que permite o agrupamento de valores com tipos diferentes
- Para tuplas com 2 valores, as funções pré-definidas *fst* e *snd* retornam 1o. e 2o. valores
- Para outros tamanhos, basta definir:
  - third (_, _, x, _) = x

Tipos Sinônimo

- Tipos sinônimo são uma maneira de chamar tipos conhecidos de uma outra forma (alias)
- Em Haskell definimos estes com a palavra reservada 'type'. Exemplos:
  - type Caracter = Char
  - type CPF = [Caracter]
  - type Nome = [Caracter]
  - type Pessoa = (Nome, CPF)
- Sinônimos podem ser utilizados para tipos isolados, listas ou tuplas

Tipo do Usuário

- O tipo pré-definido Bool pode ser definido da seguinte forma:

```
data Bool = False | True
```

- data indica que estamos definindo um tipo
- Bool é o nome do tipo definido
- False e True são construtores para os possíveis valores de Bool (neste caso, 2 apenas)

Tipo do Usuário

- Imagine que queiramos definir um tipo chamado Figura
- Um círculo poderia ser representado como a tupla (10.0, 15.5, 5.3), onde os 2 primeiros valores são o centro do círculo e o 3o. o raio
- Entretanto, isso poderia ser qualquer outro dado com 3 valores (um ponto no espaço, p.ex)
- Vamos então criar um tipo específico

```
data Figura = Circulo Float Float Float
```

Tipo do Usuário

```haskell
1  -- type Circulo = (Float, Float, Float)
2  data Figura = Circulo Float Float Float
3              deriving (Show)
4
5  area :: Figura -> Float
6  area (Circulo x y raio) = pi * raio * raio
```

Exercícios

- Crie o tipo Figura e verifique seu tipo no prompt
- Adicione o tipo Retangulo, o qual terá 2 coordenadas
- Crie um método para o cálculo da área de uma figura
- Crie o tipo Ponto e altere a definição de Figura para usá-lo

Exemplos de Definições de Tipos

```
webApp :: Request -> Response

databaseQuery :: Query -> Database -> ResultSet

databaseUpdate :: Update -> Database -> Database

game :: Events -> GameState -> GameState
```

Referências

- Site principal: http://www.haskell.org/haskellwiki/Haskell
- Instalação: http://www.haskell.org/platform/
- WikiBook: http://en.wikibooks.org/wiki/Haskell
- Tutoriais:
- http://learnyouahaskell.com/https://www.schoolofhaskell.com/ http://haskell.tailorfontela.com.br/https://www.haskell.org/tutorial/index.html

Referências

- Exemplos reais: http://book.realworldhaskell.org/read/
- Na Indústria: https://wiki.haskell.org/Haskell_in_industry http://en.wikibooks.org/wiki/Haskell
- Wikipedia:https://www.wikiwand.com/en/Functional_programming
- http://www.drdobbs.com/architecture-and-design/in-praise-of-haskell/240163246?elq=03bf1275b97046718cf499071256044e

Referências

- Bibliotecas:
- https://downloads.haskell.org/~ghc/latest/docs/html/libraries/
- Pesquisa: https://www.haskell.org/hoogle/

Looping de um Jogo

```
// Imperativo
var gameState = initGameState();
while (!gameState.isQuit()) {
    var events = pollForEvents();
    updateGameState(gameState, events);
    renderGameState(gameState);
}

// Funcional
play = loop initialGameState
loop current = do
    events <- pollEvents
    let next = updateGameState current events
    renderGameState next
    loop next
```