



**Disciplina:** Programação 3 - Programação Funcional

**Professor:** Emanuel Barreiros

**Assunto:** Funções recursivas.

**Resumo:** Utilize a linguagem de programação Haskell para resolver os problemas desta lista. Esta lista aborda o conceito de funções recursivas. O conteúdo teórico relacionado a esta lista de exercícios pode ser encontrado aqui:

<https://emanoelbarreiros.github.io/funcional/haskell-6>

- 1) Como a versão recursiva da função fatorial se comporta se dermos a ela como argumento um número negativo? Modifique a implementação clássica para não permitir números negativos adicionando uma guarda ao passo recursivo.
- 2) Defina a função recursiva `somar :: Int -> Int` que retorna a soma dos inteiros não-negativos a partir de um valor até zero. Por exemplo, `somar 3` deve retornar  $3+2+1+0 = 6$ .
- 3) Defina o operador de exponenciação `^` utilizando uma função recursiva, semelhante ao padrão usado para implementar a multiplicação com o operador `*`:  

```
(*) :: Num a => a -> a -> a  
m * 0 = 0  
m * n = m + (m * (n - 1))
```
- 4) Defina a função `euclides :: Int -> Int -> Int` que implementa o algoritmo de Euclides para calcular o máximo divisor comum de dois inteiros não-negativos: se dois números são iguais, este número é o resultado; caso contrário, o menor número é subtraído do maior e o processo é repetido passando este novo número e o menor valor passado anteriormente como argumento. Exemplo:  

```
> euclides 6 27  
3
```
- 5) Defina as funções abaixo usando recursão:
  - a) Decidir se todos os valores em uma lista são True:  

```
and :: [Bool] -> Bool
```
  - b) Concatenar uma lista de listas:  

```
concat :: [[a]] -> [a]
```
  - c) Produzir uma lista com n elementos idênticos:  

```
replicate :: Int -> a -> [a]
```
  - d) Selecionar o n-ésimo elemento em uma lista:  

```
(!!) :: [a] -> Int -> a
```
  - e) Decidir se um valor está presente em uma lista:  

```
elem :: Eq a => a -> [a] -> Bool
```



- 6) Definir a função recursiva `merge :: Ord a => [a] -> [a] -> [a]` que une duas listas ordenadas em uma lista ordenada. Exemplo:
- ```
> merge [2,5,6] [1,3,4]
[1,2,3,4,5,6]
```
- 7) Usando a função `merge`, defina a função `mergesort :: Ord a => [a] -> [a]` que implementa o algoritmo de ordenação *merge sort*, que por sua vez considera uma lista vazia e uma lista com apenas um elemento como listas ordenadas, e que qualquer outra lista é ordenada a partir da união de duas listas que resultaram da ordenação das suas metades separadamente. Dica: primeiro implemente a função `metades :: [a] -> ([a],[a])` que separa uma lista em duas partes cujos comprimentos são iguais ou no máximo diferem em uma unidade.
- 8) Implemente recursivamente funções que:
- calcule a soma de uma lista de inteiros;
  - obtenha o número de elementos de uma lista;
  - selecione o último elemento de uma lista não-vazia.