



# **Programação Funcional com Elixir**



# **Tail-Call Optimization**

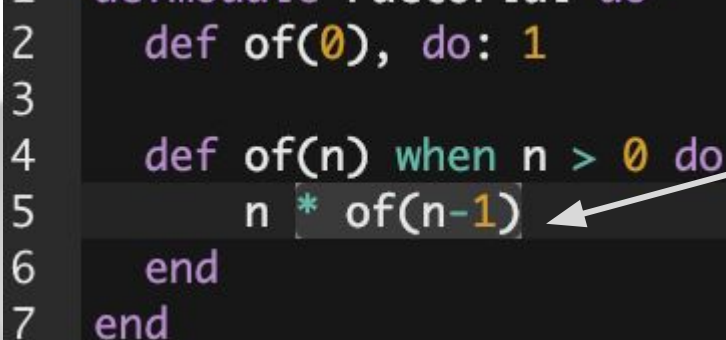
# Tail-Call Optimization

- Tail-Call Optimization é quando o compilador reduz as funções em memória sem alocar mais memória.
- Vejamos esse exemplo:

# Tail-Call Optimization

Tradicional

TC Optimization



```
1 defmodule Factorial do
2   def of(0), do: 1
3
4   def of(n) when n > 0 do
5     n * of(n-1)
6   end
7 end
```

```
1 defmodule TCFactorial do
2   def of(n), do: factorial_of(n, 1)
3
4   defp factorial_of(0, acc), do: acc
5   defp factorial_of(n, acc) when n > 0, do: factorial_of(n - 1, n * acc)
6 end
```

# Tail-Call Optimization

- Em resumo, no método tradicional, como a função em seu retorno, devolve a chamada de outra (ou mesma) função, internamente uma nova área de memória é alocada para que seja aguardado a finalização (resolução) dessa função. Já com a técnica de TC Optimization isso não ocorre.

# Tail-Call Optimization

- Prós e Contras
  - O legal da TCO é que otimizamos os recursos computacionais usados pela função.
  - A parte ruim é que geralmente uma função recursiva que se usa TCO será mais "complexa" quando se trata de entender o que de fato ela está fazendo.