Programação Funcional
Trabalho 1
Data de entrega: 09/10/2019

1) Escreva a função `triangulo` que, dados três comprimentos de lados, verifique se podem formar um triângulo e retorne seu tipo. Os retornos podem ser:

- equilátero: se todos os lados são iguais
- isósceles: se dois lados são iguais
- escaleno: se todos os lados diferentes
- não é triângulo: se um dos lados for maior ou igual que a soma dos outros dois

2) Escreva a função `equacao` que, dados três valores reais a, b, c, retorne uma tupla com as duas raízes da equação de segundo grau $ax^2 + bx + c = 0$.

3) Considere que o preço de uma passagem de avião em um trecho pode variar dependendo da idade do passageiro. Pessoas com 60 anos ou mais pagam apenas 60% do preço total. Crianças até 10 anos pagam 50% e bebês (abaixo de 2 anos) pagam apenas 10%. Faça uma função que tenha como entrada o valor total da passagem e a idade do passageiro e produz o valor a ser pago.

4) Apresente as listas geradas pelas expressões:

```
a) [n*n | n <- [1..10], even n]
b) [7 |n <- [1..4]]
c) [ (x,y) | x<-[1..3], y<-[4..7]]
d) [ (m,n) | m<-[1..3], n<-[1..m]]
e) [j | i <- [1,-1,2,-2], i>0, j <- [1..i]]
f) [a+b | (a,b) <- [(1,2),(3,4),(5,6)]]
```

5) Escreva uma função (usando compreensão de listas) que calcula a quantidade de números negativos de uma lista de números:

```
> listaNeg [1,-3,-4,3,4,-5]
3
```

6) Seja a função abaixo que recebe uma lista de pontos no plano cartesiano e calcula a distância de cada ponto à origem:

```
distancias :: [(Float,Float)] -> [Float]
distancias [] = []
distancias ((x,y):xys) = (sqrt (x^2 + y^2)) : (distancias xys)
```

Escreva uma outra versão da função `distancias` utilizando a construção de listas por compreensão.

7) Escreva a função `primos` a seguir que recebe dois valores inteiros x,y e retorna todos os números primos que se encontram entre x e y.

```
> primos 10 50
[11,13,17,19,23,29,31,37,41,43,47]
```

8) Escreva a função `notasTroco` a seguir usando compreensão de listas que calcula todas as combinações de notas para devolver o troco durante um pagamento, a partir de uma lista com os valores das notas disponíveis (definido no arquivo .hs) e o valor do troco x (argumento da função). Ex:
Considere `disponiveis = [1,2,5,10,20,50,100]`

```
> notasTroco 4
[[1,1,1,1],[1,1,2],[1,2,1],[2,1,1],[2,2]]
```

```
> notasTroco 7
[[1,1,1,1,1,1,1],[1,1,1,1,1,2],[1,1,1,1,2,1],[1,1,1,2,1,1],[1,1,1,2
,2],[1,1,2,1,1,1],[1,1,2,1,2],[1,1,2,2,1],[1,1,5],[1,2,1,1,1,1],[1,
2,1,1,2],[1,2,1,2,1],[1,2,2,1,1],[1,2,2,2],[1,5,1],[2,1,1,1,1,1],[2
,1,1,1,2],[2,1,1,2,1],[2,1,2,1,1],[2,1,2,2],[2,2,1,1,1],[2,2,1,2],[
2,2,2,1],[2,5],[5,1,1],[5,2]]
```

9) Construa a função `mmc` a seguir que calcula o valor do mínimo múltiplo comum de três números inteiros.

```
> mmc 2 3 4
12
```

10) Escreva uma função que calcula a série a seguir, dados um número real x e o número de termos a serem calculados n. Obs: se preciso, use a função `fromIntegral` para converter n de Inteiro para Float.

$$Y = \frac{1}{x} + \frac{x}{2} + \frac{3}{x} + \frac{x}{4} + \ldots$$

11) Escreva a função `fizzbuzz` a seguir que recebe um inteiro n e retorna uma lista de strings. Para cada inteiro i entre 1 e n, a lista será composta da seguinte forma.

- Se i é divisível por 3, diga Fizz.
- Se i é divisível por 5, diga Buzz.
- Se i é divisível por ambos 3 e 5, diga FizzBuzz.
- Caso contrário, diga i (use a função `show` para converter i para String).

Exemplo:

```
> fizzbuzz 20
["1","2","Fizz","4","Buzz","Fizz","7","8","Fizz","Buzz","11","Fizz"
,"13","14","FizzBuzz","16","17","Fizz","19","Buzz"]
```

12) Escreva a função `conta_ocorrencias` que recebe um elemento e uma lista qualquer e retorna o número de ocorrências do elemento na lista.

13) Escreva a função `unica_ocorrencia` a seguir que recebe um elemento e uma lista e verifica se existe uma única ocorrência do elemento na lista .

```
> unica_ocorrencia 2 [1,2,3,2]
False

> unica_ocorrencia 2 [3,1]
False

> unica_ocorrencia 2 [2]
True
```

14) Crie uma função que intercala os elementos de duas listas, de qualquer tamanho, contendo números inteiros, numa nova lista.

```
> intercala [1,2,3,4] [100,200]
[1,100,2,200,3,4]
```

15) Defina novos tipos para representar os dados contidos numa agenda telefônica pessoal. Para cada contato, armazene as informações: nome, endereço, telefone, e-mail. Em seguida, crie uma função para recuperar o nome de um contato, a partir de um número de telefone. Caso o número não seja encontrado, retornar a mensagem "Telefone desconhecido".

16) Seja o tipo Pessoa e a lista de pessoas a seguir.
O tipo pessoa é uma tupla que inclui nome, idade, altura e sexo.

```
type Pessoa = (String, Int, Float, Char)
pessoas :: [Pessoa]
pessoas = [ ("Rosa", 27, 1.66,'F'),
     ("João", 26, 1.85,'M'),
     ("Maria", 67, 1.55,'F'),
     ("Jose", 48, 1.78, 'M'),
     ("Paulo", 24, 1.93, 'M'),
     ("Clara", 38, 1.70,'F'),
     ("Bob", 12, 1.45, 'M'),
     ("Rosana", 31, 1.58,'F'),
     ("Daniel", 75, 1.74, 'M'),
     ("Jocileide", 21, 1.69,'F') ]
```

Escreva funções que, dada a lista pessoas, retornem:

- A altura média entre todas as pessoas.
- A idade da pessoa mais nova.
- O nome da pessoa mais velha.
- Os dados de cada pessoa com 60 anos ou mais.
- O número de pessoas do sexo masculino com idade superior a 18 anos.

17) Escreva a função `insere_ord a seguir`, que recebe uma lista polimórfica ordenada de elementos (critério de ordenação crescente) e um novo elemento *x* (do mesmo tipo da lista) e retorna a nova lista com o novo elemento inserido

```
> insere_ord 5 [1,4,7,11]
[1,4,5,7,11]
```

```
> insere_ord 'g' "abcjkl"
"abcgjkl"
```

18) Escreva a função `reverte` a seguir que recebe uma lista polimórfica e retorna uma lista com seus elementos ao contrário.

```
> reverte [1,2,3,4]
[4,3,2,1]

> reverte "abcd"
"dcba"
```

19) Escreva a função `sem_repetidos` a seguir que recebe uma lista polimórfica e retorna uma lista sem elementos repetidos.

```
> sem_repetidos [3,3,2,9,1,7,2,5,9,7]
[3,2,9,1,7,5]

> sem_repetidos "mississippi"
"misp"
```

20) Mostre o resultado obtido pela execução das expressões Haskell:

```
a) (\x -> x + 3) 5
b) (\x -> \y -> x * y + 5) 3 4
c) (\(x,y) -> x * y^2) (3,4)
d) (\(x,y,_) -> x * y^2) (3,4,2)
```

21) Escreva a função conta_maior5 que recebe uma lista de inteiros e retorna uma lista com os elementos da lista original maiores que 5 e quantos elementos são (maiores que 5):

> conta_maior5 [2,6,1,9,7,3]

([6,9,7], 3)

> conta_maior5 []

([], 0)

Obs: utilizar o comando **where** nessa solução.

22) Escreva a função busca_multN que recebe uma lista de inteiros positivos e um inteiro N e busca o primeiro elemento na lista que é múltiplo de N, retornando o elemento e o número de comparações feitas até encontrá-lo. Dica: resolva primeiro para um caso fixo (ex: busca_mult4).

> busca_multN [] 2

(-1, 0) -- não encontrado

> busca_multN [7,2,5,6,3,1] 3

(6, 4) -- encontrado o elemento 6 e foram realizadas 4 comparacoes

Obs: utilizar o comando **where** nessa solução.

23) Reescrever os exercícios 21 e 22 utilizando o comando **let**.