```
-- ******* Listas ****
-- Sequências de elementos. Todos do mesmo tipo (listas homogêneas)

-- Dois contrutores
-- [] e :
-- [] é a lista vazia (lista sem elementos)
-- x : xs é uma lista cuja cabeça/início é x e o resto é a lista xs
-- Obs, x é um elemento, xs é uma lista

-- []
-- 3:[]
-- 3:(3:[])
-- 5:(3:(3:[]))




-- (:) é associativo à direita, logo
-- 5:(3:(3:[]))
-- podemos escrever assim
-- 5:3:3:[]




-- Podemos escrever por extenso
-- []
-- [3]
-- [3,3]
-- [5,3,3]

-- A avaliação destas listas transforma para a notação usando : e []
-- [5,3,3] --> 5:3:3:[]




-- O operador (++) concatena listas
-- xs ++ ys
-- Forma uma lista juntando os elementos de xs com ys, colocando
primeiro xs
-- A operação xs ++ ys, percorre sequencialmente xs

-- [53,2] ++ [4,2,5] --> [53,2,4,2,5]




-- Listas podem ser comparadas usando (==) e (/=)
-- [53,2,4,2,5] == [53,2,4,2,5] --> True
-- [53,2,4,2,5] /= [2,4] --> True
```

```
-- Ordem dos elementos é relevante
-- [5,3] /= [3,5]

-- Quantidade de elementos é relevante
-- [3] /= [3,3]




-- Tipo de uma lista
-- [5,3,3] :: [Int]
-- [True, False, True] :: [Bool]

lostNumbers :: [Int]
lostNumbers = [4,8,15,16,23,42]




-- Podem haver listas de listas, por exemplo
-- [[1,2,3,4],[5,3,3,3],[1,2,2,3,4],[1,2,3]]
-- Seu tipo é
-- [[1,2,3,4],[5,3,3,3],[1,2,2,3,4],[1,2,3]] :: [[Int]]

-- O que são:
-- []
-- [[]]
-- [[],[],[]]
-- ?




-- Listas são homogêneas
-- [1, 'a']
-- dá erro




-- Strings são listas de caracteres

-- "abcd" --> ['a', 'b', 'c', 'd']

-- > :t "abcd"
-- [Char]

-- >'A':" SMALL CAT"
-- "A SMALL CAT"
```

```
-- concatenação de strings
-- "abcd" ++ "\n" ++ "zxy\n"

-- A funcão pré-definida putStr serve para *visualizar* strings
-- > putStr "abcd" ++ "\n" ++ "zxy\n"
-- abcd
-- zxy
-- >




-- ***** Ranges *****

-- [1..20]

-- ['a'..'z']

-- ['K'..'O']

-- [2,4..20]

-- [3,6..20]

-- [13,26..24*13]

-- [20,17..1]

-- Cuidado com a precisão ao trabalhar com Float ou Double
-- [0.1,0.3..1]
-- [0.1,0.3,0.5,0.7,0.8999999999999999,1.0999999999999999]




-- ******* Exercícios de fixação

-- Escreva expressões para

-- . O string formado pelas letras maiúsculas e minúsculas do
--   alfabeto inglês. Neste string, devem vir primeiro as
--   maiúsculas, na ordem alfabética, e depois as minúsculas em
--   ordem alfabética invertida.

['A'..'Z'] ++ ['z','y'..'a']

-- . Os múltiplos de 3 que estão entre 100 e 1000

[102,105..1000]
```

```
-- funções prédefinidas sobre listas

-- head
-- tail
-- last
-- init
-- head, tail, last e init são funções parciais (podem dar erro)

-- length
-- null
-- reverse
-- take
-- drop

-- maximum
-- minimum
-- sum
-- product
-- concat

-- elem

-- show

-- Exemplos
somaAte :: Int -> Int
somaAte n = sum [1..n]

-- *********
fatorial :: Integer -> Integer
fatorial n = product [1..n]
-- quanto é
-- fatorial 0?


-- Exercícios de fixação
-- Defina funções para

-- . retornar o segundo elemento de uma lista
segundo xs
   | length xs >= 2 = head (tail xs)
   | otherwise = error "list has not enough elements"

-- . calcular quantos dígitos decimais têm um número
nroDigitos :: Int -> Int
nroDigitos n = length (show (abs n))

-- . retornar o i-ésimo elemento de uma lista
esimo :: Int -> [Int] -> Int
esimo i ns
  | 0 <=i && i < length ns = head (drop i ns)
```

```
  | otherwise              = error "index out of bounds"

-- . calcular a média aritmética de uma lista de números

-- . ver se um número Integer é palíndromo
--   (Dica: usar show e reverse)
palindromo :: Int -> Bool
palindromo n = nStr == reverse nStr
  where nStr = show n

-- . verificar se todos os elementos de uma lista são iguais entre
--    eles. (Dica: usar maximum e minimum)
-- . dado um inteiro n positivo, calcular o produto dos números
--   impares de 1 até n.
-- . Calcular o número de combinações de m elementos pegos de um
--   universo de n elementos. A fórmula das combinações é:
--   n! / m!(m-n)!
--   Tente que sua função realize o menor número de multiplicações




-- ********** Listas por compreensão
-- gerar, testar e transformar

-- gerar
[x | x <- [1..10]]
-- temos que
[x | x <- [1..10]] == [1..10]

-- gerar e transformar
[x^2 | x <- [1..10]]

[paraMaiusc c | c <- "Maria Dolores: 10"]

mylength xs = sum [1 | x <- xs ]




-- Avaliação de compreensões usando uma tabela
[x^2 | x <- [1..10]]

x   | 1  2  3  4  5  6  7  8  9   10
x^2 | 1  4  9 16 25 36 49 64  81 100

[paraMaiusc c | c <- "Maria Dolores: 10"]

c           | 'M' 'a' 'r' 'i' 'a' ' ' 'D' 'o' 'l' 'o' 'r' 'e' 's' ':'
' ' '1' '0'
paraMaiusc c | 'M' 'A' 'R' 'I' 'A' ' ' 'D' 'O' 'L' 'O' 'R' 'E' 'S' ':'
' ' '1' '0'
```

```
-- Calcular o valor aproximado da constante de Euler e usando a série
-- 1/0! + 1/1! + 1/2! + 1/3! + ...
eAproxTaylor :: Int -> Float
eAproxTaylor n = sum [1 / fatorial i | i <- [0..n-1] ]

-- Dará erro de tipos, qual?
```

```
eAproxTaylor :: Integer -> Float
eAproxTaylor n =
      sum [1 / fromIntegral (fatorial i) | i <- [0..n-1] ]
```

```
-- *testes* ou *filtros* (gerar, testar e transformar)

[x^2 | x <- [1..10], 2*x >= 12]
-- 2*x >= 12 é chamado de teste ou filtro

-- avaliação
x         | 1  2  3  4  5  6  7  8  9  10
2*x >= 12 | F  F  F  F  F  T  T  T  T   T
x^2       |               36 49 64 81 100

--  outro exemplo
[3*x | x <- [1,3..20], x `mod` 5 == 0 ]

x              | 1  3  5  7  9  11  13  15  17  19
x `mod` 5 == 0 | F  F  T  F  F   F   F   T   F   F
x^2            |      25              225
```

```
-- tira dígitos

tiraDigitos String -> String
tiraDigitos cs = [ c | c <- cs, not (isDigit c) ]

isDigit :: Char -> Bool
```

```
isDigit c = '0' <= c && c <= '9'




-- Outro exemplo:

boomBangs xs =
  [ if x < 10 then "BOOM!" else "BANG!" | x <- xs, impar x]
  where impar n = n `mod` 2 /= 0
-- boomBangs [7..13]?

-- Obs. Haskell oferece as funções even e odd (par e impar,
respectivamente)
boomBangs xs =
   [ if x < 10 then "BOOM!" else "BANG!" | x <- xs, odd x]




-- vários predicados
[ x | x <- [10..20], x /= 13, x /= 15, x /= 19]

[ 3*x | x <- [1..200], odd x, x `mod` 7 == 0 ]

[3*x | x <- [1,3..200], x `mod` 7 == 0 ]

-- Como podemos escrever uma expressão equivalente porém com só um
teste?
[ 3*x | x <- [7,14..200], odd x]


-- Exercícios de Fixação
-- . Defina uma função que dada uma lista cheque se todos são
múltiplos de 5
todosMultiplos5 :: [Int] -> Bool
todosMultiplos5 ns = ns == [n | n <- ns, n `mod` 5 == 0]
-- todosMultiplos5 ns = [] == [n | n <- ns, n `mod` 5 /= 0]

-- . Escreva uma função que devolva uma lista em que os elementos
--   ímpares da lista de entrada aparecem triplicados.

-- . Sem usar maximum nem minimum, escrever uma função que checa
--   se uma lista está formada pela repetição de um único
--   elemento.
-- . Defina uma função que calcule o valor aproximado de Pi
--   utilizando a seguinte série de Leibniz
--   1 - 1/3 + 1/5 - 1/7 + 1/9 - ... = pi/4
--   A função recebe como argumento o número de termos a ser usado
--   para aproximar.
```