Exercícios de Programação Funcional
Os itens estão mais ou menos em ordem de dificuldade.

1. menorDeDois: recebe dois valores e retorna o menor

2. menorDeTres: recebe três valores e retorna o menor

3. fatorial: recebe um numero natural e retorna o seu fatorial

4. fibonacci: recebe um número inteiro positivo e retorna o n-ésimo elemento da seqüência de Fibonacci (especificar no programa se sua seqüência começa com 0 e 1 ou com 1 e 1)

5. elemento: recebe uma lista e um número inteiro positivo para retornar o n-ésimo elemento da lista
   ex.: (elemento 1 '(3 7 4 2)) ==> 3

6. pertence: recebe uma lista e um elemento qualquer e verifica se o elemento pertence à lista
   ex.: pertence 1 [3,7,4,2] = False

7. nro-elementos: recebe uma lista qualquer e retorna o número de elementos na lista
   obs.: não usar a função length

8. maior: recebe uma lista de números e retorna o maior
   obs.: não usar a função max

9. conta-ocorrencias: recebe um elemento e uma lista qualquer, retorna o número de ocorrências do elemento na lista

10. unica-ocorrencia: recebe um elemento e uma lista e verifica se existe uma única ocorrência do elemento na lista
    ex.:
    unica-ocorrencia 2 [1,2,3,2] = False
    unica-ocorrencia 2 [3,1] = False
    unica-ocorrencia 2 [2] = True

11. maiores-que: recebe um número e uma lista de números, retorna uma lista com os números que são maiores que o fornecido
    ex.:
    (maiores-que 10 '(4 6 30 3 15 3 10 7)) ==> (30 15)

12. concatena: recebe duas listas quaisquer e retorna uma terceira lista com os elementos da primeira no início e os elementos da segunda no fim
    ex.:
    (concatena '() '()) ==> ()
    (concatena '(1 2) '(3 4)) ==> (1 2 3 4)

13. remover: recebe um elemento e uma lista e retorna a lista sem a primeira ocorrência do elemento

14. remover-ultimo: recebe uma lista e remove o último elemento da lista

15. remover-repetidos: recebe uma lista e retorna outra lista sem repetição de elementos
    ex.:
    (remover-repetidos '(7 4 3 5 7 4 4 6 4 1 2)) ==> (7 4 3 5 6 1 2)

16. maiores: recebe um numero natural *n* e uma lista de números, retorna uma lista com os *n* maiores números sem alterar a ordem entre os elementos
    ex.:
    (maiores 4 '(9 3 5 7 8 4 4 7)) ==> (9 7 8 7)

17. gera-sequencia: recebe um número inteiro *n* positivo e retorna a lista (1 -1 2 -2 3 -3 ... *n* -*n*)

18. inverte: recebe uma lista e retorna outra, que contém os mesmos elementos da primeira, em ordem invertida

19. divide: recebe uma lista e um número natural *n*, retorna um par onde o primeiro elemento é uma lista com os *n* primeiros números da lista original e o segundo elemento é uma lista com o resto dos elementos da lista original
    ex.:
    (divide '(1 2 3 4) 0) ==> (() 1 2 3 4)
    (divide '(1 2 3 4) 2) ==> ((1 2) 3 4)

20. intercala: recebe duas listas e retorna outra lista com os elementos das listas originais intercalados.
    ex.:
    (intercala '(1 2 3) '(8 9)) ==> (1 8 2 8 3)
    (intercala '() '(1 2 6)) ==> (1 2 6)

21. uniao: recebe duas listas que não contenham elementos repetidos e retorna uma nova com todos os elementos das duas listas originais (sem repetição)
    ex.:
    (uniao '(3 6 5 7) '(2 9 7 5 1)) ==> (3 6 5 7 2 9 1)

22. interseccao: recebe duas listas sem elementos repetidos e retorna uma lista com os elementos que são comuns às duas
    ex.:
    (interseccao '(3 6 5 7) '(9 7 5 1 3)) ==> (3 5 7)

23. sequencia: recebe dois numeros naturais *n* e *m*, e retorna uma lista com *n* elementos, onde o primeiro é *m*, o segundo é *m*+1, etc...
    ex.:
    (sequencia 0 2) ==> ( ) (sequencia 3 4) ==> (4 5 6)

24. insere-ordenado: recebe uma lista de números em ordem crescente e um número qualquer, retorna uma lista de números em ordem crescente com os elementos da lista inicial mais o número passado.

25. ordenado?: recebe uma lista de números e verifica se eles estão ordenados ou não

26. ordena: recebe uma lista com números e retorna outra lista com os números ordenados
    ex.:
    (ordena '(7 3 5 7 8 4 4)) ==> (3 4 4 5 7 7 8)

27. rodar-esquerda: recebe um número natural, uma lista e retorna uma nova lista onde a posição dos elementos mudou como se eles tivessem sido "rodados"
    ex.:
    (rodar-esquerda 0 '(a s d f g)) ==> (a s d f g) (rodar-esquerda 1 '(a s d f g)) ==> (s d f g a)
    (rodar-esquerda 3 '(a s d f g)) ==> (f g a s d) (rodar-esquerda 4 '(a s d f g)) ==> (g a s d f)

28. rodar-direita: recebe um número natural, uma lista e retorna uma nova lista onde a posição dos elementos mudou como se eles tivessem sido "rodados"
    ex.:
    (rodar-direita 0 '(a s d f g)) ==> (a s d f g)
    (rodar-direita 1 '(a s d f g)) ==> (g a s d f)
    (rodar-direita 3 '(a s d f g)) ==> (d f g a s)
    (rodar-direita 4 '(a s d f g)) ==> (s d f g a)

29. todas-maiusculas: Recebe uma string qualquer e retorna outra string onde todas as letras são maiúsculas. Pode ser útil saber os seguintes códigos ASCII: a=97, z=122, A=65, Z=90, 0=48, 9=57, espaço=32.
    ex.: todas-maiusculas "abc 123" = "ABC 123"

30. primeiras-maiusculas: recebe uma string qualquer e retorna outra string onde somente as iniciais são maíusculas
    ex.:
    (primeiras-maiusculas "FuLaNo bElTrAnO silva") ==>
    "Fulano Beltrano Silva"

31. seleciona: recebe uma lista qualquer e uma lista de posições, retorna uma lista com os elementos da primeira que estavam nas posições indicadas
    ex.:
    (seleciona '(a b c d e f) '(0 3 2 3)) ==> (a d c d)

32. palindrome?: recebe uma string e verifica se ela é uma palíndrome ou nao
    ex.:
    (palindrome? "ana") ==> #t
    (palindrome? "abbccbba") ==> #t
    (palindrome? "abbdbbaa") ==> #f

33. primo?: verifica se um número é primo ou não

34. soma-digitos: recebe um número natural e retorna a soma de seus dígitos
    ex.:
    (soma-digitos 328464584658) ==> 63

35. bolha: recebe uma lista de números e retorna a lista ordenada, pelo método da bolha (bolha burra)

36. compactar: recebe uma lista de números e transforma todas as repetições em sub-listas de dois elementos: sendo o primeiro elemento o número de repetições encontradas e o segundo elemento é o número que repete na lista original. Os números que não repetem na lista original não devem ser alterados.
    ex.:
    (compactar '(2 2 2 3 4 4 2 9 5 2 4 5 5 5)) ==> ((3 2) 3 (2 4) 2 9 5 2 4 (3 5))
    Em Haskell, como não é possível implementar listas heterogêneas, a função deve retornar uma lista de listas. Ex.:
    compactar [2,2,2,3,4,4,2,9,5,2,4,5,5,5] = [[3,2],[3],[2,4],[2],[9],[5],[2],[4],[3,5]]

37. Faça um programa que dada uma lista, retorne uma tupla lista-lista (de inteiros) onde a lista da esquerda contém os números impares e a lista da direita os números pares
    ex:
    func :: [Int] -> ([Int],[Int])
    [1,2,3,4,5,6,7] => ([1,3,5,7],[2,4,6])

38. Dizemos que um *quadrado perfeito* é um numero cuja raiz quadrada é um número inteiro. Sabemos o que a raiz quadrada é um cálculo lento quando comparado à operações como adição ou multiplicação. Implemente uma função que verifica se um número é um quadrado perfeito sem usar uma função que calcula raiz quadrada.

39. Faça um programa que encontra a representação de um número natural numa base *b* qualquer (1 < b < 37). Exemplo:
(muda_base 17 2) ==> "10001"(muda_base 26 16) ==> "1A"

40. O conjunto de todos os subconjuntos de um segundo conjunto é denominado *conjuntos das partes* desse segundo conjunto. Faça um programa que encontra o conjunto das partes de uma lista. Exemplo:
partes [2,3,2,31] = [[],[2],[3],[31],[2,2],[2,3],[2,31],[3,31],[2,2,3],[2,2,31],[2,3,31],[2,2,3,31]]