Introdução a programação funcional com Haskell

Lucas Carvalho dos Santos

Sumário

Objetivo geral

- Conhecer as principais características de Haskell como linguagem funcional.

Objetivos específicos

- Conhecer as características da programação funcional e suas diferenças com a programação imperativa.
- Testar operações simples da linguagem
- Implementar códigos que utilizam listas em Haskell usando um ambiente de desenvovimento.

Características da programação funcional e diferenças com programação imperativa

Programação Funcional:

- é declarativa;
- é baseada em funções puras (sem efeitos colaterais)
- trabalha com dados imutáveis;
- seu foco está em o que deve ser computado.

Programação Imperativa:

- é baseada na sequência de instruções;
- alterações no estado do programa
- trabalha com dados mutáveis;
- seu foco está em como deve ser computado.

O que é uma função lambda

- é função anônima (sem nome);
- definida em uma única linha que é usada principalmente para expressar cálculos simples e concisos.;
- consiste em passar uma função como argumento para outra função.

Exemplo em Haskell:

```
main :: IO()
main = do
    let soma = \x y -> x + y
    print (soma 5 7)
```

O que é a estratégia de avaliação lenta

A estratégia de avaliação lenta ou preguiçosa (lazy evaluation) é uma técnica em programação funcional:

- em que as expressões só são avaliadas quando o seu valor é realmente necessário;
- permite o uso de estruturas como listas infinitas sem estourar a memória;
- que evita cálculos desnecessários.

Exemplos de lazy evaluation em Haskell

Exemplo 1

```
*Main> x =  5 + 10
*Main> :sprint x
x = _
*Main> x
15
*Main> :sprint x
x = 15
```

“_” = **thunk**
(expressão ainda não avaliada)

Exemplo 2

```
*Main> x = 3 + 7
*Main> :sprint x
x = _
*Main> y = x * 2
*Main> :sprint x
x = _
*Main> :sprint y
y = _
*Main> y
20
*Main> :sprint x
x = 10
*Main> :sprint y
y = 20
```

Recursividade em Haskell

Fatorial recursivo em Haskell.

```haskell
fatorial :: (Eq t, Num t) => t -> t
fatorial 0 = 1
fatorial n = n * fatorial (n - 1)

main :: IO()
main = do
    let n = 5
    let resultado = fatorial n
    putStr $ "Fatorial de " ++ show n ++ ": " ++ show resultado
```

```
Fatorial de 5: 120
```

Códigos que utilizam listas em Haskell

Código que retorna uma lista com os números triplicados da lista original

```haskell
main :: IO ()
main = do
    let multPor3 xs = [x * 3 | x <- xs]
    print (multPor3 [2,3,4,5])
```

```
Saida = [6,9,12,15]
```

xs é a lista de entrada;

[x * 3 | x <- xs]: É uma list comprehension que:

- Itera sobre cada elemento x da lista xs;
- Multiplica cada x por 3;
- Cria uma nova lista com os resultados.

Códigos que utilizam listas em Haskell

Código que retorna a soma de todos os valores de uma lista

```haskell
main :: IO ()
main = do
    let somaLista xs = sum xs
    print (somaLista [2,3,4,5])
```

```
Saída = 14
```

Usa a função sum, que calcula a soma de todos os elementos de uma lista.

Códigos que utilizam listas em Haskell

Código que retorna uma terceira lista sendo a concatenação de outras duas

```
main :: IO()
main = do
    let concatListas l1 l2 = l1 ++ l2
    print (concatListas [2,3,4,5] [6,7])
```

```
Saida = [2,3,4,5,6,7]
```

Usa o operador ++ para concatenar duas listas, ou seja, combinar os elementos de l1 e l2.

Códigos que utilizam listas em Haskell

Código que dada duas listas combina as duas listas em uma lista de pares

```
main :: IO()
main = do
    let combinaListas l1 l2 = zip l1 l2
    print (combinaListas [1,2,3] ['a', 'b', 'c'])
```

```
Saída = [(1, 'a'), (2, 'b'),(3, 'c')]
```

zip [1,2,3] ['a', 'b', 'c']:

Combina os elementos das duas listas:

- 1 com ‘a’ = (1, ‘a’)
- 2 com ‘b’ = (2, ‘b’)
- 3 com ‘c’ = (3, ‘c’)

Códigos que utilizam listas em Haskell

Código que verifica se uma lista está vazia

```haskell
main :: IO()
main = do
     let listaVazia xs = null xs
     print (listaVazia [])
     print (listaVazia [2,3,4,5])
```

Saída:

```
True
False
```

Função null retorna True se uma lista está vazia, senão retorna False.

Obrigado!