# Matemática Funcional

Wladimir Araújo Tavares

Universidade Federal do Ceará - Quixadá

1 de Março de 2018

Iniciando em Haskell

Visualizando funções

Exercícios Resolvidos

Exercícios Propostos

# Cálculo das Áreas

```
quadrado x = x*x

area_retangulo x y = x*y

area_circulo r = pi * quadrado r

volume_paralelepipedo h l p = h*l*p
```

# Cálculo das Áreas

```
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude> :load lec02.hs
[1 of 1] Compiling Main             ( lec02.hs, interpreted )
Ok, modules loaded: Main.
*Main> quadrado 4
16
*Main> area_retangulo 2 3
6
*Main> :t quadrado
quadrado :: Num a => a -> a
*Main> area_circulo 3
28.274333882308138
*Main> :t area_circulo
area_circulo :: Floating a => a -> a
```

# Visualizando funções

```
soma a b = a+b

soma3 a b c = soma a (soma  b c)

media2 a b = (soma a b) / 2

media3 a b c = (soma3 a b c) / 3
```

# Visualizando funções

```
*Main> soma 2 3
5
*Main> soma3 2 3 5
10
*Main> :t soma
soma :: Num a => a -> a -> a
*Main> :t soma3
soma3 :: Num a => a -> a -> a -> a
*Main> :t soma3 4
soma3 4 :: Num a => a -> a -> a
*Main> media2 6 8
7.0
*Main> media3 6 8 8
7.333333333333333
```

# Visualizando funções em blocos

- hipotenusa $h = \sqrt{x^2 + y^2}$
- distância entre pontos
  $d_{AB}(x_1, y_1)(x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

# Visualizando funções

```
hipotenusa a b = sqrt ( quadrado a + quadrado b )

distancia (x1,y1) (x2,y2) = hipotenusa (x1-x2) (y1-y2)
```

# Visualizando funções

```
import Test.QuickCheck

quadrado x = x*x

prop_quadrado1 x = quadrado x >= 0

prop_quadrado2 x y =
  quadrado (x+y) == quadrado x + 2*x*y + quadrado y
```

# Visualizando funções

```
*Main> quickCheck prop_quadrado1
+++ OK, passed 100 tests.
*Main> quickCheck prop_quadrado2
+++ OK, passed 100 tests.
```

# Convenções sintáticas

- Os argumentos de funções são separados por espaços
- A aplicação tem maior precedência do que qualquer operador

| Matemática | Haskell |
|---|---|
| $f(x)$ | `f x` |
| $f(g(x))$ | `f (g x)` |
| $f(g(x), h(x))$ | `f (g x) (h x)` |
| $f(x, y) + 1$ | `f x y + 1` |
| $f(x, y + 1)$ | `f x (y + 1)` |

# Algumas funções da biblioteca Prelúdio

```
*Main> map (+2) [1..10]
[3,4,5,6,7,8,9,10,11,12]
*Main> [1,2,3] ++ [4,5,6]
[1,2,3,4,5,6]
*Main> let impar x = (mod x 2) == 1
*Main> filter impar [1..10]
[1,3,5,7,9]
*Main> head [1,2,3]
1
*Main> tail [1,2,3]
[2,3]
*Main> init [1..10]
[1,2,3,4,5,6,7,8,9]
*Main> last [1..10]
10
*Main> null []
True
*Main> null [1..10]
False
*Main> length [1..10]
10
*Main> length [1..100]
100
```

# Algumas funções da biblioteca Prelúdio

```
*Main> [2,3,4] !! 0
2
*Main> [2,3,4] !! 1
3
*Main> reverse [1,2,3,4,6]
[6,4,3,2,1]
*Main> 1:[3,4]
[1,3,4]
*Main> 1:[4,5]
[1,4,5]
*Main> (:) 1 [4,5]
[1,4,5]
*Main> (++) [1,2,4] [3,4,5]
[1,2,4,3,4,5]
*Main> :t head
head :: [a] -> a
*Main> :t reverse
reverse :: [a] -> [a]
*Main> :t map
map :: (a -> b) -> [a] -> [b]
*Main> :t filter
filter :: (a -> Bool) -> [a] -> [a]
*Main> :t (:)
(:) :: a -> [a] -> [a]
```

# Algumas funções da biblioteca Prelúdio

```
*Main> foldr (+) 0 [1,2,3,4,5]
15
*Main> sum [1,2,3,4,5]
15
*Main> foldr (&&) True [True, True, False]
False
*Main> and [True, True, False]
False
*Main> foldr (*) 1 [1,2,3,4,5]
120
*Main> product [1,2,3,4,5]
120
*Main> maximum [1..5]
5
*Main> minimum [1..5]
1
*Main> concat [[1,2,3],[4,5]]
[1,2,3,4,5]
```

# Algumas funções da biblioteca Prelúdio

```
*Main> take 10 [1..20]
[1,2,3,4,5,6,7,8,9,10]
*Main> take 5 [1..20]
[1,2,3,4,5]
*Main> drop 5 [1..20]
[6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
*Main> splitAt 6 "Hello World!"
("Hello ","World!")
*Main> splitAt 5 [1..20]
([1,2,3,4,5],[6,7,8,9,10,11,12,13,14,15,16,17,18,19,20])
*Main> takeWhile (<5) [1,2,6,6,7]
[1,2]
*Main> takeWhile (>5) [1,2,6,6,7]
[]
*Main> dropWhile (<5) [1,2,6,6,7]
[6,6,7]
*Main> zip [1,2,3] [4,5,6]
[(1,4),(2,5),(3,6)]
*Main> zipWith (+) [1,2,3] [4,5,6]
[5,7,9]
*Main> zipWith (*) [1,2,3] [4,5,6]
[4,10,18]
*Main> unzip [(4,5),(3,4)]
([4,3],[5,4])
```

# Exercícios Resolvidos

1. Dado três valores a, b e c, escreva uma função que retorne quantos dos três são iguais. A resposta pode ser 3 (todos iguais), 2 (dois iguais) ou 0 (todos diferentes)

```
1  iguais x y z | x == y && y == z =  3
2               | x == y && y /= z =  2
3               | x /= y && y == z =  2
4               | x /= y && x == z =  2
5               | otherwise = 0
```

# Exercícios Resolvidos

1. Dado três valores a, b e c, escreva uma função que retorne quantos dos três são iguais. A resposta pode ser 3 (todos iguais), 2 (dois iguais) ou 0 (todos diferentes)

```
1  iguais2 x y z =
2          if x == y then
3                  if y == z then 3
4                  else 2
5          else
6                  if y == z then 2
7                  else
8                          if x == z then 2
9                          else 0
```

# Exercícios Resolvidos

2 Dado três valores a, b e c, escreva uma função que retorne quantos desses numeros são maiores que o valor médio entre eles.

```
1   maioresMedia x y z =
2           if x > media then
3                   if y > media then
4                           if z > media then 3
5                           else 2
6                   else
7                           if z > media then 2
8                           else 1
9           else
10                  if y > media then
11                          if z > media then 2
12                          else 1
13                  else
14                          if z > media then 1
15                          else 0
16          where
17                  media = media3 x y z
```

# Exercícios Resolvidos

3 Implemente a função do or-exclusivo $a \text{ xor } b = (a \vee b) \wedge \neg(a \wedge b)$:

```
1   nao True = False
2   nao False = True
3
4   e False _ = False
5   e True  x = x
6
7   ou True _ = True
8   ou False x = x
9
10  xor True False = True
11  xor False True = True
12  xor False False = False
13  xor True True = False
14
15  xor2 a b = e (ou a b) (nao (e a b))
16
17  prop_e a b = (e a b) == (a && b)
18  prop_ou a b = (ou a b) == (a || b)
19  prop_xor a b = (xor a b) == (xor2 a b)
```

# Exercícios Resolvidos

4 : Implemente a função cabeca que seleciona o primeiro elemento de uma lista sem utilizar a função `head`.

```
1  cabeca1 xs = xs !! 0
2  cabeca2 xs = y
3           where (y:ys) = xs
```

# Exercícios Resolvidos

4 : Implemente a função cabeca que seleciona o primeiro elemento de uma lista sem utilizar a função `head`.

```
*Main> head []
*** Exception: Prelude.head: empty list
*Main> cabeca1 []
*** Exception: Prelude.!!: index too large
*Main> cabeca2 []
*** Exception: lec02.hs:85:15-25: Irrefutable pattern failed for
 ↪   pattern (y : ys)
```