




Paradigmas de Programação

# Linguagem Haskell



Prof<sup>a</sup> Andréa Schwertner Charão  
DLSC/CT/UFSM



# Haskell opportunities at Facebook






 andrea@inf.ufsm.br ▾

Grupos do Google





[Haskell-cafe >](#)  
[Haskell-cafe] Haskell opportunities at Facebook  
2 postagens de 2 autores ▾ 

 **Bryan O'Sullivan** 4 de fev 

★ **Outros destinatários:** haskel...@haskell.org

[Traduzir mensagem para português](#)

Hi, friends –

I have a number of very interesting openings at Facebook HQ in Menlo Park, California. There are two different teams hiring.

The first set of positions are for an entirely new team. This project involves distributed systems, data mining, and machine learning. There may be roles on this team for less experienced candidates in a few months, but right now we are looking for people who have written a reasonable amount of Haskell, have built real production systems in some language or other (and have the scars to prove it), and can contribute in major ways to the design and construction of a demanding new system that we're building from scratch.

The second set of positions are for a cousin team, which is building on the success of our Haxl project to extend our capability to fight spam and malware. For these roles, we're open to a broader range of experience levels.

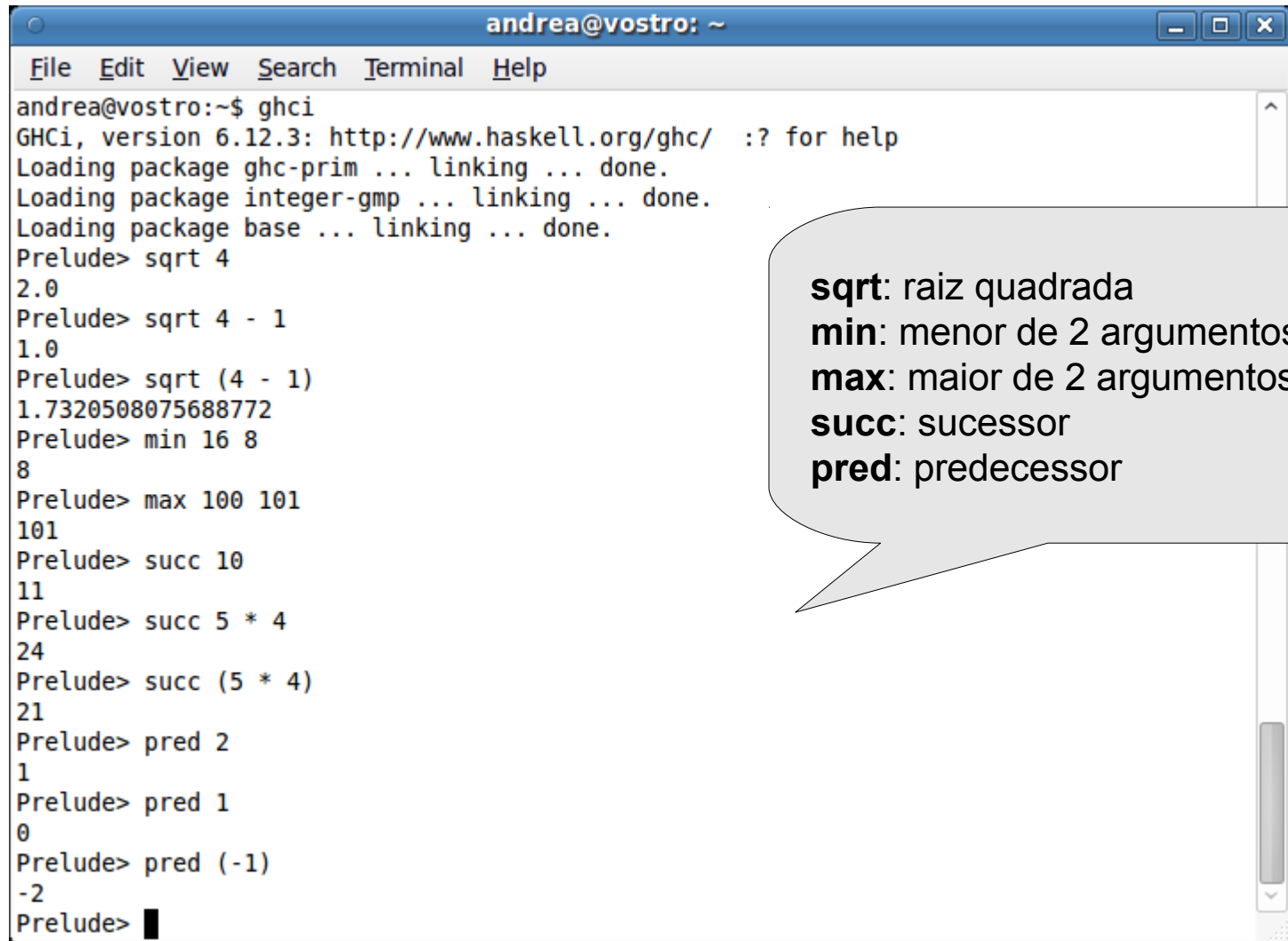
If you're interested, please drop me an email with a current CV.

Cheers,  
Bryan.

# Introdução

- Linguagem puramente funcional
- Compilada ou interpretada
  - GHC, Hugs
- Tipos de dados: Int, Integer, Bool, String, Char, Float, Double, etc.
- Fortemente tipada, com inferência de tipos
- Muitas bibliotecas de funções pré-definidas
- Listas homogêneas suportadas nativamente

# Aplicando funções pré-definidas



```
andrea@vostro: ~  
File Edit View Search Terminal Help  
andrea@vostro:~$ ghci  
GHCi, version 6.12.3: http://www.haskell.org/ghc/ :? for help  
Loading package ghc-prim ... linking ... done.  
Loading package integer-gmp ... linking ... done.  
Loading package base ... linking ... done.  
Prelude> sqrt 4  
2.0  
Prelude> sqrt 4 - 1  
1.0  
Prelude> sqrt (4 - 1)  
1.7320508075688772  
Prelude> min 16 8  
8  
Prelude> max 100 101  
101  
Prelude> succ 10  
11  
Prelude> succ 5 * 4  
24  
Prelude> succ (5 * 4)  
21  
Prelude> pred 2  
1  
Prelude> pred 1  
0  
Prelude> pred (-1)  
-2  
Prelude> █
```

**sqrt**: raiz quadrada

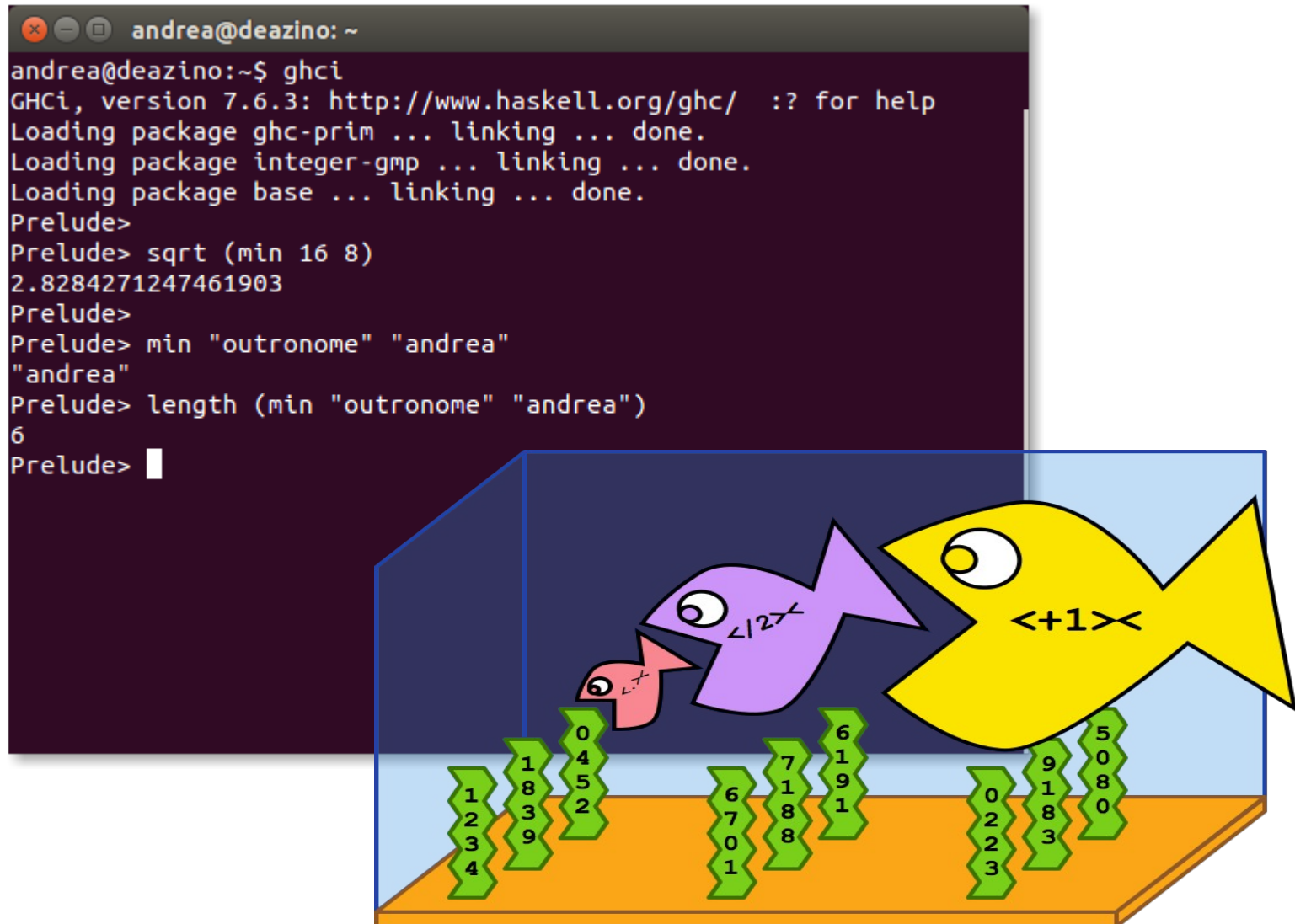
**min**: menor de 2 argumentos

**max**: maior de 2 argumentos

**succ**: sucessor

**pred**: predecessor

# Aplicando funções pré-definidas



# Definindo funções

- Exemplo de definição **sem** tipo explícito

quadrado x = x \* x

- Exemplos de aplicação

> quadrado 2

4

> quadrado 2.0

4.0

> quadrado 2.5

6.25

Colocar isso num arquivo .hs

Inferência de tipo

# Definindo funções

- Exemplo de definição **com** tipo explícito

`quadrado :: Integer -> Integer`

`quadrado x = x * x`

Colocar isso num arquivo .hs

- Exemplos de aplicação

`> quadrado 2`

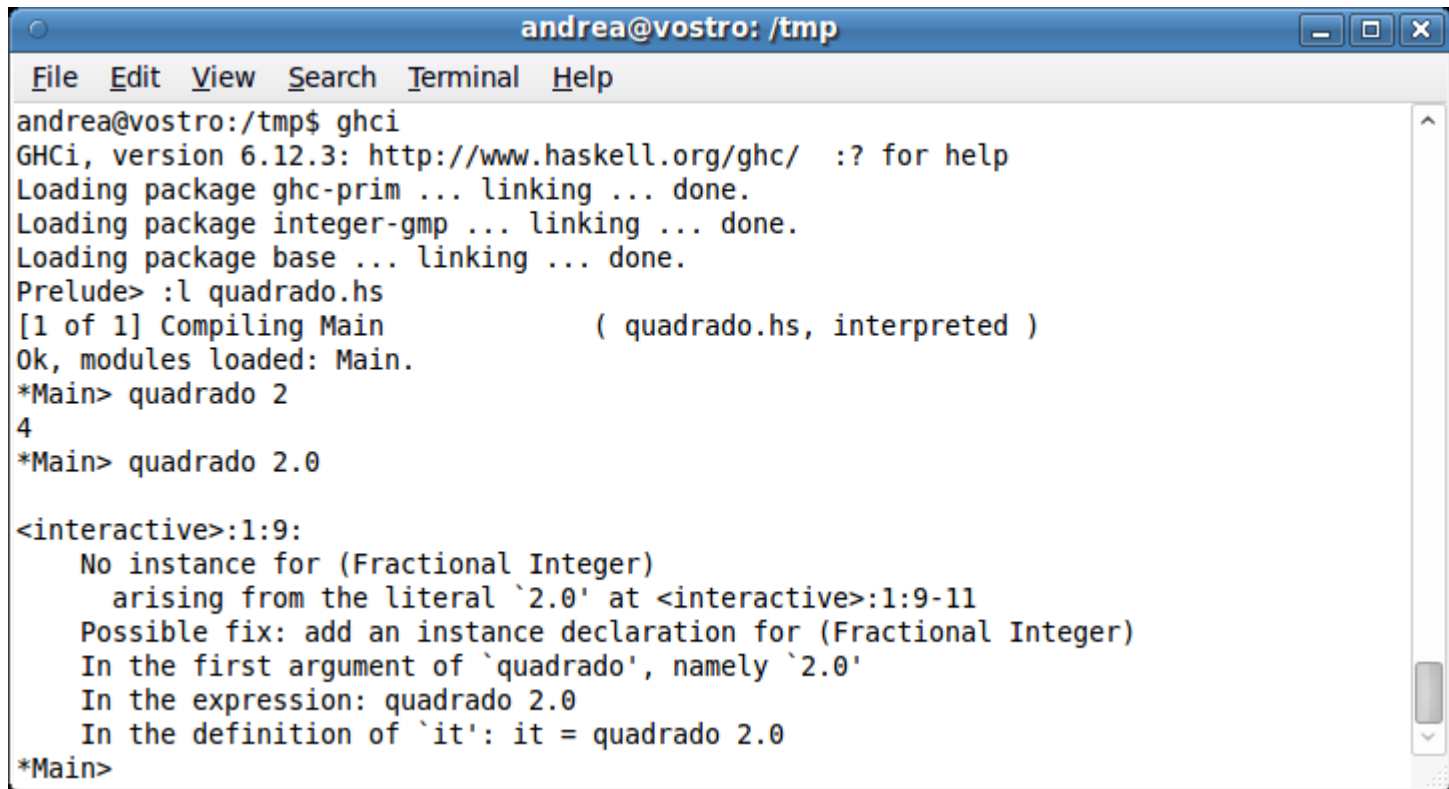
`4`

`> quadrado 2.0`

Errado, pois argumento deveria ser inteiro!

# Definindo funções

## ■ Exemplo de erro de aplicação



```
andrea@vostro: /tmp
File Edit View Search Terminal Help
andrea@vostro:/tmp$ ghci
GHCi, version 6.12.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> :l quadrado.hs
[1 of 1] Compiling Main                ( quadrado.hs, interpreted )
Ok, modules loaded: Main.
*Main> quadrado 2
4
*Main> quadrado 2.0

<interactive>:1:9:
  No instance for (Fractional Integer)
    arising from the literal `2.0' at <interactive>:1:9-11
  Possible fix: add an instance declaration for (Fractional Integer)
  In the first argument of `quadrado', namely `2.0'
  In the expression: quadrado 2.0
  In the definition of `it': it = quadrado 2.0
*Main>
```



# Definindo funções

- Funções podem ter qualquer número de argumentos
- Exemplo: função **sem argumento**

```
myPi :: Float
```

```
myPi = 3.1416
```



Colocar isso num arquivo .hs

- Exemplo: função **com 2 argumentos**

```
hipotenusa :: Double -> Double -> Double
```

```
hipotenusa c1 c2 = sqrt (c1^2 + c2^2)
```

# Tipos de dados

- Lógicos: Bool (True/False)

- Numéricos

Tipo	Descrição	Exemplos
Int	Inteiros, precisão fixa (limite superior e inferior)	0, 1, 2 , 5, etc.
Integer	Inteiros, precisão arbitrária	1, 2, 3, -9, etc.
Float	Reais, precisão simples	5.5, 3e-9, etc.
Double	Reais, precisão dupla	5.4, 3e-8, etc.

- Alfanuméricos

Tipo	Descrição	Exemplos
Char	Caracteres (Unicode)	'a', 'b', '\97', etc.
String	Lista de caracteres	"abc", etc.

# Listas

- Conjunto de dados de um mesmo tipo

- Exemplos

[1,2,3,4]      (lista de Integer/Int)

['a', 'b', 'c']    (lista de Char = String)

“abcd”      (abrev. lista de Char = String)

[]      (lista vazia)

[[1,2], [3,4]]    (lista aninhada)

- Exemplo de definição de função com lista

myfunc :: [Int] -> Int

# Funções com listas

> head [1,2,3,4]

1

> tail [1,2,3,4]

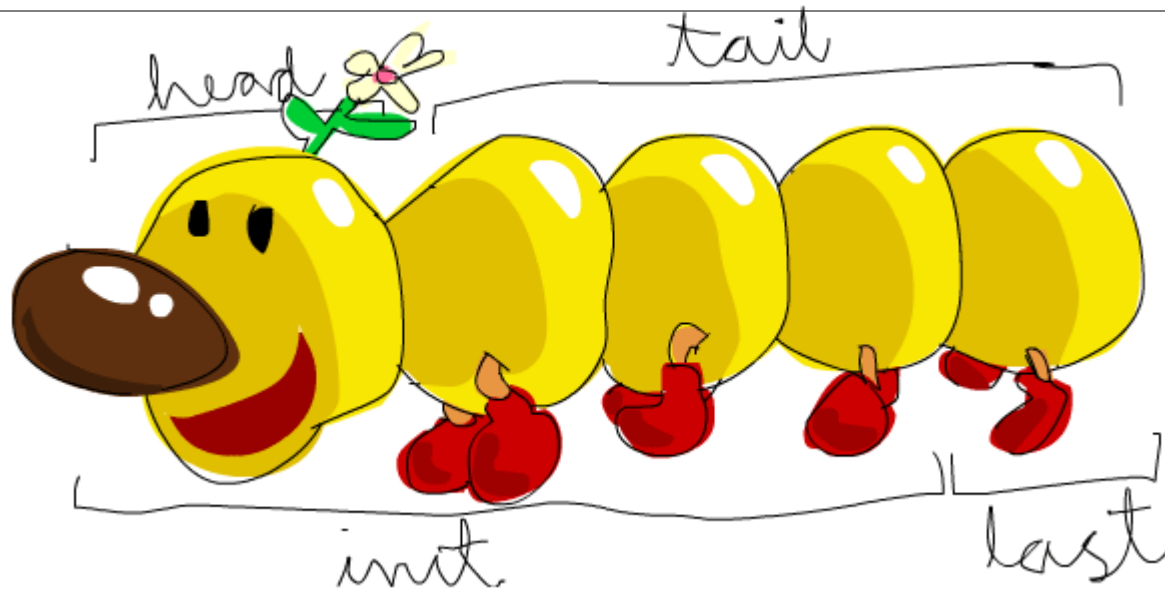
[2,3,4]

> init [1,2,3,4]

[1,2,3]

> last [1,2,3,4]

4



# Funções com listas

> head []

Erro!

> tail []

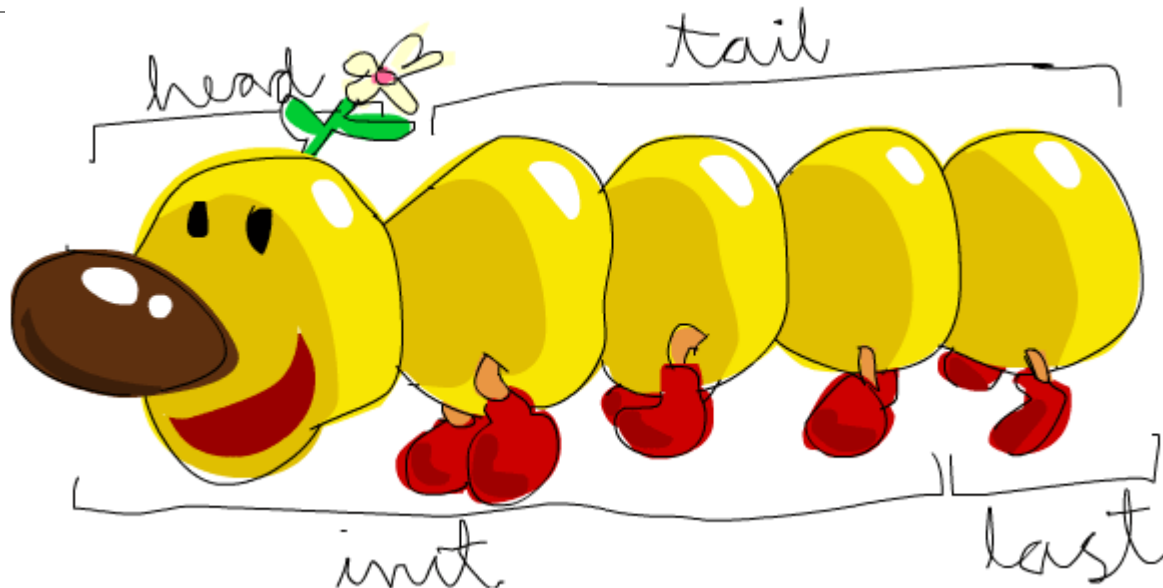
Erro!

> init []

Erro!

> last []

Erro!



# Funções com listas

```
> [1,2] ++ [3,4]
```

```
[1,2,3,4]
```

```
> sum [1,2,3,4]
```

```
10
```

```
> product [1,2,3,4]
```

```
24
```

```
> reverse [1,2,3,4]
```

```
[4,3,2,1]
```

```
> replicate 4 0
```

```
[0,0,0,0]
```

```
> length "abcd"
```

```
4
```

```
> maximum [1,8,2,4]
```

```
8
```

```
> minimum [2,8,1,4]
```

```
1
```

```
> null []
```

```
True
```

```
> null [1,2]
```

```
False
```

# Funções com listas

```
> take 3 [1,2,3,4,5]
```

```
[1,2,3]
```

```
> take 1 [3,6,9]
```

```
[3]
```

```
> take 5 [1,2]
```

```
[1,2]
```

```
> take 0 [1,2,3]
```

```
[]
```

```
> drop 3 [1,2,3,4,5]
```

```
[4,5]
```

```
> drop 4 [1,2,3,4,5]
```

```
[5]
```

```
> drop 100 [1,2,3]
```

```
[]
```

```
> drop 0 [1,2,3]
```

```
[1,2,3]
```

# Funções de alta ordem

- Do inglês: *higher order functions*
- Funções que recebem outras funções como argumento e/ou retornam funções
- Uso genérico
- Principais exemplos:
  - Mapear
  - Filtrar
  - Reduzir
  - Combinar



# Funções de alta ordem: map

- Aplica uma função a cada elemento da lista, produzindo outra lista
- Exemplo:

```
> map sqrt [4, 9, 16]  
[2.0, 3.0, 4.0]
```

# Funções de alta ordem: map

## ■ Exemplos:

```
> map (*5) [4, 9, 16]
```

```
[20,45,80]
```

```
> map (/2) [4, 9, 16]
```

```
[2.0,4.5,8.0]
```

```
> map (10/) [4,2]
```

```
[2.5, 5.0]
```

```
> map (subtract 5) [4,9,16]
```

```
[-1,4,11]
```

```
> map length ["ab", "cde", "fghi"]
```

```
[2,3,4]
```

# Funções de alta ordem: filter

- Aplica um teste a cada elemento da lista, produzindo outra lista somente com os elementos cujo teste resultar True
- Exemplo:

```
> filter (>3) [5,2,1,0,7,9]  
[5,7,9]
```

# Funções de alta ordem: filter

## ■ Exemplos:

```
> filter odd [4,5,1,0,8,7]
```

```
[5,1,7]
```

```
> filter (/=' ') "o rato roeu a roupa"
```

```
"oratoroeuaroupa"
```

```
> filter null [ "", "ab", "", "cd" ]
```

```
[ "", "" ]
```

```
> filter (== 0) [0,1,2,0,4,5,0]
```

```
[0,0,0]
```

# Funções de alta ordem: zipWith

- Aplica uma função a pares de elementos de 2 listas
- A função deve ser aplicável a 2 argumentos
- Exemplo:

```
> zipWith (+) [1,2,3] [4,5,6]  
[5,7,9]
```

# Funções de alta ordem: zipWith

## ■ Exemplos:

```
> zipWith replicate [1,2,3] ['a','b','c']  
["a","bb","ccc"]  
  
> zipWith take [1,2,1] ["ab", "cd", "ef"]  
["a","cd","e"]
```

E se a função a aplicar ainda não estiver definida?

Basta defini-la e aplicá-la!

# Funções anônimas (notação lambda)

- Recurso para definir funções rapidamente, sem precisar declarar seu nome e tipos de argumentos

```
> (\x -> x + 1) 10
```

```
11
```

```
> (\x y -> x + y*2) 10 10
```

```
30
```

```
> (\a -> length a < 5) [1,2]
```

```
True
```

## Mais sobre isso em...

- Sebesta, R. Conceitos de Linguagens de Programação. Bookman, 2011. Capítulo 15: Linguagens de programação funcional.

