Módulo Prelude

Wladimir Araújo Tavares

Universidade Federal do Ceará - Quixadá

27 de Fevereiro de 2018

operações em listas

sublistas

funções zip e unzip

strings

listas infinitas

funções especiais de dobra (fold)

operações em listas

```
Prelude> :t map
map :: (a -> b) -> [a] -> [b]
Prelude> map (2*) [1,2,3,4,5]
[2,4,6,8,10]
Prelude> map (*2) [1,2,3,4,5]
[2,4,6,8,10]
Prelude> [1,2,3] ++ [4,5,6]
[1,2,3,4,5,6]
Prelude> :t filter
filter :: (a -> Bool) -> [a] -> [a]
Prelude> filter (even) [1,2..10]
[2,4,6,8,10]
Prelude> :t init
init :: [a] -> [a]
Prelude> :t reverse
reverse :: [a] -> [a]
```

operações em listas

```
Prelude> null [1,2,4]
False
Prelude> null []
True
Prelude> elem 4 [5,6,2,7]
False
Prelude> elem 4 [5,6,4,2,7]
True
Prelude> length [4,5,6,7,8]
5
Prelude> [4,5,6,7,8] !! 1
5
Prelude> (!!) [4,5,6,7,8] 4
8
```

sublistas

```
Prelude> take 2 [1,2..10]
[1,2]
Prelude> :t splitAt
splitAt :: Int -> [a] -> ([a], [a])
Prelude> splitAt 4 [1..10]
([1,2,3,4],[5,6,7,8,9,10])
Prelude> :t takeWhile
takeWhile :: (a -> Bool) -> [a] -> [a]
Prelude> takeWhile (>2) [4,5,6,1,6]
[4,5,6]
Prelude> dropWhile (>2) [4,5,6,1,6]
[1,6]
Prelude> :t break
break :: (a -> Bool) -> [a] -> ([a], [a])
Prelude> break (>3) [1,2,1,3,4,6,7]
([1,2,1,3],[4,6,7])
```

zip e unzip

```
Prelude> :t zip
zip :: [a] -> [b] -> [(a, b)]
Prelude> zip [1,2,3] [4,5,6]
[(1,4),(2,5),(3,6)]
Prelude> zipWith (+) [1,2,3] [4,5,6]
[5,7,9]
Prelude> unzip( zip [1,2,3] [4,5,6] )
([1,2,3],[4,5,6])
Prelude> :t unzip
unzip :: [(a, b)] -> ([a], [b])
Prelude> :t zipWith
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
```

strings

```
Prelude> lines "1\n2\n3\n4\n"
["1","2","3","4"]
Prelude> words "1 2 3 4 5"
["1","2","3","4","5"]
Prelude> unlines ["1","2","3","4"]
"1\n2\n3\n4\n"
Prelude> unwords ["1","2","3","4"]
"1 2 3 4"
Prelude> :t lines
lines :: String -> [String]
Prelude> :t words
words :: String -> [String]
Prelude> :t unlines
unlines :: [String] -> String
Prelude> :t unwords
unwords :: [String] -> String
```

strings

```
Prelude> :t notElem
notElem :: (Eq a, Foldable t) => a -> t a -> Bool
Prelude> notElem 5 [1,2,3,4,5,6]
False
Prelude> notElem 5 [1,2,3,4,6]
True
Prelude> :t lookup
lookup :: Eq a => a -> [(a, b)] -> Maybe b
Prelude> let id = zip ["ufc","uece","catolica"] [2,3,4]
Prelude> id
[("ufc",2),("uece",3),("catolica",4)]
Prelude> lookup "uece" id
Just 3
Prelude> lookup "ufc" id
Just 2
Prelude> lookup "ifce" id
Nothing
```

listas infinitas

```
Prelude> let pot = iterate (2*) 1
Prelude> take 4 pot
[1,2,4,8]
Prelude> take 10 pot
[1,2,4,8,16,32,64,128,256,512]
Prelude> let l = repeat 3
Prelude> take 10 l
[3,3,3,3,3,3,3,3,3,3]
Prelude> replicate 4 2
[2,2,2,2]
Prelude> :t replicate
replicate :: Int -> a -> [a]
Prelude> let l = cycle [0,1,2,3]
Prelude> take 5 l
[0,1,2,3,0]
Prelude> take 20 l
[0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3]
```

funções especiais de dobra (fold)

```
Prelude> let l = map (even) [2,4,5,8]
Prelude> and l
False
Prelude> or l
True
Prelude> any (>0) [-1,-2,5]
True
Prelude> any (>0) [-1,-2,-3]
False
Prelude> all (>0) [-1,-2,-3]
False
Prelude> all (>0) [1,2,3]
True
Prelude> concat [[1,2],[4,5],[6,7]]
[1,2,4,5,6,7]
```

funções especiais de dobra (fold)

```
Prelude> sum [1..10]
55
Prelude> foldr (+) 0 [1..10]
55
Prelude> product [1..5]
120
Prelude> foldr (*) 1 [1..5]
120
Prelude> maximum [2,3,4,5]
5
Prelude> and [True,True,False]
False
Prelude> foldr (&&) True [True,True, False]
False
```