Listas

- Listas
- Tipo Lista
- Operações com Listas
- Strings

Listas

- Lista é uma estrutura de dados formada por uma sequência de valores (elementos) do **mesmo tipo**.

- Os elementos de uma lista são identificados pela **posição** em que ocorrem na lista.

Listas

- Em Haskell uma expressão lista é formada por uma sequência de expressões separadas por vírgula e delimitada por colchetes:
- $[exp_1, exp_2, \ldots, exp_n]$
- onde n >= 0, e $exp_1, exp_2, \ldots, exp_n$ são expressões cujos valores são os elementos da lista.

Listas

- Por exemplo:
- [‘a’,‘b’,‘c’] é uma lista de caracteres
- [1,2,3,4,5] é uma lista de números inteiros
- [“aaaa”,”bbbb”,”cccc”] é uma lista de strings
- Portanto, por definição, Lista é uma coleção do **mesmo tipo** de dados separados por vírgula.

Listas

- Como outros tipos de dados, você não precisa declarar uma lista como uma lista.
- Haskell é inteligente o suficiente para decodificar sua entrada olhando para a sintaxe usada na expressão.
- [exp1 , exp2 , exp3, exp4, …. expN]

Listas

- Dê uma olhada no exemplo a seguir, que mostra como Haskell constrói uma Lista.
- Prelude> [1,2,3,4,5]
- Isso produzirá a seguinte saída:

[1,2,3,4,5]

Listas

- Também podemos associar um nome a lista e construí-la dessa maneira:
- Prelude> let numeros = [1,2,3,4,5]
- Prelude> numeros
- Isso produzirá a seguinte saída:

[1,2,3,4,5]

Listas

- Podemos construir uma Lista de caracteres.
- Prelude> let caracteres = [‘a’,’b’,’c’,’d’]
- Prelude> caracteres
- Isso produzirá a seguinte saída:
- [‘a’,’b’,’c’,’d’] ou “abcd”

Listas

- Podemos construir uma Lista de listas.
- Isto é, os elementos de uma lista podem ser outras listas
- Prelude> let listadelistas = [[1,2],[3,4],[5,6]]
- Prelude> listadelistas
- Isso produzirá a seguinte saída:

[[1,2],[3,4],[5,6]]

Listas

- As listas em Haskell são homogêneas por natureza,
- O que significa que não permitem que você declare uma lista de tipos diferentes de dados.
- Qualquer lista como [1,2,3,4,5, ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’] irá produzir um erro.

Listas

- Prelude> [1,2,3,4,5,’a’,’b’,’c’,’d’,’e’,’f’]
- Este código produzirá o seguinte erro:
- <interactive>:17:12: Variable not in scope: 'a'
- <interactive>:17:14: Not in scope: 'b'
- <interactive>:17:16: Not in scope: 'c'
- <interactive>:17:18: Not in scope: ‘d'
- <interactive>:17:20: Not in scope: 'e'
- <interactive>:17:22: Not in scope: 'f'

Listas

- Prelude> [‘1’,’2’,’3’,’4’,’5’,’a’,’b’,’c’,’d’,’e’,’f’]
- E agora?

Tipo Lista

- Um tipo lista é formado pelo tipo dos seus elementos delimitado por colchetes:
  - [t]
- onde t é o tipo dos elementos da lista.
- Por exemplo:
- caracteres::[Char]
- Observe que o tamanho de uma lista (quantidade de elementos) não é codificado no seu tipo.

Tipo Lista

- A tabela a seguir mostra alguns exemplos de tipos de lista:

lista	tipo
`['O','B','A']`	`[Char]`
`['B','A','N','A','N','A']`	`[Char]`
`[False,True,True]`	`[Bool]`
`[ [False,True], [], [True,False,True] ]`	`[[Bool]]`
`[1,8,6,10.48,-5]`	`Fractional a => [a]`

Operações com Listas

- Vejamos algumas operações com listas definidas na biblioteca padrão:
- null: verifica se uma lista é vazia:
- Prelude> null [ ]

True

- Prelude> null [1,2,3,4,5]

False

- Prelude> null [‘a’,’b’,’c’,’d’]

False

Operações com Listas

- Também podemos associar uma variável a uma operação com listas:
- Prelude> v = null [ ]
- Prelude> v

True

Operações com Listas

- head: seleciona a cabeça (primeiro elemento) de uma lista:
- Prelude> head [1,2,3,4,5]

1

- Prelude> head[‘a’, ‘b’, ‘c’, ‘d’]

‘a’

- head [3]

?

Operações com Listas

- head: seleciona a cabeça (primeiro elemento) de uma lista (cont.):
- Prelude> head [3]

3

- Prelude> head[‘a’]

‘a’

- Prelude> head [ ]

*** Exception: Prelude.head: empty list

Operações com Listas

- tail: seleciona a cauda da lista, ou seja, a lista formada por todos os elementos exceto o primeiro:
- Prelude> tail [1,2,3,4,5]

[2,3,4,5]

- Prelude> tail [‘a’,’b’,’c’,’d’]

[‘b’,’c’,’d’] ou “bcd”

- Prelude> tail [5*4, 5*6]

[30]

Operações com Listas

- tail: seleciona a cauda da lista, ou seja, a lista formada por todos os elementos exceto o primeiro (continuação):
- Prelude>tail [7]

[ ]

- Prelude> tail [8-1]

[ ]

- Prelude> tail [ ]

*** Exception: Prelude.tail: empty list

Operações com Listas

- init: seleciona todos os elementos exceto o último:
- Prelude> init [1,2,3,4,5]

[1,2,3,4]

- Prelude> init [‘a’,’b’,’c’,’d’]

[‘a’,’b’,’c’] ou “abc”

Operações com Listas

- length: calcula o tamanho (quantidade de elementos) de uma lista:
- Prelude> length [1,2,3,4,10]

5

- Prelude> length [‘a’,’b’,’c’,’d’]

4

- Prelude> length [ ]

0

Operações com Listas

- (!!): seleciona o i-ésimo elemento de uma lista ($0 \leq i < n$, onde n é o comprimento da lista):
- Prelude> [1,2,3,4,5] !! 2

3

- Prelude> [1,2,3,4,5] !! 0

1

- Prelude> [1,2,3,4,5] !! 10

*** Exception: Prelude.(!!): index too large

Operações com Listas

- (!!): seleciona o i-ésimo elemento de uma lista ($0 \leq i < n$, onde n é o comprimento da lista) (continuação):
- Prelude> [‘a’,’b’,’c’,’d’] !! 1

‘b’

- Prelude> [‘a’,’b’,’c’,’d’] !! 0

‘a’

- Prelude> [‘a’,’b’,’c’,’d’] !! 6

*** Exception: Prelude.(!!): index too large

Operações com Listas

- take: seleciona os primeiros n elementos de uma lista:
- Prelude> take 3 [1,2,3,4,5]

[1,2,3]

- Prelude> take 3 [‘a’,’b’,’c’,’d’]

[‘a’,’b’,’c’] ou “abc”

Operações com Listas

- drop: remove os primeiros n elementos de uma lista:
- Prelude> drop 3 [1,2,3,4,5]

[4,5]

- Prelude> drop 3 [‘a’,’b’,’c’,’d’]

[‘d’] ou “d”

Operações com Listas

- sum: calcula a soma dos elementos de uma lista de números:
- Prelude> sum [1,2,3,4,5]

15

- Prelude> s = sum [1,2,3,4,5,6]

s ?

Operações com Listas

- product: calcula o produto dos elementos de uma lista de números:
- Prelude> product [1,2,3,4,5]

120

Operações com Listas

- (++): concatena duas listas:
- Prelude> [1,2,3] ++ [4,5]

[1,2,3,4,5]

- Prelude> [‘a’,’b’] ++ [‘c’,’d’]

[‘a’,’b’,’c’,’d] ou “abcd”

Operações com Listas

- reverse: inverte uma lista:
- Prelude> reverse [1,2,3,4,5]

[5,4,3,2,1]

Prelude> reverse [‘a’,’b’,’c’,’d’]

[‘d’,’c’,’b’,’a’] ou “dcba”

Operações com Listas

- zip: junta duas listas em uma única lista formada pelos pares dos elementos correspondentes:
- Prelude> zip ["pedro","ana","carlos"] [19,17,22]

[("pedro",19),("ana",17),("carlos",22)]

Strings

- Em Haskell strings são listas de caracteres.
- O tipo String é um sinônimo para o tipo [Char] .
- A tabela a seguir mostra alguns exemplos de strings:

string	notação de lista
`"ufop"`	`['u','f','o','p']`
`"bom\ndia"`	`['b','o','m','\n','d','i','a']`
`""`	`[]`

Funções sobre Strings

- words: fornece a lista de palavras de um texto
- tipo da função:
- words :: String -> [String]
- Exemplo:
- Prelude> words “aaaa bbbb cccc”
- [“aaaa”,”bbbb”,”cccc”]

Funções sobre Strings

- unwords: constrói um texto a partir de uma lista de palavras
- tipo da função:
- unwords :: [String] -> String
- Exemplo:
- Prelude> unwords [“aaaa”,”bbbb”,”cccc”]
- “aaaa bbbb cccc”

Funções sobre Strings

- lines: fornece a lista de linhas de um texto
- tipos da função:
- lines :: String -> [String]
- Exemplo:
- Prelude> lines “aaaa bbbb cccc\neeee ffff gggg”
- [“aaaa bbbb cccc”,”eeee ffff gggg”]

ATS 1

- Verifique se as seguintes expressões são válidas e determine o seu tipo em caso afirmativo.
- a) [’a’,’b’,’c’]
- b)[a,b,c]
- c) [[False,True],[’0’,’1’]]
- d) [tail,init,reverse]

ATS 1

- Verifique se as seguintes expressões são válidas e determine o seu tipo em caso afirmativo.
- e) [[]]
- f) [[10,20,30],[],[5,6],[24]]
- g) ["bom","dia","brasil"]
- h) [sum,length]