```
-- ***** 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
```

```
-- [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"
```

-- Ordem dos elementos é relevante

```
-- 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]
-- [0.1,0.3,0.5,0.7,0.8999999999999,1.09999999999999]
-- ***** 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
-- . calcular quantos dígitos decimais têm um número
-- . retornar o i-ésimo elemento de uma lista
-- . calcular a média aritmética de uma lista de números
-- . ver se um número Integer é palíndromo
    (Dica: usar show e reverse)
-- . 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)!
```

```
-- ******* Listas por compreensão
-- gerar, testar e transformar
-- gerar
-- [x \mid x < - [1..10]]
-- obviamente temos que
-- [x \mid x < - [1..10]] == [1.10]
-- gerar e transformar
-- [x^2 | x < - [1..10]]
-- [paraMaiusc c | c <- "Maria Dolores: 10"]
-- mylength xs = \dots *******
-- Calcular o valor aproximado da constante de Euler e usando a série
-- 1/0! + 1/1! + 1/2! + 1/3! + ...
eAproxTaylor :: Integer -> 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
-- [3*x | x <- [1,3..20], x `mod` 5 == 0 ]
-- tira dígitos
tiraDigitos String -> String
tiraDigitos cs = ... ****
```

Tente que sua função realize o menor número de multiplicações

```
-- Outro exemplo:
boomBangs xs = [if x < 10 then "BOOM!" else "BANG!" | x <- xs, impar
  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]
-- usando a funçao pré-definida odd
-- boomBangs xs = [if x < 10 then "BOOM!" else "BANG!" | x <- xs, odd
x]
-- vários predicados
-- [ x \mid x \leftarrow [10..20], x \neq [13, x \neq 15, x \neq 19]
-- [ 3*x \mid x < - [1..200], odd x, x `mod` k == 0 ]
-- Como podemos escrever uma expressão equivalente porém com só um
teste?
[..** | ...***]
```

- -- Exercícios de Fixação
- -- . Defina uma função que dada uma lista cheque se todos são múltiplos de $\mathbf{5}$
- $\ensuremath{\mathsf{--}}$. Escreva uma função para selecionar os ímpares de uma lista de inteiros.
- -- desta lista e devolve uma lista em que os elementos ímpares aparecem triplicados.
- -- . Sem usar maximum nem minimum, escrever uma função que checa se uma lista está formada
- -- pela repetição de um único elemento