Exercícios sobre funções de alta ordem e funções anônimas

prof. André Rauber Du Bois

Universidade Federal de Pelotas
dubois@inf.ufpel.edu.br

1 Questionário

1. Defina a função

   `aplica_duas_vezes/2`

   que recebe uma função e um valor inicial e aplica a função duas vezes nesse valor inicial

   ```
   iex(1)> Aula7.aplica_duas_vezes(fn (x) -> x+1 end, 10)
   12
   ```

2. Implementar a função recursiva `iter/3` que recebe uma função `f`, um número de repetições e um valor inicial e computa:

   ```
   iter(f,1,n) = f(n)
   iter(f,2,n) = f(f(n))
   iter(f,3,n) = f(f(f(n)))
   (...)
   ```

3. Uma outra função de alta ordem muito utilizada em programação funcional é a função `filter/2`. A função `filter` recebe uma lista e um predicado (função que recebe um valor e devolve um booleano), e filtra a lista de acordo com o predicado. Exemplos:

   ```
   iex(1)> Aula7.filter([1,2,3,4,5,6], fn (x) -> x > 3 end)
   [4, 5, 6]

   iex(2)> Aula7.filter([1, 2, 3, 4, 5, 6], fn x -> rem(x, 2) == 0 end)
   [2, 4, 6]
   ```

   Implementar a função filter

4. Outra função de alta ordem muito utilizada em programação funcional é a função `reduce` (também chamada de `foldl` em algumas linguagens). A função `reduce` recebe uma lista, um valor inicial e uma função de dois argumentos (operador), e aplica essa função *entre* os elementos da lista, ou seja:

```
reduce([a,b,c], acc, f) = f(f(f(acc,a),b),c)
```

Por exemplo:

```
iex(1)> Aula7.reduce([1,2,3,4],0, fn (x,y) -> x + y end)
10
```

Que calcula:

```
((((0+1)+2)+3)+4)
```

Implementar a função `reduce`

5. Outra versão da função `reduce` seria o `foldr`:

   ```
   foldr([a,b,c], acc, f) = f(a,f(b,f(c,acc)))
   ```

   Por exemplo, o programa:

   ```
   iex(1)> Aula7.foldr([1,2,3,4],0, fn (x,y) -> x + y end)
   10
   ```

   calcula:

   ```
   1 + (2 + ( 3 + (4 + 0)))
   ```

   Implementar a função `foldr`
6. Implementar a função `concatena` usando `foldr`:

   ```
   iex(1)> Aula7.concatena([[1,2],[4,5,6],[5]])
   [5, 4, 5, 6, 1, 2]
   ```

7. Usando `map`, `filter` e `reduce`, implemente a função `soma_quad_positivos/1` que calcula a soma dos quadrados de todos os números positivos de uma lista.
8. Implementar uma função que conta quantos elementos tem uma lista usando `map` e `reduce`
9. Implementar a função `conta_neg` que conta quantos números negativos tem em uma lista. Usar `map` e `reduce` para implementar essa função.
10. Implementar uma função que soma uma lista de listas usando `map` e `reduce`:

    ```
    iex(1)> soma_lista_lista([[1,2],[3,4,5],[6]])
    21
    ```