Princípios de Programação

Exercícios

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Engenharia Informática

2019/2020

Começando pelo princípio

Tópicos endereçados neste capítulo: Expressões, algumas funções pré-definidas, as minhas primeiras funções, listas e algumas funções pré-definidas sobre listas, intervalos, listas infinitas, listas em compreensão, tuplos.

1. Escreva funções que recebam três inteiros e que devolvam:
   (a) a sua soma.
   (b) a sua soma se forem todos positivos e zero caso contrário.
2. Escreva uma função que receba três inteiros e devolva `True` se a distância entre os dois primeiros for inferior ao terceiro e `False` caso contrário. Assuma que o terceiro inteiro é não negativo. Sugestão: utilize a função `abs` para obter o valor absoluto de um número.
3. Escreva uma função `addDigit` que receba dois inteiros, o segundo dos quais entre `0` e `9`, e que devolva o inteiro resultante de acrescentar o segundo no fim do primeiro. Por exemplo:

```
ghci> addDigit (-123) 4
-1234
```

4. Quantos elementos tem cada uma das seguintes listas?
   (a) `['a','b']`
   (b) `"ab"`

(c) `[['a','b']]`

(d) `[['a','b'], ['c','d']]`

(e) `[[['a','b']]]`

(f) `[]`

(g) `[[]]`

(h) `[[],[]]`

5. Para resolver os exercícios abaixo considere as seguintes funções constantes no Prelude.

   - `1 : [2,3]` _ devolve a lista obtida pela junção de um elemento à cabeça de uma lista, `[1,2,3]`.
   - **`head`** `[1,2,3]` _ devolve o primeiro elemento de uma lista, `1`.
   - **`tail`** `[1,2,3]` _ devolve a lista excluindo o primeiro elemento, `[2,3]`.
   - **`last`** `[1,2,3]` _ devolve o último elemento de uma lista, `3`.
   - **`init`** `[1,2,3]` _ devolve a lista excluindo o último elemento, `[1,2]`.
   - **`null`** `[1,2,3]` _ devolve true se a lista for vazia, **`False`**.
   - **`length`** `[1,2,3]` _ devolve o número de elementos na lista, `3`.
   - **`reverse`** `[1,2,3]` _ devolve a lista, mas em ordem inversa, `[3,2,1]`.
   - **`take`** `2 [1,2,3]` _ devolve os primeiros `2` (ou n) elementos da lista, `[1,2]`.
   - **`sum`** `[1,2,3]` _ devolve a soma dos elementos da lista, `6`.

   Escreva as seguintes funções:

   (a) Uma função que devolve **`True`** se uma dada lista tem mais que `10` elementos, **`False`** caso contrário.

   (b) Uma função que verifica se uma lista *não* é vazia.

   (c) Uma função que retira o primeiro e o último caracter de uma **`String`**.

   (d) Uma função que devolve o segundo elemento de uma lista.

   (e) Uma função que devolve o penúltimo elemento de uma lista.

   (f) Uma função que devolve o n-ésimo elemento de uma lista. Assuma que os índices começam em zero e que n está entre 0 e o comprimento da lista menos um. Reescreva as duas funções anteriores.

(g) Uma função que inverte todos os elementos de uma lista excepto o primeiro. O primeiro elemento da lista permanece na primeira posição.

(h) Uma função que calcula o somatório dos primeiros `5` elementos de uma lista.

(i) Uma função que calcula o somatório dos primeiro `n` elementos de uma lista. Reescreva a função da alínea anterior utilizando este resultado.

(j) Uma função que recebe duas listas e devolve verdadeiro se a) o último elemento de ambas as listas for igual, b) as listas tiverem o mesmo comprimento, e c) as listas forem não nulas.

6. Prefixos e sufixos de listas.

   (a) Defina uma função que verifica se uma string é um prefixo de uma outra string.

   ```
   ghci> prefix "ab" "abcd"
   True
   ghci> prefix "" "abcd"
   True
   ghci> prefix "abcde" "abcd"
   False
   ```

   (b) Defina uma função que verifica se uma string é sufixo de uma outra string. (Dica: prefixo e sufixo são conceitos bastante parecidos...)

   ```
   ghci> suffix "bcde" "abcde"
   True
   ghci> suffix "abc" "abcd"
   False
   ```

   (c) Escreva a função `prefixOrSuffix` baseada nas anteriores.

7. Escreva uma função que divide o intervalo entre dois valores em `n` partições iguais. O resultado deverá ser uma lista de `n+1` elementos onde a primeira partição é dada pelos primeiros dois elementos da lista, a segunda partição pelo segundo e terceiro elementos da lista, e assim adiante. Por exemplo:

   ```
   ghci> particao 0 1 2
   [0.0,0.5,1.0]
   ghci> particao 10 20 4
   [10.0,12.5,15.0,17.5,20.0]
   ```

8. Determine o valor de cada expressão.

   (a) `[2*x | x <- [1,2,3]]`

(b) `[x^2 | x <- [1..8], x 'mod' 2 == 0]`

(c) `[x | x <- ['6'..'S'], isDigit x]`

(d) `[(x,y)| x <- [1..3], odd x, y <- [1..3]]`

(e) `[(x,y)| x <- [1..3], y <- [1..3], odd x]`

9. Utilizando uma lista em compreensão escreva uma expressão que calcule a soma $1^2 + 2^2 + ... + 100^2$ dos quadrados dos primeiros 100 inteiros.

10. Dizemos que triplo (x, y, z) é Pitagórico se $x^2 + y^2 = z^2$. Utilizando uma lista em compreensão defina a função `pitagoricos` que calcule a lista de todos os triplos pitagóricos até um dado limite. Evite colocar triplos que representem o mesmo triângulo, por exemplo `(3,4,5)` e `(4,3,5)`.

```
ghci> pitagoricos 10
[(3,4,5),(6,8,10)]
```

11. Dizemos que um inteiro positivo é *perfeito* se é igual à soma de todos os seus fatores, excluindo o próprio número.

    (a) Utilizando uma lista em compreensão escreva a função `fatores`, que devolve os fatores do inteiro dado (por uma qualquer ordem).

    (b) Defina agora uma função `perfeitos` que calcula a lista de todos os números perfeitos até um dado limite.

    Por exemplo:

```
ghci> perfeitos 500
[6,28,496]
```

12. Defina a lista infinita com todas as potências de dois.

13. Defina a função `produtoEscalar` que calcula o produto escalar de dois vetores:

$$\text{produtoEscalar } x\ y = \sum_{i=0}^{n-1} x_i * y_i$$

    Assuma que cada vetor é representado por uma lista e que as duas listas têm o mesmo comprimento.

14. Utilizando uma lista em compreensão defina uma função com assinatura `reproduz` que troca cada número n numa lista de inteiros positivos por n cópias dele próprio. Por exemplo:

```
ghci> reproduz [3,5,1]
[3,3,3,5,5,5,5,5,1]
```

Sugestão: Utilize a função **replicate** onde **replicate** `n x` é uma lista de comprimento `n` em que cada elemento é igual a `x`.

15. Mostre como uma lista em compreensão com geradores duplos (por exemplo, `[(x,y)| x <- [1,2,3], y <- [4,5,6]]`) pode ser descrita utilizando apenas geradores simples. Sugestão: utilize a função **concat** e uma compreensão dentro da outra.

16. Quais das seguintes frases são expressões Haskell?

    (a) `['a','b','c']`

    (b) `('a','b','c')`

    (c) `['a',True]`

    (d) `[True, False]`

    (e) `["a disciplina de PP", "é fixe"]`

    (f) `[('a',False),('b',True)]`

    (g) `[isDigit 'a', isLower 'f', isUpper 'h']`

    (h) `(['a','b'],[False,True])`

    (i) `[['a','b'],[False,True]]`

    (j) `[isDigit, isLower, isUpper]`

17. Defina a função `pares` de modo a que `pares n` seja a lista composta por todos os valores (i, j) com $1 \leq i, j \leq n$ que satisfaçam a condição $i \neq j$.

18. Defina a função `fromTo` que obtém a secção de uma lista entre dois índices. Por exemplo:

    ```
    ghci> fromTo 2 4 "gfedcba"
    "edc"
    ```

    Defina as funções **tail**, **init** e `!!` usando `fromTo`.

19. Defina a função `matId` que, recebendo um inteiro positivo, devolve a matriz identidade com essa dimensão. Por exemplo, a matriz identidade de dimensão 3 será `[[1,0,0],[0,1,0],[0,0,1]]`.

20. Defina a função `matMult` que recebe duas matrizes e devolve o resultado da multiplicação dessas duas matrizes.