Observação: Todas as questões desta avaliação referem-se à linguagem de programação funcional Haskell

Aluno(a): _____

1. Definir a função **parImpar** e sua assinatura que recebe uma lista de inteiros e retorna duas listas, sendo a primeira uma lista com os números pares e a segunda com os ímpares.

```
parImpar :: [Int]->([Int]),[Int])
parImpar xs = (listaPar xs,listaImpar xs)

listaPar :: [Int]->[Int]
listaPar [] = []
 listaPar (x:xs)
      | even x = x : listaPar xs
      | otherwise = listaPar xs

 listaImpar :: [Int]->[Int]
 listaImpar [] = []
 listaImpar (x:xs)
 | odd x = x : listaImpar xs
 | otherwise = listaImpar xs

 -- Usando função filter do Prelúdio
 parImpar :: [Int]->([Int]),[Int])
 parImpar xs = (filter even xs,filter odd xs)
```

2. 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
   ```

Sim. A primeira função aceita lista de qualquer tipo e sua assinatura é igual a da função hd3, enquanto a função hd2 aceita somente lista de inteiros.

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

   (a) O primeiro elemento de um par

   ```
   prim :: (a,b)->a
   prim (x:_) = x
   ```

   (b) Um dado par com a ordem dos elementos trocados

   ```
   trocado :: (a,b)->(b,a)
   trocado (x:y) = (y,x)
   ```

   (c) O primeiro elemento de um triplo

   ```
   triplo :: (a,b,c)->a
   triplo  (x:_:_) = x
   ```

(d) Um dado triplo com os dois primeiros elementos trocados

```
trocado :: (a,b)->(b,a)
trocado (x:y) = (y,x)
```

(e) O segundo elemento de uma lista

```
segundo :: (a,b)->b
trocado (_:y) = y
```

(f) O segundo elemento do primeiro par de uma lista de pares

```
listaPares :: [(a,b)]->(a,b)
listaPares (x:xs) = b
   where
      (a,b) = x
```

4. Qual o tipo mais geral da assinatura das seguintes funções?

(a) `second xs = head (tail xs)`

```
second :: [a]->a
```

(b) `swap (x,y) = (y,x)`

```
swap :: (a,b)->(b,a)
```

(c) `pair x y = (x,y)`

```
pair :: a->b->(a,b)
```

(d) `double x = 2 * x`

```
double :: Num a => a->a
```

(e) `palin xs = reverse xs == xs`

```
palin :: [a]->[a]
```

(f) `twice f x = f (f x)`

```
twice :: (a->a)->a->a
```

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

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

```
retornaListaSup :: Num a => a->[a]->[a]
retornaListaSup n (x:xs)
            | x > n = x : retornaListaSup n xs
            | otherwise = retornaListaSup n xs
```

6. Refaça a Questão 5 utilizando apenas a função *filter* do módulo **Prelude**.

```
retornaListaSup :: Ord a => a->[a]->[a]
retornaListaSup n xs = filter (>n) xs
```

7. 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]
```

```
insereEmOrdem :: Ord a => a->[a]->[a]
insereEmOrdem n (x:xs)
        | n <= x = x:n:xs
        | otherwise = x:insereEmOrdem n xs
```

8. Refaça a Questão 7 com a função retornando também a lista original.

```
insereEmOrdem :: Ord a => a->[a]->([a],[a])
insereEmOrdem n xs = (xs, insereEmOrdem2 n xs)

insereEmOrdem2 :: Ord a => a->[a]->[a]
insereEmOrdem2 n (x:xs)
        | n <= x = x:n:xs
        | otherwise = x:insereEmOrdem2 n xs
```