02

Processamento de Listas em Sequência

Capítulo 7 - Processamento de listas em sequência

Nesta aula iremos trabalhar um pouco mais com tratamento de listas em Clojure. Já terminamos nosso jogo de forca e podemos ver outros conceitos agora.

Comecemos definindo uma lista com preços de carros através do Terminal:

```
user=> (def carros [50000.0, 60000.0])
#'forca.core/carros
forca.core=> carros
[50000.0 60000.0]
```

Aprendemos que o `map` pega os valores e aplica uma função:

```
user=> (map (fn [x] (* x 2)) carros)
(100000.0 120000.0)
```

Existe ainda uma outra função chamada `reduce`, a qual transforma todos os elementos de uma lista em um só:

```
user=> (reduce (fn [acc n] (+ acc n)) carros)
110000.0
```

A função que será aplicada recebe dois valores: o acumulador (`acc`) e o valor corrente (`n`). Podemos acumular a soma, a multiplicação, e assim por diante. Neste caso os elementos foram somados.

Se você ainda não ouvira falar de `map`, `reduce`, com certeza ouvirá bastante quando estudar *bigdata*. O algoritmo de *MapReduce* é muito comum para dividir o processamento em várias máquinas e juntar depois em um lugar só.

Revendo os conceitos, temos:

- `map` : aplica uma função qualquer em cada elemento de uma lista,
- `reduce` : transforma a lista em apenas um elemento usando algum acumulador (soma, multiplicação, etc).

Ambas as funções são muito dinâmicas e podem ser utilizadas em conjunto usando o atalho `->>` :

```
user=> (->> carros (map (fn [x] (* x 2))) (reduce (fn [acc n] (+ acc n))))
220000.0
```

Perceba que primeiro foi feito o produto dos elementos por 2 e depois a acumulação dos mesmos usando soma:

1. `[50000.0 * 2, 60000.0 * 2] = [100000.0, 120000.0]`
2. `100000.0 + 120000.0 = 220000.0`

Se quisermos inserir esse código em um documento de Clojure, poderemos formatá-lo para melhor entendimento:

```
(->> carros
    (map (fn [x] (* x 2)))
    (reduce (fn [acc n] (+ acc n))))
```

Perceba como o atalho `->>` é muito bem-vindo. Se ele não existisse teríamos algo assim:

```
(reduce (fn [acc n] (+ acc n)) (map (fn [x] (* x 2)) carros))
```