Lista de Exercícios - Paradigma Funcional - Haskell

1. Escrever uma função que encontre as raízes reais de uma equação do 2º grau (ax2+bx + c = 0). Sugestão de assinatura de função: `raizes :: Float -> Float -> Float -> (Float, Float)`

2. Fornecidos três valores diferentes entre si, a, b e c, elaborar uma função que retorne quantos desses três números são maiores que o valor médio entre eles. Sugestão de assinatura da função: `maioresMedia :: Float -> Float -> Float -> Int`

3. Calcular a soma dos números inteiros compreendidos em um intervalo [x,y], incluindo e excluindo os limites. Sugestão de assinatura da função: `somaIntervalo :: Int -> Int -> (Int, Int)`, onde é produzido uma tupla com o primeiro valor refere-se à soma incluindo os limites e o segundo excluindo os limites. Você pode usar funções auxiliares se necessário.

4. Construa uma função que retorne o MMC (Mínimo Múltiplo Comum) entre dois números inteiros. Sugestão de assinatura da função: `mmc :: Int -> Int -> Int`.

5. Construa uma função que retorne o MMC (Mínimo Múltiplo Comum) de uma lista de números inteiros. Sugestão de assinatura da função: `mmc :: [Int] -> Int`.

6. Construa uma função que retorne o Máximo Divisor Comum entre dois números inteiros. Sugestão de assinatura da função: `mdc :: Int -> Int -> Int`.

7. Construa uma função que retorne o Máximo Divisor Comum de uma lista de números inteiros. Sugestão de assinatura da função: `mdc :: [Int] -> Int`.

8. Defina uma função recursiva que recebe dois inteiros m e n, onde m < n, e retorna o produto de todos os números no intervalo [m,n]. Sugestão de assinatura da função: `produtoIntervalo :: Int -> Int -> Int`.

9. A raiz quadrada inteira de um número inteiro positivo n é o maior número inteiro cujo quadrado é menor ou igual a n. Por exemplo, a raiz quadrada inteira de 15 é 3, e a raiz quadrada inteira de 16 é 4. Defina uma função recursiva para calcular a raiz quadrada inteira. Sugestão de assinatura da função: `raizInteira :: Int -> Int`.

10 Elaborar uma função para concatenar duas listas, sem utilizar o operador de concatenação do Haskell (++). Sugestão de assinatura da função: `concatenaListas :: [a] -> [a] ->[a]`. Se necessário, você pode criar funções auxiliares.

11. Elabore uma função que recebe uma string e remove espaços seguidos: ao encontrar dois ou mais espaços seguidos, deixa apenas um. Sugestão de assinatura da função: `removeEspacos :: String -> String`. Se necessário, você pode criar funções auxiliares.

12. Defina uma função que dada uma lista de números reais A, retorne uma lista B com os elementos presentes em A que são menores do que a média de todos os elementos de A. Sugestão de assinatura da função: `menoresQueMedia :: (Num a) => [a] -> [a]`. Se necessário, você pode criar funções auxiliares.

13. Apresentada uma base de dados de 10 alunos matriculados em uma disciplina, construa funções que retornem: (a) número de reprovados na disciplina. Considere como aprovado o aluno que obteve uma nota acima ou igual a 6.0; (b) o nome do(a) aluno(a) que obteve a menor nota.

```
type Nome = String
type Curso = String
type Periodo = Int
type Nota = Float
type Aluno = (Nome, Curso, Periodo, Nota)
type Disciplina = [Aluno]

alunos :: Int -> Aluno
alunos matricula  | matricula == 1 = ("Rodrigo", "S.Inf.", 3, 6.0)
                  | matricula == 2 = ("Joao", "Eng.Comp.", 5, 5.0)
                  | matricula == 3 = ("Lucas", "C.Comp.", 8, 3.5)
                  | matricula == 4 = ("Ana", "C.Comp.", 5, 8.0)
                  | matricula == 5 = ("Maria", "C.Comp.", 7, 9.5)
                  | matricula == 6 = ("Paulo", "C.Comp", 6, 6.0)
                  | matricula == 7 = ("Jose", "S.Inf.", 8, 7.0)
                  | matricula == 8 = ("Eduarda", "C.Comp.", 4, 1.0)
                  | matricula == 9 = ("Carla", "Eng.Comp.", 6, 6.5)
                  | matricula == 10 = ("Luiz", "C.Comp.", 7, 5.7)
```

Exemplo de uso:

```
Main> contar_reprovados 10
4
Main> menor_nota 10
"Eduarda"
```

14.Crie uma função zip3 com funcionamento similar à função zip, mas recebendo três listas ao invés de apenas duas. Crie uma função zip4 com funcionamento similar à função zip, mas recebendo quatro listas ao invés de apenas duas.

19. Construa a função de Ackermann, a qual é definida por:

- A(m, n) = n + 1 se m = 0
- A(m, n) = A(m-1, 1) se m ≠ 0 e n = 0
- A(m, n) = A(m-1, A(m, n-1)) se m ≠ 0 e n ≠ 0

15. Sejam as duas funções abaixo que verificam se um dado número é par. Teste cada função e explique a estratégia utilizada na implementação de cada uma[1].

par x = (mod x 2) == 0

par1 x = if (x == 0) then True
else not (par1 (x-1))

16. Faça uma função que, dado um ano, verifica se o mesmo é bissexto[2].

17. Seja o cadastro de pessoas dado pela função a seguir[3]:

```
type Pessoa = (String, Int, Float,Char)

pess :: Int->Pessoa
pess x
      |x==1 = ("Rosa", 27, 1.66,'F')
      |x==2 = ("João", 26, 1.85,'M')
      |x==3 = ("Maria", 67, 1.55,'F')
      |x==4 = ("Jose", 48, 1.78, 'M')
      |x==5 = ("Paulo", 24, 1.93, 'M')
      |x==6 = ("Clara", 38, 1.70,'F')
      |x==7 = ("Bob", 12, 1.45, 'M')
      |x==8 = ("Rosana", 31, 1.58,'F')
      |x==9 = ("Daniel", 75, 1.74, 'M')
      |x==10 = ("Jocileide", 21, 1.69,'F')
      |otherwise = ("Acabou!",0, 0.0, 'x')
```

Construa funções que retornem os seguintes dados:

- O número do registro da pessoa de maior idade.
- A idade média de todas das pessoas.
- O número de pessoas do sexo masculino com idade superior a 25 anos.

18. Faça uma função que calcule a soma de todos os números ímpares de 1 à N, utilizando recursão de cauda[4].

---

[1] Retirado de lista de exercícios da profa. Michele Nasu Tomiyama, UFU.
[2] Retirado de lista de exercícios da profa. Michele Nasu Tomiyama, UFU.
[3] Retirado de lista de exercícios da profa. Michele Nasu Tomiyama, UFU.
[4] Retirado de lista de exercícios da profa. Michele Nasu Tomiyama, UFU.