Paradigmas de Programação

Fabrício Olivetti de França

13 de Junho de 2018

Lista de exercícios 02

1. Defina uma função

```
penultimo :: [a] -> a
```

que devolve o penúltimo elemento de uma lista, apresentando uma mensagem de erro nos casos de lista vazia e lista com apenas um elemento.

2. Defina uma função

```
maximoLocal :: [Int] -> [Int]
```

que devolve uma lista com os máximos locais de uma lista de inteiros. Um máximo local é um elemento maior que seu antecessor e que seu sucessor. Por exemplo, em [1,3,4,2,5] 4 é um máximo local, mas 5 não, pois não possui sucessor.

3. Usando compreensão de listas, defina a função

```
perfeitos :: Int -> [Int]
```

que recebe um inteiro n e retorna uma lista dos números perfeitos até n. Um número perfeito é igual à soma de seus fatores, excluindo a si mesmo. O número 28 é perfeito, pois 1 + 2 + 4 + 7 + 14 = 28. Exemplo:

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

4. Defina a função

```
produtoEscalar :: Num a => [a] -> [a] -> a
```

que devolve o produto escalar de dois vetores, usando compreensão de listas.

5. Escreva uma função recursiva

```
palindromo :: [Int] -> Bool
```

que verifica se os elementos da lista formam um palíndromo.

6. Defina uma função

```
ordenaListas :: (Num a, Ord a) => [[a]] -> [[a]]
```

que ordene uma lista de listas pelo tamanho de suas sublistas.

7. Mostre como a compreensão de lista

```
coord :: [a] -> [a] -> [(a,a)]
coord x y = [(i,j) | i <- x, j <- y]
```

que possui duas funções geradoras, pode ser redefinida com duas listas de compreensão aninhadas, cada uma contendo uma única função geradora.

8. O algoritmo de Luhn para a verificação dos dígitos de um cartão de crédito segue os seguintes passos:

a. considere cada dígito como um número
b. a partir da direita, dobre os números alternadamente, começando pelo penúltimo
c. some todos os dígitos dos números
d. se o total for divisível por 10, o número do cartão é válido.

a. Defina as funções

```
digitosRev :: Int -> [Int]
```

que converte um inteiro em uma lista contendo seus dígitos na ordem reversa.

b. Escreva a função

```
dobroAlternado :: [Int] -> [Int]
```

que recebe uma lista de números e dobra a partir da esquerda o segundo, quarto etc, elemento, devolvendo uma lista atualizada. Note que por termos escrito a função anterior para retornar os dígitos invertidos, podemos fazer essa operação da esquerda ao invés de da direita conforme descrição do algoritmo. Por exemplo, para [3,5,6,4] a a saída é [3,10,6,8].

c. Defina a função

```
somaDigitos :: [Int] -> Int
```

que soma todos os dígitos da lista de inteiros. Com o uso função anterior, alguns números possuem dois dígitos, que precisam ser somados individualmente. Exemplo: [6,5,12,4] = 6 + 5 + 1 + 2 + 4 = 18

d. Utilize as funções criadas anteriormente para definir a função

```
luhn :: Int -> Bool
```

que verifica se o número é uma sequência válida para um cartão de crédito. Exemplos:

```
> luhn 1784
True
> luhn 4783
False
> luhn 4012888888881881
True
> luhn 4012888888881882
False
```