02

Começando a usar funções

Capítulo 2 - Começando a usar funções

Vimos na aula passada que em programação funcional *variáveis não variam*. Se definimos o total de vidas de um jogador, e ele as vai perdendo, como isso será possível com essa mudança de paradigma? Será através de um contador. Mas vejamos algumas coisas mais simples primeiro.

No terminal, depois de termos dado o comando `lein repl`, para imprimir uma mensagem usamos a função `print`:

```
forca.core=> (print "Você perdeu")
Você perdeunil
```

O "nil" aparece porque toda invocação de função em Clojure devolve algo e, como não passamos valores, retorna vazio. Só que desse jeito o `print` não está relacionado, ainda não está em função de nenhuma instrução. Então, no código vamos transformar esse comando em função:

```
(defn perdeu [] (print "Você perdeu"))
```

O `defn` define uma função que, no caso, não recebe nada (`[]`). O *repl* precisa ser chamado de novo para que possamos observar a função no console. Mas existe um jeito mais fácil de carregar o programa novamente. Fazemos no Terminal:

```
forca.core=> (require '[forca.core :as forca] :reload)
nil
```

Este comando recarrega nosso código. Agora sim podemos chamar a função:

```
forca.core=> (forca/perdeu)
Você perdeunil
```

O que faremos agora é definir outra função:

```
(defn jogo [vidas]
    (if (= vidas 0)
        (perdeu)
        (jogo (- vidas 1))
    )
)
```

Aqui definimos quando o jogador ganha, perde ou continua o jogo. Perceba que não é o caso das variáveis variarem, mas sim de mudar os parâmetros que serão passados para as funções, nós reinvocamos a função `jogo` dentro dela mesma (caso as vidas não sejam igual a zero). Recarregamos o programa e testamos o que acabamos de escrever, sempre lembrando que, como nosso projeto se chama "forca", a função é invocada da forma `(forca/[função])`:

```
forca.core=> (forca/jogo 5)
Você perdeunil
```

Do jeito que nos apareceu, não dá para perceber as funções dentro das funções trabalhando até chegarmos em vidas = 0. Então pediremos para que o programa imprima os valores do parâmetro **`vida`** em outros momentos:

```
(defn jogo [vidas]
    (if (= vidas 0)
        (perdeu)
        (do
            (print vidas)
            (jogo (dec vidas))
    )
    )
)
```

No Terminal:

```
forca.core=> (forca/jogo 5)
54321Você perdeunil
```

Imprimiu-se todos os valores sendo decrementados (por isso podemos substituir **`(- vidas 1)`** por **`(dec vidas)`**) até chegar ao 0, o que faz imprimir a frase de derrota.