

Disciplina de Programação Funcional

3ª Lista de Exercícios

Curso de Engenharia de Computação

UEMG Ituiutaba

<https://bit.ly/2A0eUl8>

<https://github.com/mauro-hemerly/UEMG-2018-2>

1. Qual o tipo mais geral das seguintes funções?

- (a) `second xs = head (tail xs)`
- (b) `swap (x,y) = (y,x)`
- (c) `pair x y = (x,y)`
- (d) `double x = 2 * x`
- (e) `palin xs = reverse xs == xs`
- (f) `twice f x = f (f x)`

2. Qual a assinatura (o tipo mais geral) das seguintes funções?

- (a) `f1 x y = x < y`
- (b) `ff2 x y z = x == y || z`
- (c) `ff3 x y z = x == (y || z)`
- (d) `f4 x y = show x ++ y`
- (e) `ff5 x y = show (x ++ y)`
- (f) `f6 x y z = x + y > z`

Nota: para obter o tipo mais geral de uma qualquer expressão no ghci utilize o comando `:t`.

3. Verdadeiro ou falso?

- (a) f1 tem tipo
`Int -> Int -> Bool`
- (b) f1 tem tipo
`Integer -> Integer -> Bool`
- (c) f1 tem tipo
`Int -> Integer -> Bool`
- (d) f2 tem tipo
`[Char] -> [Char] -> Bool -> Bool`
- (e) f2 tem tipo
`[a] -> [a] -> Bool -> Bool`
- (f) f4 tem tipo

`Bool -> [Char] -> [Char]`

(g) f4 tem tipo

`(Int -> Int) -> [Char] -> [Char]`

4. Seja uma função com a seguinte assinatura

`(Ord a, Num b) => (a->a) -> a -> b`

O que podemos concluir sobre os seus parâmetros e resultado?

5. Determine um tipo e o valor para cada expressão.

- (a) `map (+1) [1..3]`
- (b) `map (>0) [3,-5,-2,0]`
- (c) `filter (>5) [1..6]`
- (d) `filter even [1..10]`
- (e) `filter (>0) (map (^2) [-3..3])`
- (f) `map (^2) (filter (>0) [-3..3])`
- (g) `map (++"s") ["A", "arte", "do", "aluno"]`
- (h) `map ("s"++) ["0", "aluno", "bem-comportado"]`

6. Usando *pattern matching* escreva funções e a assinatura que devolvam:

- (a) O primeiro elemento de um par
- (b) Um dado par com a ordem dos elementos trocados
- (c) O primeiro elemento de um triplo
- (d) Um dado triplo com os dois primeiros elementos trocados
- (e) O segundo elemento de uma lista
- (f) O segundo elemento do primeiro par de uma lista de pares

7. Qual a diferença entre as seguintes funções?

- (a) `f1 0 = 0`
`f1 x = x-1`
- (b) `f2 x = if x == 0 then 0 else x-1`

```
(c) f3 x = x-1 f3 0 = 0
(d) f4 x
    | x /= 0 = x-1
    | otherwise = 0
```

8. Qual a diferença entre as seguintes funções?

```
(a) add1 (x,y) = x + y
(b) add2 x y = x + y
```

Escreva a função

```
successor :: Int -> Int
```

recorrendo a cada uma delas.

9. As funções abaixo diferem? Se sim, como?

```
(a) hd1 (x:_) = x
(b) hd2 :: [Int] -> Int
    hd2 (x:_) = x
(c) hd3 :: [a] -> a
    hd3 (x:_) = x
```

10. Defina a função **trocaPares** que troca cada elemento de uma lista com o elemento seguinte, repetindo o processo de par em par de elementos. Se a lista contiver um número ímpar de elementos, o último elemento não é modificado.

11. Defina a função **retornaListaSup** que dada uma lista de inteiros e um número n, retorne outra lista contendo apenas de elementos de valor superior a n.

```
Exemplos:
Prelude> retornaListaSup 5 [3, 2, 5, 6, 9]
[6, 9]
```

12. Defina a função **removeDup** que remove os elementos repetidos em uma lista.

```
Exemplos:
Prelude> removeDup [1,2,5,2,5,1,3,5,1]
[1,2,5,3]
Prelude> removeDup ['a','b','c','a','d','b']
['a','b','c','d']
Prelude> removeDup "Ituiutaba"
"Ituiab"
```

13. Defina a função **dobraLista** que duplica os elementos em uma lista.

```
Exemplos:
Prelude> dobraLista [1,2,3,4]
[1,1,2,2,3,3,4,4]
Prelude> dobraLista "Ituiutaba"
"IIIttuuiiuuttaabbaa"
Prelude> dobraLista ["Ituiutaba","UEMG"]
["Ituiutaba","Ituiutaba","UEMG","UEMG"]
```

14. Defina a função **insereEmOrdem** que insere um inteiro em lista conforme ordem crescente dos seus elementos.

Exemplos:

```
Prelude> insereEmOrdem 15 [1,3..20]
[1,3,5,7,9,11,13,15,15,17,19]
```

15. Defina a função **encode** que recebe uma lista e retorna uma lista da quantidade de elementos repetidos em sequência.

Exemplos:

Suponha o seguinte dicionário:

```
Prelude> encode "carro"
[(1,'c'),(1,'a'),(2,'r'),(1,'o')]
Prelude> encode [1,1,2,2,2,2,3,3,4,5,5,5]
[(2,1),(5,2),(2,3),(1,4),(3,5)]
Prelude> encode ["foo","foo","foo","bar","bar"]
[(3,"foo"),(2,"bar")]
```

16. Defina a função **uniao** para realizar a união de duas listas. A função deve receber duas listas (sem elementos repetidos) e retornar uma nova lista com todos os elementos das listas originais sem repetições.

Exemplo:

```
Prelude> uniao [3, 6, 5, 7] [2, 9, 7, 5, 1]
[3, 6, 5, 7, 2, 9, 1]
```

17. Defina a função **intercala** que receba duas listas e retorne outra lista com os elementos das listas originais intercalados.

Exemplo:

```
Prelude> intercala [1, 2, 3] [4, 5, 6]
[1, 4, 2, 5, 3, 6]
```

18. Defina a função **consoantList** que retorna verdadeiro se somente se todas as consoantes da segunda lista, incluindo repetições, ocorrem na primeira lista, na mesma ordem.

Exemplos:

```
Prelude> consoantList "vsc" "vasco"
True
Prelude> consoantList "sdd" "saude"
False
Prelude> consoantList "sdd" "saudade"
True
```

19. Defina a função **matches** que recebe uma lista de palavras e uma sequência de consoantes e retorna uma lista de possíveis palavras representadas pelas consoantes.

Exemplos:

Suponha o seguinte dicionário:

```
Prelude> type Dicionario = [[Char]]
Prelude> dic = ["arara","arreio","vaca","vicio"]::Dicionario
Prelude> matches dic "rr"
["arara","arreio"]
Prelude> matches dic "vc"
["vaca","vicio"]
```