IF686 - Paradigmas de Linguagens Computacionais
Segunda lista de exercícios

1. Implemente a função mapMaybe que se comporta como map e filter . Os argumentos são uma lista ([a]) e uma função de tipo a -> Maybe b. A função é aplicada a todos os valores da lista. Caso retorne Just x, x estará na lista resultante; se a função retornar Nothing, nada será adicionado à lista resultante.

   Exemplos

   ```
   let f x = if x>0 then Just (2*x) else Nothing
           in mapMaybe f [0,1,-1,4,-2,2]                    ==> [2,8,4]

   mapMaybe Just [1,2,3]   ==> [1,2,3]

   mapMaybe (\x -> Nothing) [1,2,3]   ==> []
   ```

2. Defina a função classifica que recebe uma lista de valores Either a b e retorna uma lista de valores Left e uma lista de valores Right

   ```
   classifica  [Left 1, Right True, Left 0, Right False]
   ==> ([1,0],[True,False])
   ```

3. Descobrir como duas listas diferem uma da outra. Se tiverem comprimentos diferentes, devolva

   Just "<comprimento da lista> /= <comprimento de outra lista>"

   Se tiverem o mesmo comprimento, encontrar o primeiro índice i para o qual os os elementos diferem, e o retorno será

   Just "<valor no indice i> /= < outro valor no indice i>" ,

   Se as listas forem as mesmas, devolver Nothing

   Escreva a assinatura de tipo para findDifference. Que classes de tipo são necessárias?

   Exemplos

   findDifference [True,False] [True,True] ==> Just "False /= True"

   findDifference [0,0,0] [0,0,0,0] ==> Just "3 /= 4"

4. Este é um tipo para um vetor 3D. Implemente uma instância de Eq para ele.

   ```
   data Vetor = Vetor Integer Integer Integer
                      deriving Show
   ```

5. Implemente uma instância de Num para Vetor de modo que as operações aritméticas funcionem componente a componente. A descrição sobre a classe Num pode ser encontrada em

   `https://hackage.haskell.org/package/hspec-2.7.4/docs/Test-Hspec-Discover.html#t:Num`

   Obsevem que funções estão na definição completa minimal

   Exemplos

```
Vetor 1 2 3 + Vetor 0 1 1 ==> Vetor 1 3 4
Vetor 1 2 3 * Vecor 0 1 2 ==> Vetor 0 2 6
abs (Vetor (-1) 2 (-3))     ==> Vetor 1 2 3
signum (Vetor (-1) 2 (-3)) ==> Vetor (-1) 1 (-1)
```