



Programação Funcional: um guia para iniciantes

Conhecendo uma perspectiva diferente na criação de soluções.

Apresentação

João Vitor Ferrari da Silva

- Tecnólogo em Análise de Sistemas;
 - Especialista em Engenharia de Software com UML;
 - Mestre em Computação Aplicada com ênfase em Inteligência Computacional;
 - Engenheiro de Software @ Nubank;
- Desde 2008 aprendendo POO;
 - Desde 2022 aprendendo Programação Funcional;

LinkedIn



Paradigmas de programação



Paradigmas de programação

- **Programação Imperativa:**
 - Instruções exatas que devem ser passadas ao computador na sequência em que serão executadas;
 - Exemplos de linguagens: C e PHP.
- **Programação Orientada a Objetos (POO):**
 - Baseado no conceito de "*objetos*" e atribuição de comportamentos;
 - Modularidade do código;
 - Estados são descritos pelas classes como atributos;
 - Comportamento é definido por meio de métodos;
 - Exemplos de linguagens: PHP, Java, Ruby, C# e Python.



Programação Funcional

- Baseado no modelo computacional Cálculo Lambda (Alonzo Church, 1930);
- Tudo são funções:
 - Sequência de funções matemáticas que, juntas, vão resolver um problema;
- Exemplos de linguagens: Clojure, Haskell e Elixir.

Programação Funcional

Clojure:

- Linguagem de programação funcional;
- Interpretada e compilada para a JVM;
 - Interoperabilidade com Java;
- Criada em 2007 por Rich Hickey, com base na linguagem Lisp;

Ferramentas:

- **REPL** (Read-Eval-Print Loop): executar o código;
- **Leiningen** (Lein): ferramenta que ajuda no desenvolvimento de projetos;

```
(def hello (fn [] "Hello world"))
```

```
(hello)
```

```
=> "Hello world"
```

```
(+ 10 20)
```

```
=> 30
```



Alguns conceitos da programação funcional



Imutabilidade

- Dados não mudam após serem criados;
- Validação das informações no momento em que eles são criados;
- Aproveitar dados antigos para a construção de novos dados;
- Cenário processamento concorrente;

```
(def numbers [1 2 3 4 5])
```

```
(defn incrementa-numeros []  
  (mapv inc numbers))
```

```
(println (incrementa-numeros))  
=> [2 3 4 5 6]  
(println numbers)  
=> [1 2 3 4 5]
```

```
(def status-disponiveis '(:aberto :fechado))  
=> (:aberto :fechado)
```

```
(def status-parciais (cons :aguardando status-disponiveis))  
=> (:aguardando :aberto :fechado)
```

```
(rest status-parciais)  
=> (:aberto :fechado)
```

```
(identical? (rest status-parciais)  
            status-disponiveis)  
=> true
```




Funções Puras

- Deve retornar sempre o mesmo resultado dado os mesmos argumentos;
 - Não deve causar nenhum efeito colateral;
- Código mais fácil de entender, prever e testar;

Exemplo: Função Impura

```
(def fator 3.14)
```

```
(defn calc [num]  
  (* num fator))
```

```
(calc 3)  
=> 9.42
```

Exemplo: Função Pura

```
(defn calc [num fator]  
  (* num fator))
```

```
(calc 3 3.14)  
=> 9.42
```

```
(defn maior? [num1 num2]  
  (if (> num1 num2)  
    num1  
    num2))
```

```
(maior? 30 20)
```

```
=> 30
```

```
(maior? 10 20)
```

```
=> 20
```



First-Class e Higher-Order Functions

Função de Primeira Classe

- Funções são tratadas como qualquer outra variável;
 - Passar uma função como argumento;

Função de Grandeza Superior

- Funções que recebem funções como argumento, ou que retornam uma função como resultado;
 - `map`, `filter`, e `reduce`;

```
(def transacoes
  [{:valor 200.0M :tipo "debito" :data "21/11/2023"}
   {:valor 1200.0M :tipo "credito" :data "01/12/2023"}
   {:valor 150.0M :tipo "debito" :data "03/12/2023"}])
```

```
(defn debito? [transacao]
  (= (:tipo transacao) "debito"))
```

```
(def eh-debito debito?)
```

```
(defn soh-valor [transacao]
  (:valor transacao))
```

```
(def total-debito (reduce +
  (map soh-valor
    (filter debito? transacoes)))))
```

```
(def total-debito (-> (filter eh-debito transacoes)
  (map soh-valor)
  (reduce +)))
```

```
(println "Valor total de débitos: R$" (str total-debito))
Valor total de débitos: R$ 350.0
```



Side-effect

- Funções impuras;
 - Alteram estado;
 - Efeitos colaterais;
- Interação como o mundo externo:
 - Acessar banco de dados;
 - Realizar chamadas assíncronas,
 - Alterar valor de propriedades;
 - I/O (Entrada/Saída);
 - Impressão em Console;

```
(def contador (atom 0))
```

```
(defn incrementa-contador! []  
  (swap! contador inc)  
  (println "State:" contador))
```

```
(incrementa-contador!)  
=> State: #object[clojure.lang.Atom 0x158d255c {:status :ready,  
:val 1}]  
(println "Contador:" @contador)  
=> Contador: 1
```



Recursão

- Uma função chama a si mesma para resolver um problema menor;
- Não podemos iterar em uma coleção e modificar seu estado (por exemplo, **for**, **while**);
- `StackOverflowError...`
 - Tail Call Optimization (TCO);
 - Otimizar a alocação de dados na pilha;

```
(defn fibonacci [n]
  (cond
    (<= n 0) 0
    (= n 1) 1
    :else (+ (fibonacci (- n 1))
              (fibonacci (- n 2)))))
```

```
(println (fibonacci 3))
=> 2
(println (fibonacci 4))
=> 3
```

```
(defn print-list [lst]
  (if-let [num (first lst)]
    (do
      (println num)
      (recur (rest lst)))))
```

```
(print-list [1 2 3 4 5])
=> 1
=> 2
=> 3
=> 4
=> 5
```

Caso de uso



Loja virtual

- Pedidos de um cliente da loja virtual:
 - Precisamos contabilizar o valor total de pedidos em um período específico:

Pedido	Data do Pedido	Item	Nome do Produto	Quantidade	Valor Unitário
1000	2023-11-23	1	Produto D	10	R\$ 50,00
1001	2023-12-02	2	Produto A	10	R\$ 20,00
1001	2023-12-02	3	Produto B	5	R\$ 15,50
1002	2024-01-14	4	Produto C	8	R\$ 30,00
1002	2024-01-14	5	Produto D	2	R\$ 50,00
1002	2024-01-14	6	Produto E	15	R\$ 10,00
1003	2024-01-27	7	Produto A	15	R\$ 20,00

Programação Orientada a Objetos (POO)

```
public class Pedido
{
    public Pedido(int id, DateTime dataCadastro)
    {
        Id = id;
        DataCadastro = dataCadastro;
    }

    public int Id { get; private set; }
    public DateTime DataCadastro { get; private set; }

    public virtual ICollection<PedidoItem> Itens { get;
private set; } = new List<PedidoItem>();

    public void AdicionarItem(PedidoItem item)
    {
        Itens.Add(item);
    }
}
```

```
public class PedidoItem
{
    public PedidoItem(int id, string produto, int quantidade, decimal
valorUnitario)
    {
        Id = id;
        Produto = produto;
        Quantidade = quantidade;
        ValorUnitario = valorUnitario;
    }

    public int Id { get; private set; }
    public string Produto { get; private set; }
    public int Quantidade { get; private set; }
    public decimal ValorUnitario { get; private set; }

    public decimal ValorTotal
    {
        get { return Quantidade * ValorUnitario; }
    }
}
```

POO - Carregar os pedidos



```
var pedido1000 = new Pedido(1000, new DateTime(2023, 11, 23));  
pedido1000.AdicionarItem(new PedidoItem(1, "Produto D", 10, 50.00m));
```

```
var pedido1001 = new Pedido(1001, new DateTime(2023, 12, 2));  
pedido1001.AdicionarItem(new PedidoItem(2, "Produto A", 10, 20.00m));  
pedido1001.AdicionarItem(new PedidoItem(3, "Produto B", 5, 15.50m));
```

```
var pedido1002 = new Pedido(1002, new DateTime(2024, 1, 14));  
pedido1002.AdicionarItem(new PedidoItem(4, "Produto C", 8, 30.00m));  
pedido1002.AdicionarItem(new PedidoItem(5, "Produto D", 2, 50.00m));  
pedido1002.AdicionarItem(new PedidoItem(6, "Produto E", 15, 10.00m));
```

```
var pedido1003 = new Pedido(1003, new DateTime(2024, 1, 27));  
pedido1003.AdicionarItem(new PedidoItem(7, "Produto A", 15, 20.00m));
```

```
var pedidos = new List<Pedido> { pedido1000, pedido1001, pedido1002, pedido1003 };
```


POO: retornar valor total dos pedidos de um período

```
public decimal CalcularValorTotalPorPeriodo(List<Pedido> pedidos, DateTime dataInicial, DateTime dataFinal)
{
    decimal pedidosPeriodo = 0;
    foreach (var pedido in FiltarPedidosPorPeriodo(pedidos, dataInicial, dataFinal))
        pedidosPeriodo += CalcularValorTotalItensPedido(pedido);
    return pedidosPeriodo;
}

private List<Pedido> FiltarPedidosPorPeriodo(List<Pedido> pedidos, DateTime dataInicial, DateTime dataFinal)
{
    var pedidosFiltrados = new List<Pedido>();
    foreach (var pedido in pedidos)
    {
        if (EstaNoPeriodo(pedido.DataCadastro, dataInicial, dataFinal))
            pedidosFiltrados.Add(pedido);
    }
    return pedidosFiltrados;
}

private bool EstaNoPeriodo(DateTime data, DateTime dataInicial, DateTime dataFinal) => data >= dataInicial && data <= dataFinal;

private decimal CalcularValorTotalItensPedido(Pedido pedido)
{
    decimal valorTotal = 0;
    foreach (var item in pedido.Itens)
        valorTotal += item.ValorTotal;
    return valorTotal;
}
```

POO: resultado

```
var pedidos = new List<Pedido> { pedido1000, pedido1001, pedido1002, pedido1003 };

var dataInicial = new DateTime(year: 2023, month: 12, day: 1);
var dataFinal = new DateTime(year: 2024, month: 1, day: 20);

var pedidoService = new PedidoService();
var valorTotalPedidosPeriodo = pedidoService.CalcularValorTotalPorPeriodo(pedidos, dataInicial, dataFinal);
Console.WriteLine(value: $"Valor total dos pedidos no período: R$ {valorTotalPedidosPeriodo}");

// LINQ
var valorTotalPedidosPeriodoFp = pedidos
    .Where(p => p.DataCadastro >= dataInicial && p.DataCadastro <= dataFinal)
    .Sum(p => p.Itens.Sum(i => i.ValorTotal));
Console.WriteLine(value: $"Valor total dos pedidos no período (LINQ): R$ {valorTotalPedidosPeriodoFp}");
```



Microsoft Visual Studio Debug Console



Valor total dos pedidos no período: R\$ 767.50
Valor total dos pedidos no período (LINQ): R\$ 767.50

D:\Repos\LojaVirtual.Web\LojaVirtual.Terminal\bin\Debug\net6.0\LojaVirtual.Terminal.exe (process 24548) exited with code 0.

To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.

Press any key to close this window.

Programação Funcional

```
(require '[clj-time.format :as f])
```

```
(def custom-formatter (f/formatter "yyyy-MM-dd"))
```

```
(def pedidos [{:id      1000,
               :data-cadastro (f/parse custom-formatter "2023-11-23"),
               :itens      [{:id 1 :produto "Produto D" :quantidade 10 :valor-unitario 50.00M}]}
              {:id      1001,
               :data-cadastro (f/parse custom-formatter "2023-12-02"),
               :itens      [{:id 2 :produto "Produto A" :quantidade 10 :valor-unitario 20.00M}
                           {:id 3 :produto "Produto B" :quantidade 5 :valor-unitario 15.50M}]}
              {:id      1002,
               :data-cadastro (f/parse custom-formatter "2024-01-14"),
               :itens      [{:id 4 :produto "Produto C" :quantidade 8 :valor-unitario 30.00M}
                           {:id 5 :produto "Produto D" :quantidade 2 :valor-unitario 50.00M}
                           {:id 6 :produto "Produto E" :quantidade 15 :valor-unitario 10.00M}]}
              {:id      1003,
               :data-cadastro (f/parse custom-formatter "2024-01-27"),
               :itens      [{:id 7 :produto "Produto A" :quantidade 15 :valor-unitario 20.00M}]})
```

FP: retornar valor total dos pedidos de um período

```
(defn soma-pedido-periodo
  [pedidos
   data-inicial
   data-final]
  (let [data-inicial* (f/parse custom-formatter data-inicial)
        data-final* (time/plus (f/parse custom-formatter data-final) (time/hours 23))]
    (-> (filter #(filtro-periodo? % data-inicial* data-final*) pedidos)
        (map valor-total-pedido)
        (reduce +))))
```

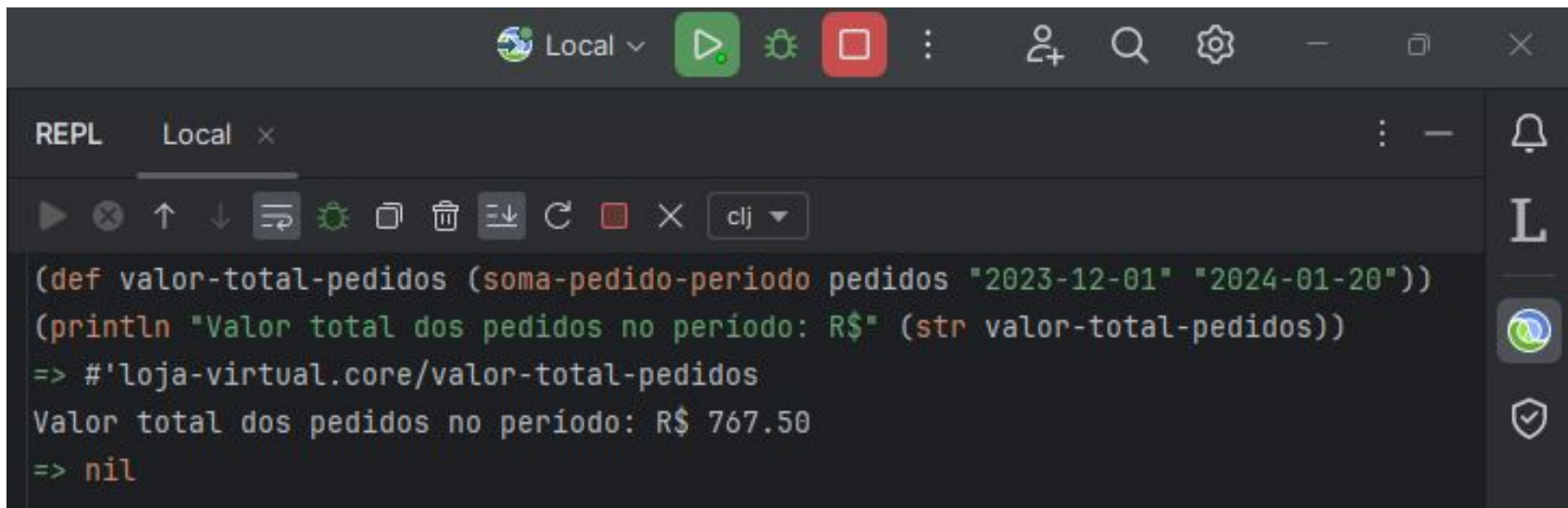
```
(defn filtro-periodo?
  [{:keys [data-cadastro]}
   data-inicial
   data-final]
  (and (time/after? data-cadastro data-inicial)
       (time/before? data-cadastro data-final)))
```

```
(defn valor-total-item-pedido
  [{:keys [quantidade valor-unitario]}]
  (* quantidade valor-unitario))
```

```
(defn valor-total-pedido
  [{:keys [itens]}]
  (-> (map #(valor-total-item-pedido %) itens)
      (reduce +)))
```

Programação Funcional

Resultado:



The screenshot shows a REPL window with a dark theme. The title bar includes a 'Local' dropdown, a play button, a gear icon, a red stop button, and standard window controls. The REPL header shows 'REPL' and 'Local' with a close button. Below the header is a toolbar with icons for running, undo, redo, and other REPL actions, followed by a 'cljs' dropdown. The main area contains the following code and output:

```
(def valor-total-pedidos (soma-pedido-periodo pedidos "2023-12-01" "2024-01-20"))  
(println "Valor total dos pedidos no período: R$" (str valor-total-pedidos))  
=> #'loja-virtual.core/valor-total-pedidos  
Valor total dos pedidos no período: R$ 767.50  
=> nil
```

On the right side of the REPL window, there is a vertical sidebar with a bell icon, a large 'L' icon, a globe icon, and a shield icon.

Conclusão



- Caso de uso: OOP vs. Programação Funcional:
 - Podemos usar mais de um **paradigma** a uma mesma **solução**;
- Princípios como: **imutabilidade** e **pureza**:
 - Complexidade acidental acarretada pela arquitetura de microsserviços;
 - Concorrência;
- **Clojure**: sintaxe concisa e expressiva;
 - Estrutura de dados;
- Previsibilidade do sistema:
 - Reuso;
 - Escalabilidade;
 - Testabilidade;
 - Manutenção;

Referências



Programação Funcional:

- <https://www.alura.com.br/artigos/programacao-funcional-o-que-e>
- <https://building.nubank.com.br/pt-br/o-que-e-programacao-funcional-e-como-usamos-esta-tecnologia-no-nubank/>
- <https://blog.geekhunter.com.br/quais-sao-os-paradigmas-de-programacao/>
- <https://www.geeksforgeeks.org/functional-programming-paradigm/>

Clojure:

- Livros
 - <https://www.braveclojure.com/clojure-for-the-brave-and-true/>
 - <https://www.casadocodigo.com.br/products/livro-programacao-funcional-clojure? pos=2& sid=48d4327f1& ss=r>
 - <https://www.amazon.com.br/Joy-Clojure-Michael-Fogus/dp/1617291412>
- Curso
 - <https://www.alura.com.br/curso-online-clojure-introducao-a-programacao-funcional>
 - <https://4clojure.oxal.org/> (Aprender Clojure)
 - https://arielortiz.info/apps/s201911/tc2006/notes_setting_up_clojure/ (Configurar)
 - <https://mishadoff.com/blog/clojure-design-patterns/>
 - <https://stuartsierra.com/tag/dos-and-donts> (Recomendações)