Resoluções das Questões do Moodle
Programação Funcional

1- Escreva uma função soma que recebe dois parâmetros e devolve a soma dos dois parâmetros.

soma x y = x + y

2- Defina a função interior tal que (interior xs) é uma lista obtida eliminando os extremos da lista xs.
Por exemplo: *interior [2,5,3,7,3] == [5,3,7]*

interior xs = tail (init xs)

OBS.: tail : ele aceita uma lista e retorna a lista sem seu primeiro item.
init: eles aceita uma lista e retorna a lista sem o último item.

3- Dado três valores a, b e c, escreva uma função iguais3 que retorne quantos dos três são iguais. A resposta pode ser 3 (todos iguais), 2 (dois iguais) ou 0 (todos diferentes).

```
iguais3 a b c  | a == b && a == c  && c == b = 3
               | a == b | | a ==  c | | c == b = 2
               | otherwise = 0
```

4- Defina a função max3 tal que (max3 x y z) é o máximo entre x, y e z.
Por exemplo: *max3 6 2 4 == 6 // max3 6 7 4 == 7 // max3 6 7 9 == 9*

max3 x y z = max(max x y) z

OBS.: max: retorna o maior de seus dois argumentos.

5- Defina uma função somaImpares tal que (somaImpares xs) devolve a soma dos elementos ímpares de uma lista.
Por exemplo: *somaImpares [2,3,1,5] == 9 // somaImpares [1,1,4,2] == 2*
Dica: *a função odd, filter.*

somaImpares xs = sum (filter odd (xs))

OBS.: sum: calcula uma soma de todos os elementos na lista.
filter: retorna uma lista construída a partir de membros de uma lista (o segundo argumento) cumprindo uma condição fornecida pelo primeiro argumento.
Odd: retorna True se a integral for ímpar, caso contrário, False.

6- Defina a função neglist xs que computa o número de elementos negativos em uma lista xs.
Por exemplo: *neglist [1 , 2 , 3 , 4 , 5] == 0 // neglist [1 , −3 , −4 ,3 ,4 , −5] == 3*

Dica: *usa a função filter e length.*

neglist xs = length (filter (< 0) xs)

OBS.: lenght: retorna o número de itens em uma lista.
filter: retorna uma lista construída a partir de membros de uma lista (o segundo argumento) cumprindo uma condição fornecida pelo primeiro argumento.

7- Defina a função final tal que (final xs) é uma lista formada pelos n elementos finais de xs.

Por exemplo: *final 3 [2,5,4,7,9,6] == [7,9,6] // final 2 [2,5,4,7,9,6] == [9,6] // final 1 [2,5,4,7,9,6] == [6]*

final n xs = if (n >= length xs) then xs else (drop n xs)

OBS.: lenght: retorna o número de itens em uma lista.
Drop: Exclua os primeiros N elementos de uma lista.

8- Joãozinho acaba de mudar de escola e a primeira coisa que percebeu na nova escola é que a gangorra do parquinho não é simétrica, uma das extremidades é mais longa que a outra. Após brincar algumas vezes com um amigo de mesmo peso, ele percebeu que quando está em uma extremidade, a gangorra se desequilibra para o lado dele (ou seja, ele fica na parte de baixo, e o amigo na parte de cima), mas quando eles trocam de lado, a gangorra se desequilibra para o lado do amigo.

Sem entender a situação, Joãozinho pediu ajuda a outro amigo de outra série, que explicou que o comprimento do lado interfere no equilíbrio da gangorra, pois a gangorra estará equilibrada quando

$$P_1 \times C_1 = P_2 \times C_2$$

onde P_1 e P_2 são os pesos da criança no lado esquerdo e direito, respectivamente, e C_1 e C_2 são os comprimentos da gangorra do lado esquerdo e direito, respectivamente.

Escreva uma função gangorra que recebe quatro inteiros P_1, C_1, P_2 e C_2, que são respectivamente, o peso da criança e o comprimento da gangorra do lado esquerdo e o o peso da criança e o comprimento da gangorra do lado direito. Se a gangorra estiver equilibrada, a função gangorra devolve 0. Se ela estiver desequilibrada de modo que a criança esquerda esteja na parte de baixo, a função gangorra devolve -1, senão, devolve 1.

Por exemplo: *gangorra 30 100 60 50 == 0 // gangorra 40 40 38 60 == 1*

gangorra p1 c1 p2 c2 | (p1 * c1) == (p2 * c2) = 0
| (p1 * c1) > (p2 * c2) = -1
| (p1 * c1) < (p2 * c2) = 1

9- Defina uma função segmento tal que (segmento n m xs) é uma lista dos elementos de xs compreendidos entre as posições m e n.

Por exemplo: *segmento 3 4 [3,4,1,2,7,9,0] == [1,2] // segmento 1 2 [3,4,1,2,7,9,0] == [3,4] // segmento 1 3 [3,4,1,2,7,9,0] == [3,4,1] // segmento 1 1 [3,4,1,2,7,9,0] == [3] // segmento 1 4 [3,4,1,2,7,9,0] == [3,4,1,2] // segmento 1 5 [3,4,1,2,7,9,0] == [3,4,1,2,7]*

Dica: *use a função drop e take*

segmento n m xs = drop (n-1) (take m xs)

OBS.: drop: Exclua os primeiros N elementos de uma lista.
take: cria uma lista, o primeiro argumento determina, quantos itens devem ser retirados da lista passada como o segundo argumento.

10- Defina a função somaQuadrados que recebe um inteiro n como argumento e devolve a soma dos quadrados dos primeiros n inteiros, ou seja,

somaQuadrados n == 1^2 + 2^2 + .. + n^2

Por exemplo: *somaQuadrados 1 == 1 // somaQuadrados 2 == 5 // somaQuadrados 3 == 14 // somaQuadrados 150 == 1136275*

somaQuadrados n = sum (map(\x → x*x) [1..n])

OBS.: sum: calcula uma soma de todos os elementos na lista.
map: retorna uma lista construída aplicando uma função (o primeiro argumento) a todos os itens em uma lista passada como o segundo argumento.

11- Defina a função fact tal que (fact n) calcula o fatorial de n, ou seja, fact n = 1*2*...*n

Por exemplo: *fact 1 == 1 // fact 2 == 2 // fact 4 == 24 // fact 5 ==120*

Dica: *use a função product*

fact n = product [1..n]

OBS.: product: calcula um produto de todos os elementos na lista.

12- Escreva uma função divisores tal que (divisores n) devolve uma lista dos número que são divisores de n.

Por exemplo: *divisores 15 == [1,3,5,15] // divisores 74 == [1,2,37,74]*

Dica: *Utilize a função filter sobre a lista [1..n]*

divisores n = filter (\x → mod n x == 0) [1..n]

OBS.: filter: retorna uma lista construída a partir de membros de uma lista (o segundo argumento) cumprindo uma condição fornecida pelo primeiro argumento.
mod: resto da divisão.

13- Todos devem conhecer o jogo Zerinho ou Um (em algumas regiões também conhecido como Dois ou Um), utilizado para determinar um ganhador entre três ou mais jogadores. Para quem não conhece, o jogo funciona da seguinte maneira. Cada jogador escolhe um valor entre zero ou um; a um comando (geralmente um dos competidores anuncia em voz alta ?Zerinho ou... Um!?), todos os participantes mostram o valor escolhido, utilizando uma das mãos: se o valor escolhido foi um, o competidor mostra o dedo indicador estendido; se o valor escolhido foi zero, mostra a mão com todos os dedos fechados. O ganhador é aquele que tiver escolhido um valor diferente de todos os outros; se não há um jogador com valor diferente de todos os outros (por exemplo todos os jogadores escolhem zero, ou um grupo de jogadores escolhe zero e outro grupo escolhe um), não há ganhador.

Escreva uma função zeroUm que recebe três inteiros a, b e c, que são, respectivamente, o valor escolhido por Alice, o valor escolhido por Beto e o valor escolhido por Clara, cada valor é zero ou um. Se o vencedor é Alice, a função zeroUm devolve 'A', se o vencedor é Beto, a função zeroUm devolve'B', se o vencedor é Clara, a função zeroUm devolve 'C' e se não há vencedor, a função de '*' (asterisco).

ZeroUm a b c | (a==b && a==c && b==c) = '*'
| (a/=b && b==c) = 'A'
| (b/=a && a==c) = 'B'
| (c/=a && a==b) = 'C'

14- Parte do treinamento de um novo garçom é carregar uma grande bandeja com várias latas de bebidas e copos e entregá-las todas numa mesa do restaurante. Durante o treinamento é comum que os garçons deixem cair as bandejas, quebrando todos os copos.

A Sociedade Brasileira de Copos (SBC) analisou estatísticas do treinamento de diversos garçons e descobriu que os garçons em treinamento deixam cair apenas bandejas que têm mais latas de bebidas que copos. Por exemplo, se uma bandeja tiver 10 latas e 4 copos, certamente o garçom em treinamento a deixará cair, quebrando os 4 copos. Já se a bandeja tiver 5 latas e 6 copos, ele conseguirá entregá-la sem deixar cair.

Escreva uma função coposQuebrados :: [(Int,Int)] -> Int tal que (coposQuebrados xs) devolve o total de copos quebrados por um garçom considerando uma lista de bandejas que o garçom tentou entregar. Cada bandeja é representada por uma tupla (L,C), onde L é o número de latas e C é o número de copos.

Por exemplo: *coposQuebrados [(10,5), (6,8),(3,3)] == 5*

Observe que, apenas na primeira bandeja, o número de latas é maior que copos.

CoposQuebrados xs = sum(map(\lata,copo) → if(lata > copo) then copo else 0) xs)

OBS.: sum: calcula uma soma de todos os elementos na lista.
map: retorna uma lista construída aplicando uma função (o primeiro argumento) a todos os itens em uma lista passada como o segundo argumento.

15- Muitas crianças gostam de decidir todas as disputas através do famoso jogo de Par ou Ímpar. Nesse jogo, um dos participantes escolhe Par e o outro Ímpar. Após a escolha, os dois jogadores mostram, simultaneamente, uma certa quantidade de dedos de uma das mãos. Se a soma dos dedos das mãos dos dois jogadores for par, vence o jogador que escolheu Par inicialmente, caso contrário vence o que escolheu Ímpar.

Escreva uma função (parImpar :: [(Int,Int)] -> Int) tal que (parImpar xs) devolve o número de vitórias do primeiro jogador. Cada partida de Par ou Ímpar é descrita por dois inteiros A e B que representam o número de dedos do primeiro jogador e o número de dedos do segundo jogador, respectivamente. Considere que o primeiro jogador sempre escolhe Par.

Por exemplo: *parImpar [(2,4),(3,5),(1,0)] == 2 // parImpar [(1,5),(2,1),(1,4),(2,2)] == 2 // parImpar [(1,5),(2,3)] == 1*

parImpar xs = sum(map(\ (a,b) → if(mod(a+b) 2==0) then 1 else 0)xs)

OBS.: sum: calcula uma soma de todos os elementos na lista.
map: retorna uma lista construída aplicando uma função (o primeiro argumento) a todos os itens em uma lista passada como o segundo argumento.

16- Dois times, Cormengo e Flaminthians, participam de um campeonato de futebol, juntamente com outros times. Cada vitória conta três pontos, cada empate um ponto. Fica melhor classificado no campeonato um time que tenha mais pontos. Em caso de empate no número de pontos, fica melhor classificado o time que tiver maior saldo de gols. Se o número de pontos e o saldo de gols forem os mesmos para os dois times então os dois times estão empatados no campeonato.

Escreva uma função campeonato que recebe seis inteiros cv , ce , cs , fv , fe e fs , que são, respectivamente, o número de vitórias do Cormengo, o número de empates do Cormengo, o saldo de gols do Cormengo, o número de vitórias do Flaminthians, o número de empates do Flaminthians e o saldo de gols do Flaminthians. Se Cormengo é melhor classificado que Flaminthians, a funcao classifica devolve a letra 'C', se Flaminthians é melhor classificado que Cormengo, a funcao classificacao devolve a letra 'F', e se os dois times estão empatados a funcao classificacao devolve apenas o caractere '='.

Por exemplo: *campeonato 10 5 18 11 1 18 == 'C' // campeonato 10 5 18 11 2 18 == '=' // campeonato 9 5 -1 10 2 10 == 'F'*

campeonato cv ce cs fv fe fs

|(cv * 3) + ce > (fv * 3) + fe = 'C'
|(cv * 3) + ce < (fv * 3) + fe = 'F'
|(cs > fs) = 'C'
|(fs > cs) = 'F'
|otherwise = '='

17- O valor π pode ser calculado através da seguinte soma de uma sequência infinita:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} + \ldots + \frac{4(-1)^n}{2n-1} + \ldots$$

Usando a função foldl defina a função calcPi :: (Integral a, Fractional b) => a -> b tal que (aproximae n) pode ser calculado da seguinte maneira:

calcPi n = $\sum_{i=0}^{n} \frac{4(-1)^i}{2i+1}$

Por exemplo: *calcPi 10 == 3.23231580940559 // calcPi 100 == 3.15149340107099 // calcPi 1000 == 3.14259165433954*

calcPi n = sum[(4*(-1)^(x)) / (fromIntegral (2*x+1)) | x ← [0..n]]

OBS.: sum: calcula uma soma de todos os elementos na lista.
fromIntegral: converte qualquer tipo inteiro para qualquer outro tipo numérico.

18- O número *e* pode ser calculado como a soma da seguinte série infinita:

$$e = \sum_{n=0}^{\infty} \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

Usando a função foldl defina a função aproximae :: (Integral a, Fractional b) => a -> b tal que (aproximae n) pode ser calculado da seguinte maneira:

aproximae n = $\sum_{i=0}^{i=n} \frac{1}{i!}$

Por exemplo: *aproximae 0 ==1.0 // aproximae 1 == 2.0 // aproximae 2 == 2.5 // aproximae 3 == 2.66666666666667 // aproximae 4 == 2.70833333333333 // aproximae 5 == 2.71666666666667 // aproximae 6 == 2.71805555555556 // aproximae 7 == 2.71825396825397 // aproximae 8 == 2.71827876984127*

Dica 1: Use a função map

Dica 2: Utilize a definição da função fact n = n!

Dica 3: Utilize a função fromIntegral para realizar a divisão fracionária.

fatorial n = product [1..n]
aproximae n = sum[(1 / fromIntegral (fatorial x)) | x ← [0..n]]

OBS.: product: calcula um produto de todos os elementos na lista.
sum: calcula uma soma de todos os elementos na lista.
fromIntegral: converte qualquer tipo inteiro para qualquer outro tipo numérico.

19- As cadeias de DNA e RNA são uma sequência de nucleotídeos. Os quatro nucleotídeos encontrados no DNA são adenina (A), citosina (C), guanina (G) e timina (T). Os quatro nucleotídeos encontrados no RNA

são adenina (A), citosina (C), guanina (G) e uracila (U). Dada uma sequência de DNA, sua sequência de RNA transcrita é formada substituindo cada nucleotídeo por seu complemento:

G -> C

C -> G

T -> A

A -> U

Defina uma função toRNA :: String -> String tal que (toRNA xs) devolve a cadeia de RNA formada a partir da cadeia-molde de DNA, xs.

Por exemplo: *toRNA "ACGTGGTCTTAA"== "UGCACCAGAAUU"*

```
rna a
    | a == 'G' = 'C'
    | a == 'C' = 'G'
    | a == 'T' = 'A'
    | a == 'A' = 'U'
toRNA xs = [rna x | x ← xs]
```

COMPREENSÃO DE LISTAS

20- Defina a função capitalises que recebe uma lista de caracteres e devolve a uma lista substituindo as letras minúsculas por maiúsculas.

Por exemplo: capitalises "Minority Report" == "MINORITY REPORT"

Dica 1: *use a função toUpper*

Dica 2: *Não esqueça de colocar import Data.Char na primeira linha do arquivo .hs para utilizar a função toUpper do módulo Data.Char*

```
import Data.Char

capitalises xs = [toUpper x | x ← xs]
```

OBS.: toUpper: converte uma letra na letra maiúscula correspondente, mantendo qualquer outro caractere inalterado. Qualquer letra Unicode que tenha um equivalente em maiúscula é transformada.

21- Números abundantes são números natural que a soma dos seus divisores próprios é maior do que ele mesmo. Por exemplo, os divisores próprios de 12 são 1, 2, 3, 4 e 6. A soma dos divisores é 16. Logo, 12 é um número abundante. Por outro lado, o número 5 não é abundante uma vez que ele possui apenas um divisor próprio que é 1.

Por exemplo: *abundante 12 == True // abundante 5 == False*

```
divisores n = [x | x ← [1..(n-1)], mod n x == 0]
abundante n = sum(divisores n) > n
```

OBS.: mod: resto da divisão.
sum: calcula uma soma de todos os elementos na lista.

22- Escreva uma função todosPrefixos tal que (todosPrefixos xs) devolve uma lista formada com todos os prefixos da lista xs. Um prefixo de uma lista xs é qualquer sequência inicial de elementos da lista xs.

Por exemplo: *todosPrefixos [2,3,5,6,7,8]=[[],[2],[2,3],[2,3,5],[2,3,5,6],[2,3,5,6,7],[2,3,5,6,7,8]] // todosPrefixos [2,3,1,5] = [[],[2],[2,3],[2,3,1],[2,3,1,5]]*

```
todosPrefixos xs = [take i xs | i ← [0..n]]
    where n = length xs
```

OBS.: take: cria uma lista, o primeiro argumento determina, quantos itens devem ser retirados da lista passada como o segundo argumento.
lenght: retorna o número de itens em uma lista.
Where: A cláusula where faz definições locais à equação, ou seja, o escopo dos nomes definidos em uma cláusula where restringe-se à equação contendo a cláusula where.

23- Escreva uma função todosSufixos tal que (todosSufixos xs) devolve uma lista formada com todos os sufixos da lista xs. Um sufixo de uma lista xs é qualquer sequência final de elementos da lista xs.

Por exemplo: *todosSufixos [2,3,5,6,7,8] = [[],[8],[7,8],[6,7,8],[5,6,7,8],[3,5,6,7,8],[2,3,5,6,7,8]] // todosSufixos [2,3,1,5] = [[],[5],[1,5],[3,1,5],[2,3,1,5]] // todosSufixos [2,3,5,6,7,8,5,6,1,2,8,9] = [[],[9],[8,9],[2,8,9],[1,2,8,9],[6,1,2,8,9],[5,6,1,2,8,9],[8,5,6,1,2,8,9],[7,8,5,6,1,2,8,9], [6,7,8,5,6,1,2,8,9],[5,6,7,8,5,6,1,2,8,9],[3,5,6,7,8,5,6,1,2,8,9],[2,3,5,6,7,8,5,6,1,2,8,9]]*

Dica: *use a função final.*

```
todoSufixos xs = [drop i xs | i ← [n, (n-1)..0]]
    where n = lenght xs
```

OBS.: Drop: Exclua os primeiros N elementos de uma lista.
lenght: retorna o número de itens em uma lista.
Where: A cláusula where faz definições locais à equação, ou seja, o escopo dos nomes definidos em uma cláusula where restringe-se à equação contendo a cláusula where.

24- Defina a função interseccao :: Eq a => [a] -> [a] -> [a] tal que (interseccao xs ys)

devolve a intersecçao dos conjuntos xs e ys seguindo a ordem em que eles aparecem na lista xs.

Por exemplo: interseccao [3 , 2 , 5] [5 , 7 , 3 , 4] == [3 , 5] // interseccao [3 , 2 , 6] [5 , 7 , 3 , 4] == [3] // interseccao [8 , 2 , 6] [5 , 7 , 3 , 4] == [ ] // interseccao [3 , 1 , 4, 6] [6, 3, 1, 9] == [3, 1, 6]

intersecao xs ys = [filter (\n → elem n ys)xs]

OBS.: filter: retorna uma lista construída a partir de membros de uma lista (o segundo argumento) cumprindo uma condição fornecida pelo primeiro argumento.
Elem: retorna True se a lista contiver um item igual ao primeiro argumento.

25- Defina a função somaConsecutivos tal que (somaConsecutivos xs) é a soma dos pares de elementos consecutivos de uma lista xs.

Por exemplo: *somaConsecutivos [3,1,5,2] == [4,6,7] // somaConsecutivos [3] == []*

pares xs = zip xs (tail xs)

somaConsecutivos xs = [x + y | (x, y) ← pares xs]

OBS.: zip: faz uma lista de tuplas, cada uma delas contém os elementos de ambas as listas ocorrendo na mesma posição.
tail : ele aceita uma lista e retorna a lista sem seu primeiro item.

26- Defina a função abundantesMenores tal que (abundantesMenores n) devolve uma lista de números abundantes menores ou igual a n.

Por exemplo: *abundantesMenores 50 == [12,18,20,24,30,36,40,42,48] // abundantesMenores 100 == [12,18,20,24,30,36,40,42,48,54,56,60,66,70,72,78,80,84,88,90,96,100]*

divisores n = [x | x ← [1..(n-1)], mod n x == 0]
abundante n = sum(divisores n) > n
abundantesMenores n = [x | x ← [1..n], abundante x]

OBS.: mod: resto da divisão.
sum: calcula uma soma de todos os elementos na lista.

26- Defina a função somaMultiplos :: [Integer] -> Integer -> Integer tal que somaMultiplos x s n devolve a soma de todos os múltiplos únicos de algum número da lista x s estritamente menores que n. Por exemplo, todos os números naturais abaixo de 20, que são múltiplos de 3 ou 5, obtemos 3, 5, 6, 9, 10, 12, 15 e 18. A soma desses múltiplos é 78.

Por exemplo: *somaMultiplos [3,5] 20 == 78*

```
import Data.List
multiplo m n = if (mod m n == 0) then m else 0
somaMultiplos xs n = sum(nub[multiplo m z| m ← [1..(n-1)], z ← xs])
```

OBS.: mod: resto da divisão.
sum: calcula uma soma de todos os elementos na lista.
Nub: remove elementos duplicados de uma lista.

RECURSÃO

27- Defina a função merge :: [a] -> [a] -> [a] tal que (merge xs ys) é uma lista ordenada obtida pela entrelaçamento de duas listas ordenadas xs e ys.

Por exemplo: merge [2,5,6] [1,3,4] == [1,2,3,4,5,6] // merge [3,5,6, 8] [1,3,4,5] == [1,3,3,4,5,5,6,8]

```
merge xs [ ] = xs
merge [ ] ys = ys
merge (x:xs) (y:ys)
    | x <= y = x: merge xs (y:ys)
    |otherwise = y:merge (x:xs) ys
```

28- Defina uma função subconjunto :: Eq a => [a] -> [a] -> Bool tal que (subconjunto xs ys) verifica xs é um subconjunto de ys.

Por exemplo: *subconjunto [3 , 2 , 3] [2 , 5 , 3 , 5] == True // subconjunto [3 , 2 , 3] [2 , 5 , 6 , 5] == F a l s e*

Dica: *use a função elem do Prelude.*

```
Subconjunto [ ] ys = True
subconjunto (x:xs) ys = if (elem x ys) then subconjunto xs ys else False
```

OBS.: elem: retorna True se a lista contiver um item igual ao primeiro argumento.

29- Defina uma função inserir :: Ord a => a -> [a] -> [a] tal que (inserir x xs) recebe um elemento x e uma lista ordenada de maneira crescente devolvendo uma lista ordenada ascendentemente, oriunda da inserção apropriada de x em xs.

Por exemplo: *inserir 3 [2,7,12] == [2,3,7,12]*

```
inserir x [ ] = [x]
inserir x (y:ys) = if (x < y) then x:y:ys else y:inserir x ys
```

30- Defina uma função remove :: Eq a => a -> [a] -> [a] tal que (remove x xs) devolve uma lista obtida removendo uma ocorrências de x em xs, caso ela exista.

Por exemplo: remove 2 [2,3,4,5] == [3,4,5] // remove 2 [1,3,2,4] == [1,3,4]

remove x [ ] = [ ]
remove num (x:xs) = if(num==x) then xs else x: remove num xs

31- Defina a função frequencia :: a -> [a] -> Int tal que (frequencia x xs) devolve o número de ocorrências de x em u.

Por exemplo: frequencia 2 [1,2,3,2,4] == 2 // frequencia 2 [2,1,2,3,2,4,2] == 4

frequencia a [ ] = 0
frequencia a (x:xs) = if (a==x) then (1+(frenquencia a xs)) else frequencia a xs

---

32- Usando a função foldr, defina a função remdups tal que (remdups xs) remove todo elementos duplicados em xs.

Por exemplo: remdups [1,2,1,3,2,3] == [1,2,3] // remdups [2,2,1,3,2,3] == [1,2,3] // remdups [4,3,2,2,1,4,3,2,3] == [1,4,2,3]

remdups xs = foldr (\x z → if (elem x z) then z else x: z) [ ] xs

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.
elem: retorna True se a lista contiver um item igual ao primeiro argumento.

33- Usando apenas funções da Biblioteca Prelude, escreva a função metadePares :: [Integer] -> [Integer] tal que (metadePares xs) devolve a lista dos elementos que são pares de xs divididos por 2.

Por exemplo: *metadePares [0,-1,3,4,-16,3] == [0,2,-8]*

metadePares xs = [div x 2 | x ← xs, mod x 2==0]

OBS.: div: retorna quantas vezes o primeiro número pode ser dividido pelo segundo.
mod: resto da divisão.

34- Usando compreensão de listas, Escreva a função noIntervalo :: Int -> Int -> [Int] -> [Int] tal que (noIntervalo a b xs) retorna todos os valores de xs que são maiores ou iguais a e menores ou iguais a a b.

Por exemplo: *noIntervalo 5 10 [1..15] == [5,6,7,8,9,10] // noIntervalo 2 1 [1,2,3,4] == [] // noIntervalo 1 2 [1,2,2,3,4,1] == [1,2,2,1]*

noIntervalo a b xs = [x | x ← xs, x >= a, x <= b]

35- Defina a função intercala :: a -> [a] -> [a] que intercala um elemento entre valores consecutivos numa lista; se a lista tiver menos de dois valores deve ficar inalterada. Exemplos:

> intercala 0 [1 . . 4]

[1 ,0 ,2 ,0 ,3 ,0 ,4]

> i n t e r c a l a ' , ' ” abcd ”

”a , b , c , d”

> i n t e r c a l a ' , ' ”a”

”a”

> intercala 1 [ ]

[]

intercala x [ ] = [ ]
intercala y (x:xs) = y: x: (intercala y xs)

36- A função insert :: Ord a => a -> [a] -> [a] tal que (insert x xs) devolve uma lista ordenada obtida pela inserção de x na lista ordenada xs.

Por exemplo: insert 2 [0, 1, 3, 5] == [0, 1, 2, 3, 5]

inserir x [ ] = [x]
inserir x (y: ys) = if (x <= y) then x:y:ys else y:(insert x ys)

37- Usando a função insert, escreva a função insertSort :: Ord a => [a] -> [a] tal que (insertSort xs) recebe uma lista possivelmente não ordenada xs e devolve uma lista ordenada implementando ordenação pelo método de inserção:

• Se a lista é vazia, então já está ordenada;

• Se a lista é não vazia (x : xs), então inserimos x na lista ordenada obtida pela ordenação de xs.

insert x [ ] = [x]
insert x (y:ys) = if (x <= y) then x:y:ys else y:(insert x xs)

insertSort xs = foldr insert [ ] xs

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.

38- Defina a função agrupa :: Eq a => [[a]] -> [[a]] tal que (agrupa xss) é uma lista de listas obtidas agrupando os primeiros elementos, os segundos elementos, de forma que o comprimento das listas dos resultados seja igual a lista mais curta de xss.

Por exemplo: *agrupa [[1 . . 6] , [7 . . 9] , [1 0 . . 2 0]] == [[1 , 7 , 1 0] , [2 , 8 , 1 1] , [3 , 9 , 1 2]] // agrupa [ ] == [ ]*

Dica: *use a função map, head e tail.*

```
agrupa :: Eq a => [[a]] -> [[a]]

vazia xss = elem [ ] xss
agrupa xss  | elem [ ] xss
            | otherwise = (map head xss) : agrupa (map tail xss)
```

OBS.: elem: retorna True se a lista contiver um item igual ao primeiro argumento.
map: retorna uma lista construída aplicando uma função (o primeiro argumento) a todos os itens em uma lista passada como o segundo argumento.
head: retorna o primeiro item de uma lista.
tail: ele aceita uma lista e retorna a lista sem seu primeiro item.

39- Escreva uma função filtrandoCaudas :: [[Int]] -> [[Int]] usando compreensão de listas tal

que (caudas xss) devolve uma lista contendo a cauda das listas não vazias.

Por exemplo: *filtrandoCaudas [[2,3,5,6],[2,5,7,9],[4,5,6,7]] == [[3,5,6],[5,7,9],[5,6,7]]*

Dica : *use as funções head, tail, null.*

```
filtrandoCaudas :: [[Int]] -> [[Int]]

FiltrandoCaudas xss = [tail xs | xs ← xss, not(null xs)]
```

OBS.: tail: ele aceita uma lista e retorna a lista sem seu primeiro item.
Null: retorna True se uma ista estiver vazia, caso contrário, False.

40- Defina a função descompacta :: :: [(a, b)] -> ([a], [b]) que transforma uma lista de pares ordenado em um par ordenado onde o primeiro elemento é uma lista dos primeiros componentes dos pares ordenados e o segundo elemento é uma lista dos segundos componentes dos pares ordenados.

Por exemplo: *descompacta [(1 , 2) , (3 , 4) , (5 , 6) , (4 , 5)] == ([1 , 3 , 5 , 4] , [2 , 4 , 6 , 5]) // descompacta [(1 , 2) , (3 , 4) , (5 , 6) , (4 , 5) , (5 , 6)] == ([1 , 3 , 5 , 4 , 5] , [2 , 4 , 6 , 5 , 6])*

```
descompacta :: [(a, b)] → ([a], [b])
descompacta xs = ([a | (a, b) ← xs], [b | (a,b) ← xs])
```

41- Defina uma função filtrandoListas :: [[a]]->[[a]] tal que (filtrandoListas xss) devolve uma lista contendo o maior prefixo do mesmo tamanho de cada lista de xss.

Por exemplo: filtrandoListas [[3,2,1],[3,4],[4,3,2,1]] == [[3,2],[3,4],[4,3]]

```
filtrandoListas :: [[a]]→[[a]]

menor xss = minimum [length xs | xs ← xss]

filtrandoListas xss = [take menorL xs | xs ← xss]
        where menorL = menor xss
```

OBS.: minimum: retorna o valor mínimo da lista.
lenght: retorna o número de itens em uma lista.
take: cria uma lista, o primeiro argumento determina, quantos itens devem ser retirados da lista passada como o segundo argumento.

42- Defina a função temLetraOuDigito :: String -> Bool, usando foldr, que recebe um argumento do tipo String e devolve True, se a string contém algum letra (minúscula ou maiúscula) ou algum dígito, e False, caso contrário.

Dica: *Use as funções isLetter :: Char -> Bool e isDigit:: Char -> Bool importando o módulo Data.Char adicionando a seguinte instrução import Data.Char.*

```
Import Data.Char
temLetraOuDigito :: String → Boll

temLetraOuDigito c = foldr (\x cc → isLetter x | | isDigit x | | cc) False c
```

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.
isLetter: verdadeiro se o caractere for um caractere alfabético.
IsDigit: verdadeiro se o caractere for um caractere numérico decimal (0..9)

43- A função remdups remove elementos duplicados adjacentes de uma lista.

Por exemplo: *remdups [1 , 2 , 2 , 3 , 3 , 3 , 1 , 1] = [1 , 2 , 3 , 1]*

```
remdups [ ] = [ ]
remdups [x] = [x]
remdups (x:y:xs)
        | x == y = remdups (y:xs)
        | otherwise = x: remdups (y:xs)
```

44- Num sorteio que distribui prêmios, um participante inicialmente sorteia uma lista de valores inteiros. O número de pontos do participante é o tamanho da maior sequência de valores consecutivos iguais. Por exemplo, suponhamos que um participante sorteia 11 valores e, nesta ordem, os valores são [30, 30, 30, 30, 40, 40, 40, 40, 40, 30,30]. Então, o participante ganha 5 pontos, correspondentes aos 5 valores 40 consecutivos.

Note que o participante sorteou 6 valores iguais a 30, mas nem todos são consecutivos.

Escreva uma função pontos :: [Int] -> Int tal que (pontos xs) devolve o número de pontos do participante considerando a lista de valores sorteados xs.

Por exemplo: *pontos [30,30,30,40,40,40,40,40,30,30,30] == 5 // pontos [1,1,1,20,20,20,20,3,3,3,3,3,3,3] == 7*

Dica: *use a função takeWhile :: [a] -> (a->Bool)->[a] e dropWhile :: [a] -> (a->Bool)->[a]*

```
pontos [ ] = 0
pontos (x:xs) = if(z > pontos w) then z else pontos w
        where z = length(takeWhile (==x) xs) +1
              w = dropWhile (==x) xs
```

OBS.: takeWhile: cria uma lista a partir de outra, inspeciona a lista original e leva seus elementos para o momento em que a condição falha e depois para de processar.

dropWhile: cria uma lista a partir de outra, inspeciona a lista original e retira seus elementos desde o momento em que a condição falha pela primeira vez até o final da lista**.**

lenght: retorna o número de itens em uma lista.

45- Dado uma lista de números inteiros, definiremos o maior salto como o maior valor das diferenças (em valor absoluto) entre números consecutivos da lista. Por exemplo, dada uma lista [2,5,-3,7]

- 3 (valor absoluto de 2 - 5)
- 8 (valor absoluto de 5 - (-3))
- 10 (valor absoluto de -3 - 7)

Portanto o maior salto é 10. Não esta definido o maior salto para uma lista com menos de 2 elementos.

Defina a função maiorSalto :: [Integer] -> Integer tal que maiorSalto xs é o maior salto da lista xs.

Por exemplo: *maiorSalto [1,5] == 4 // maiorSalto [10,-10,1,4,20,-2] == 22*

```
maiorSalto xs
        | length xs == 2 = abs(head xs – head (tail xs))
        | otherwise = if(z > maiorSalto w) then z else maiorSalto w
                where z = abs(head xs – head(tail xs))
                      w = tail xs
```

OBS.:lenght: retorna o número de itens em uma lista.
abs: valor absoluto do número
head: retorna o primeiro item de uma lista.
tail: ele aceita uma lista e retorna a lista sem seu primeiro item.

46- Você vê a seguinte oferta especial pela loja de conveniência:

"Uma garrafa de Choco Cola para cada 3 garrafas vazias retornadas”

Agora você decide comprar algumas (digamos N) garrafas de refrigerante da loja. Você gostaria de saber como você pode obter o máximo de choco cola deles.

A figura abaixo mostra o caso onde N = 8. O método 1 é o modo padrão: depois de terminar o seu 8 garrafas de cola, você tem 8 garrafas vazias. Tome 6 deles e você recebe 2 novas garrafas de cola. Agora depois bebendo-os você tem 4 garrafas vazias, então você pega 3 delas para obter outra nova cola. Finalmente, você tem apenas 2 garrafas na mão, então você não pode mais usar cola nova. Portanto, você bebeu 8 + 2 + 1 = 11 garrafas de cola. Você pode realmente fazer melhor! No Método 2, você primeiro toma emprestado uma garrafa vazia do seu amigo, então você pode beber de 8 + 3 + 1 = 12 garrafas de cola! Claro, você terá que Devolva sua garrafa vazia restante para o seu amigo.

Defina uma função chococola :: Int -> Int tal que (cola n) devolve o número de garrafas de cola bebidas usando o método 2.

chococola 81 == 121

chococola 40 == 60

chococola 20 == 30

chococola 60 == 90

Dica: Faça uma função auxiliar garrafa :: Int -> Int -> Int tal que (garrafas cheias vazias)

devolve o número de chococolas beb idas considerando que cheias representa o número de garrafas cheias e vazias representa o número de garrafas vazias.

chococola n = n + chococola1 (n + 1)
chococola1 n = if(n <= 2) then 0 else (n `div` 3) + chococola1 (n `div` 3 + n `mod` 3)

OBS.: div: retorna quantas vezes o primeiro número pode ser dividido pelo segundo.
mod: resto da divisão.

FUNÇÕES DE ALTA ORDEM

47- Escreva a definição da função concatenaFold :: [[a]] -> [a] que concatena uma lista de listas usando a função foldr::(a -> b -> b) -> b -> [a] -> b.

Por exemplo: *concatenaFold [[1,2],[3,4]] = [1,2,3,4] // concatenaFold [[1,2],[3,4],[6,7,8]] = [1,2,3,4,6,7,8]*

concatenaFold xs = foldr(\x z → x ++ z)[ ] xs

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.

48- Escreva a definição da função inverteFold :: [a] -> [a] tal que (inverteFold xs) devolve a lista xs invertida usando a função foldr :: (a -> b -> b) -> b -> [a] -> b

Por exemplo: *inverteFold [1,2,,3,4] == [4,3,2,1]*

- Não utilize a função reverse do módulo Prelude.

Dica: *A definição da função inverteFold será*

inverteFold xs = foldr (\x z ->) [] xs

x representa um elemento da lista

z representa a inversão da lista à direita do elemento x

inverteFold xs = foldr(\x z → z ++ [x])[ ] xs

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.

49- Escreva uma função paridadeFold :: [Bool] -> Bool que calcule a paridade de uma lista de boleanos): se o número de valores True for par então a paridade é True, caso contrário é False.

Por exemplo: *paridadeFold [True,True, False,True] = False // paridadeFold [True,True, False,True, True] = True*

paridadeFold xs
| mod (foldr(\x y → if (x == True) then y+1 else y) 0 xs) 2 == 0 = True
| otherwise = False

OBS.: mod: resto da divisão.
foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.

50- A função duplicarFold :: String -> String repete duas vezes cada vogal (letras 'a', 'e', 'i', 'o', 'u' minúsculas ou maiúsculas) numa cadeia de carateres; os outros carateres devem ficar inalterados.

Por exemplo: *duplicar "Ola, mundo!"== "OOlaa, muundoo!"*

Dica: *Crie uma lista com as vogais minúsculas e maiúsculas.*

DuplicarFold :: String → String

vogais = “A, E, I, O, U, a, e, i, o, u”
duplicarFold xs = foldr(\x acc → if (elem x vogais) then x:x:acc else x:acc)[ ] xs

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.
elem: retorna True se a lista contiver um item igual ao primeiro argumento.

51- Defina a função filtraAplicaFold :: (a->b) -> (a->Bool)->[a]->[b] tal que (filtraAplicaFold f p xs) é uma lista obtida aplicando a função f aos elementos de xs que satisfazem o predicado p usando a função foldr.

Por exemplo: *filtraAplicaFold (4+) (<3) [1..7] == [5,6]*

filtraAplicaFold : : (a → b) → (a→Boll) → [a] → [b]

filtraAplicaFold f p xs = foldr(\x acc → if (p x) then f x:acc else acc)[ ] xs

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.

52- Defina função mapFold :: (a->b) -> [a] -> [b] tal que (mapFold f xs) devolve uma lista obtida aplicando a função f a cada elemento da lista xs, ou seja, mapFold f xs == map f xs.

Por exemplo: *mapFold (*2) [1,2,3] == [2,4,6]*

mapFold : : (a → b) → [a] → [b]

mapFold f xs = foldr(\x z → f x:z)[ ] xs

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.

53- Usando a função foldr, defina a função removeLista tal que (removeLista xs ys) remove todo elemento de ys que ocorre na lista xs.

Por exemplo: *removeLista [1,2] [1,1,3,2,2,4,5] == [3,4,5]*

removeLista xs ys = foldr(\x z → if (elem x xs) then z else [x] ++ z)[ ] ys

OBS.: foldr: ele pega o segundo argumento e o último item da lista e aplica a função; depois, o penúltimo item do final e do resultado e assim por diante.
elem: retorna True se a lista contiver um item igual ao primeiro argumento.

#LISTAS INFINITAS