02

Lendo do teclado e Relacionamento entre Funções

Capítulo 4 - Lendo do teclado e Relacionamento entre funções

Daremos continuidadae à construção do nosso jogo. Vimos que as variáveis não variam e isso não é um problema pois chamamos funções dentro de outras passando novos valores. Esse método é muito bom para funções de tratamento de listas, como `remove` e `map`.

Até agora só estamos imprimindo a frase "Chuta, amigo" quando ainda há vidas e os acertos não são iguais às letras da palavra. Precisamos implementar a função que irá ler a letra digitada pelo jogador:

```
(defn le-letra! [] (read-line))
```

Usamos o ponto de exclamação quando a função muda de estado, há um efeito colateral pois o resultado será diferente de acordo com o que digitarmos no teclado. Dentro da função `jogo` será avaliado esse chute e passaremos também todos os parâmetros:

```
(defn jogo [vidas palavra acertos]
    (if (=vidas 0)
        (perdeu)
        (if (acertou-a-palavra-toda? palavra acertos)
            (ganhou)
            (avalia-chute (le-letra!) vidas palavra acertos)
```

Então precisaremos implementar o `avalia-chute`:

```
(defn avalia-chute [chute vidas palavra acertos]
    (if (acertou? chute palavra)
        (jogo vidas palavra (conj acertos chute))
        (jogo (dec vidas) palavra acertos)
    )
)
```

Ou seja, se o jogador acertar o chute, este será somado aos acertos. Se não `vidas` será decrementado. Falta implementarmos a função `acertou?`. Se estivéssemos escrevendo o código em Java, bastava fazer `palavra.contains(chute)`, mas estamos em Clojure, que por sua vez funciona na JVM, ou seja, podemos sim invocar funções do Java:

```
(defn acertou? [chute palavra] (.contains palavra chute))
```

Vejamos se nosso código funciona. No Terminal, depois de recarregar o programa, fazemos

```
forca.core=> (forca/jogo 2 "MELANCIA" #{})
```

O programa ficará esperando uma letra, afinal não passamos nenhuma para o conjunto de acertos. Se digitarmos, nesse caso de duas vidas, duas letras que não estão contidas na palavra "MELANCIA":

```
forca.core=> (forca/jogo 2 "MELANCIA" #{})
Q
W
Você perdeunil
```

Se passarmos todas as corretas:

```
forca.core=> (forca/jogo 2 "MELANCIA" #{})
M
E
L
A
N
C
I
Você ganhou!nil
```

Uma boa prática para quem está começando a aprender uma nova linguagem é escrever as execuções do programa. Faça testes de mesa para nosso jogo de forca!