# Listas

De uma forma simples, listas são coleções de elementos do mesmo tipo na qual a ordem importa. Em uma lista, o primeiro elemento é a cabeça e, o restante, a calda. Uma String em Haskell nada mais é do que uma lista de char (`[ Char ] == String`).

Uma lista de 1 a 10 pode ser obtida com `[ 1..10 ], [ 10,9..1 ]` resulta em uma lista de 10 a 1, [2,4..10] resulta em uma lista com os números pares de 2 a 10 e [] representa uma lista vazia.

Para concatenar duas listas, utilizamos ( ++ ), de modo que "UmDois" pode ser obtido com ( `"Um" ++ "Dois"` ). É importante observar que ( ++ ) atua sobre duas listas. De modo que, tentar algo como ( `2 ++ [3,4]` ) não funciona. Para esse caso, o operador correto é ( : ), chamado de "cons". Se quisermos adicionar 'a' no inicio de "manha", por exemplo, basta aplicar ( `a : "manha"` ).

Através desse operador, podemos ilustrar como cada lista é construída. "amanha", pode ser escrita como ( `a : m : a : n : h : a : []` ).

Algumas funções interessantes sobre listas:

`head` retorna a cabeça de uma lista
`head "amanha"` retorna `'a'`

`tail` retorna a calda de uma lista
`tail "amanha"` retorna `"manha"`

É comum que o elemento de uma lista seja outra lista também. Teríamos, nesse caso, uma lista de listas. Por exemplo, ( `head [ "abc", "def" ]` ) retorna `"abc"`.

A função abaixo retorna a soma dos elementos de uma lista. Essa mesma função já é implementada e é conhecida como "sum".

```
sumList :: [ Int ] -> Int
sumList [  ] = 0
sumList ( x:xs ) = x + sumList xs
```

A função sumList recebe uma lista de números inteiros e retorna um inteiro que representa a soma desses elementos contidos na lista. A estratégia é simples: construir uma recursão que retorna o elemento atual somado com a aplicação da

mesma função para os demais elementos da lista. A recursão acaba quando a lista acaba, ou seja, quando a lista é []. Nesse caso, 0 é retornado pondo um fim na recursão.

```
sumList [ 1,2,3 ]
= 1 + ( sumList [2,3] )
= 1 + 2 + ( sumList [3] )
= 1 + 2 + 3 + ( sumList [ ] )
```

Nesse ponto, como definido na função, sumList [ ] = 0. Portanto, a resultado final segue":

```
= 1 + 2 + 3 + 0
= 6
```

A função digits abaixo faz uma filtragem e deixa apenas números em uma string

```
digits :: String -> String
digits x = clear "0123456789" x
    where clear numeros [] = []
          clear numeros ( a:as )
            | elem a numeros = a : digits as
            | otherwise = digits as

firstDigit :: String -> Char
firstDigit st = case ( digits st ) of
                []          -> '\0'
                ( a:as )    -> a
```