Programação Funcional com Elixir

Controlando Fluxos

Controlando Fluxos

- O Elixir, assim como outras linguagens, possui alguns controles de fluxo e é isso que veremos agora.

Controlando Fluxos

- **If**
  - Esse é o controle de fluxo mais tradicional de todos

```
iex(1)> x = 10
10
iex(2)> if x == 10 do
...(2)> "x é 10"
...(2)> end
"x é 10"
```

Controlando Fluxos

- **Unless**
  - O unless pode ser lido como “a menos que”, ou seja, ele é o contrário do If.

```
iex(1)> unless x == 10 do
...(1)> "x não é 10"
...(1)> else
...(1)> "x é 10 sim"
...(1)> end
"x é 10 sim"
```

Controlando Fluxos

- **Forma reduzida (one line)**
  - Também é permitida a forma reduzida.

```
iex(1)> if true, do: 1 + 2
3

iex(2)> if false, do: :tiago, else: :joao
:joao
```

Controlando Fluxos

- **Cond**
  - Usamos o cond quando precisamos checar diversas condições e encontrar a primeira condição válida. Podemos compará-lo ao **else if** em linguagens imperativas.

```
iex(1)> cond do
...(1)> 2 + 4 == 5 -> "Isso não é verdade"
...(1)> 2 + 3 == 6 -> "Isso também não é verdade"
...(1)> 2 + 2 == 4 -> "Ok, você acertou!"
...(1)> end
"Ok, você acertou!"
```

Controlando Fluxos

- **Cond**
  - Ainda sobre o cond, é importante saber que, se nenhuma das condições for verdadeira, um erro será levantado.

```
iex(1)> cond do
...(1)> 2 + 4 == 5 -> "Isso não é verdade"
...(1)> 2 + 3 == 6 -> "Isso também não é verdade"
...(1)> 2 + 2 == 7 -> "Você está louco!"
...(1)> end
** (CondClauseError) no cond clause evaluated to a true value
```

Controlando Fluxos

- **Case**
  - O case tenta casar um valor com vários padrões até encontrarmos um que corresponde e faça match com sucesso.

```
iex(1)> case :tobias do
...(1)> :manoel -> "Isso não casa com :tobias"
...(1)> 10 -> "Isso muito menos"
...(1)> :junior -> "Estou ficando cansado..."
...(1)> :tobias -> "Ok, você casou :tobias com :tobias!"
...(1)> end
"Ok, você casou :tobias com :tobias!"
```

Controlando Fluxos

- **Case**
  - Lembre-se que por se tratar de match, podemos usar o corinda “_” underscore

```
iex(1)> case 10 do
...(1)> 11 -> "10 não é 11"
...(1)> 12 -> "10 não é 12"
...(1)> _ -> "10 não é underscore, mas underscore é um coringa
que casa com tudo, ok?"
...(1)> end
"10 não é underscore, mas underscore é um coringa que casa com tudo,
ok?"
```

Controlando Fluxos

- **Case com cláusulas guarda**
  - Podemos também usar cláusulas guarda para o case.

```
iex(1)> case {1,2,3} do
...(1)> {1,x,3} when x > 0 -> "Isso vai casar porque 2 é maior qu
e zero"
...(1)> _ -> "Isso casaria se não tivesse casado antes"
...(1)> end
"Isso vai casar porque 2 é maior que zero"
```