Lista de Exercícios – Funções em Haskell

Respondido por Daniel Nunes Duarte

Conteúdo: https://profsergiocosta.notion.site/3-Fun-es-c7eaa942530f4ccbaadd0aed2f9ec13e

1. Funções básicas

1. 1

```haskell
-- Defina uma função quad que receba um número e devolva o seu quadrado.

quad :: Double -> Double
quad x = x *  x

main = do
   print (quad 5)
   print (quad 2.5)
   print (quad (-3))
```

1.2

```haskell
--Escreva uma função hipotenusa que receba os catetos de um triângulo retângulo e devolva a hipotenusa.

hipotenusa :: Floating a => a -> a -> a -> a

hipotenusa x y z = sqrt (x^2 + y^2 + z^2)



main = do

   print (hipotenusa 3 4 5)

   print (hipotenusa 5 12 13)

   print (hipotenusa 8 15 17)
```

---

2. Definições locais (`where` e `let`)

2.1

```
--Reescreva a função hipotenusa usando where para definir variáveis auxiliares.



hipotenusa x y z = sqrt xzy2
   where
      xzy2 = x2 + y2 + z2
      x2 = x^2
      y2 = y^2
      z2 = z^2


main = do
   print(hipotenusa 3 4 5) -- 7.0710678118654755
```

2.2

```
--Reescreva a mesma função usando let ... in ....
hipotenusa x y z = let
   x2 = x ^ 2
   y2 = y ^ 2
   z2 = z ^ 2
   xzy2 = x2 + y2 + z2
   in sqrt xzy2
main = do
   print(hipotenusa 3 4 5) -- 7.0710678118654755
```

---

3. Condicionais e guards

3.1

```
--1. Defina a função `sinal` usando `if-then-else` que devolva:
--    - `-1` se o número for negativo
--    - `0` se for zero
--    - `1` se for positivo
sinal num =
    if num < 0 then -1
    else if num == 0 then 0
    else 1
main = do
    print(sinal (-10)) -- -1
    print(sinal 0)     -- 0
    print(sinal 10)    -- 1
```

3.2

```
--Reescreva a função sinal usando guards.
sinal n
    | n < 0 = -1
    | n == 0 = 0
    | otherwise = 1
main = do
    print(sinal (-10)) -- -1
    print(sinal 0)     -- 0
```

```
  print(sinal 10)    -- 1
```

3.3

```
--Implemente a função classificaNota que recebe um número (0–10) e retorna:

--"Aprovado" se a nota for maior ou igual a 7

--"Recuperação" se a nota for maior ou igual a 5 e menor que 6.9

--"Reprovado" se a nota for menor que 5


classificarNota nota
  | nota < 5 = "Reprovado"
  | nota >= 5 && nota < 6.9  = "Recuperacao"
  | nota >= 7 = "Aprovado"
  | otherwise = "Nota inválida"


main = do
  print(classificarNota 4)    -- "Reprovado"
  print(classificarNota 5.5)  -- "Recuperação"
  print(classificarNota 8)    -- "Aprovado"
  print(classificarNota 11)   -- "Nota inválida"
```

---

4. Pattern Matching

4.1

```haskell
--Reescreva a função negar :: Bool -> Bool usando pattern matching.

negar :: Bool -> Bool

negar True = False

negar False = True




main = do

   print (negar True)   -- Output: False

   print (negar False)  -- Output: True
```

4.2

```haskell
-- 2. Defina a função `diasSemana` que recebe um número de 1 a 7 e retorna o nome do dia correspondente

-- (1 → "Domingo", 2 → "Segunda", etc.).

--    - Use **padrões explícitos**

diasSemana :: Int -> String

diasSemana 1 = "Domingo"

diasSemana 2 = "Segunda"

diasSemana 3 = "Terça"

diasSemana 4 = "Quarta"

diasSemana 5 = "Quinta"

diasSemana 6 = "Sexta"

diasSemana 7 = "Sábado"
```

```haskell
diasSemana _ = "Dia inválido"



main = do

  print (diasSemana 1)  -- Output: "Domingo"

  print (diasSemana 5)  -- Output: "Quinta"

  print (diasSemana 7)  -- Output: "Sábado"
```

4.3

```haskell
-- Defina a função fatorial com pattern matching nos casos base.

fatorial :: Int -> Int
fatorial 1 = 1
fatorial  0 = 0

fatorial n = n * fatorial (n - 1)



main = do
  print (fatorial 5)  -- Output: 120
  print (fatorial 0)  -- Output: 0
  print (fatorial 1)  -- Output: 1
  print (fatorial 6)  -- Output: 720
```

5. Recursão

5.1

```haskell
--Implemente a função potencia b e que calcula beb^ebe usando recursão (sem usar (^)).

potencia :: Int -> Int -> Int

potencia b e

  | e == 0 = 1

  | e == 1 = b

  | otherwise = b * potencia b (e - 1)
```

```haskell
main = do
    print (potencia 2 3)  -- Output: 8
    print (potencia 5 0)  -- Output: 1
    print (potencia 3 4)  -- Output: 81
    print (potencia 7 2)  -- Output: 49
```

5.2

```haskell
--Escreva uma função mdc a b que calcule o máximo divisor comum (algoritmo de Euclides).
mdc :: Int -> Int -> Int
mdc a 0 = a
mdc a b = mdc b (a `mod` b)
main = do
    print (mdc 48 18)  -- Output: 6
    print (mdc 56 98)  -- Output: 14
    print (mdc 101 10) -- Output: 1
    print (mdc 54 24)  -- Output: 6
```

5.3

```haskell
--Defina a sequência de Fibonacci de forma recursiva simples.
fibonacci :: Int -> Int
fibonacci 0 = 0
fibonacci 1 = 1
fibonacci n = fibonacci (n - 1) + fibonacci (n - 2)
main =  do
```

```
  print (fibonacci 0)  -- Output: 0
  print (fibonacci 1)  -- Output: 1
  print (fibonacci 5)  -- Output: 5
  print (fibonacci 10) -- Output: 55
  print (fibonacci 15) -- Output: 610
```

5.4

```
--Refaça a função de Fibonacci usando a técnica de recursão em cauda.
fibonacci :: Int -> Int
fibonacci n = fibHelper n 0 1
  where
    fibHelper 0 a _ = a
    fibHelper n a b = fibHelper (n - 1) b (a + b)
main = do
  print (fibonacci 0)  -- Output: 0
  print (fibonacci 1)  -- Output: 1
  print (fibonacci 5)  -- Output: 5
  print (fibonacci 10) -- Output: 55
  print (fibonacci 15) -- Output: 610
```