

Clojure

PARADIGMAS DE LINGUAGENS DE PROGRAMAÇÃO

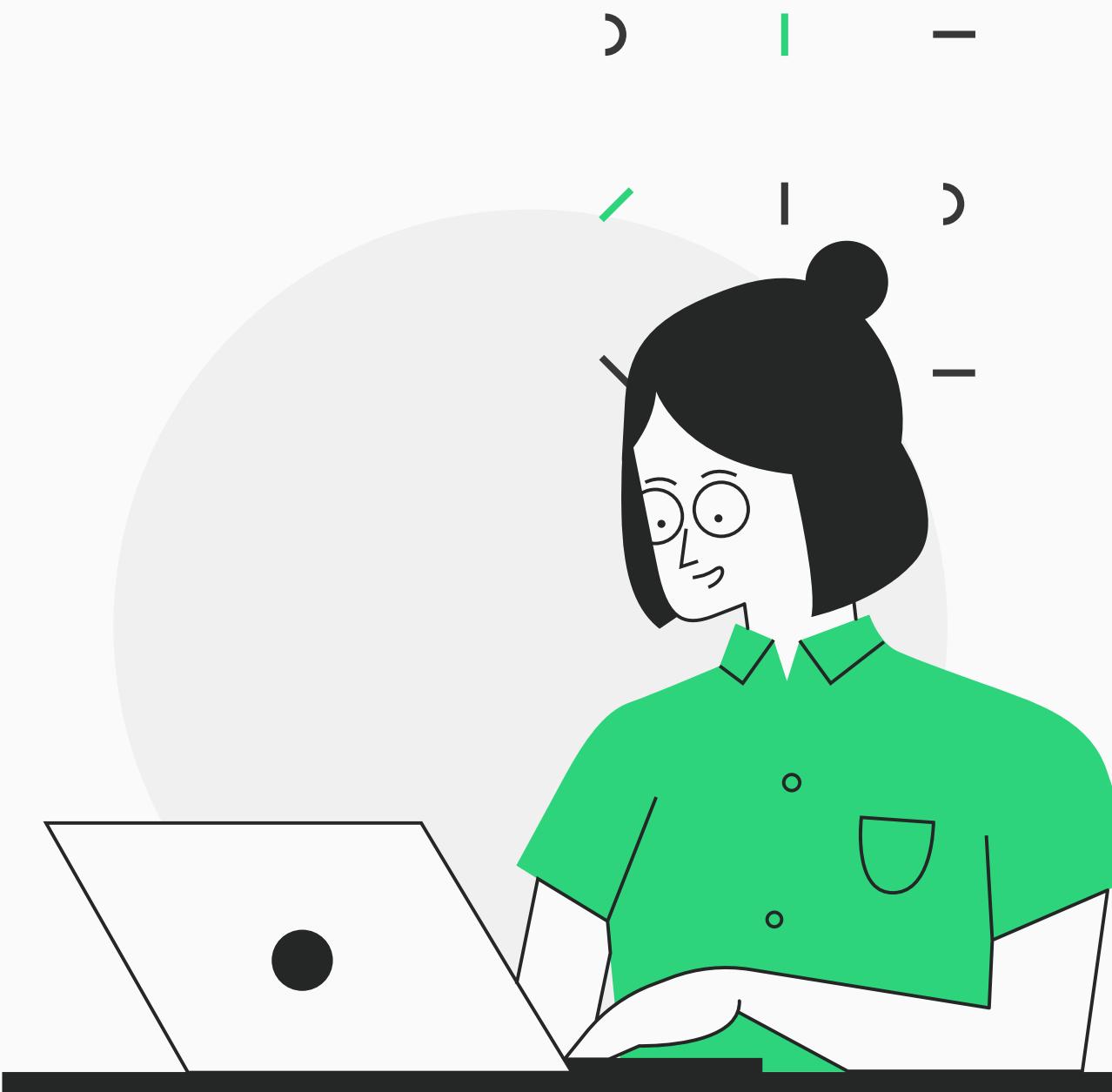


Equipe

- Beatriz Cerqueira
- Bruno de Lucas
- Elis Alcântara
- Lávio do Vale
- Lucas Morais
- Monique Silva

Paradigma funcional

Paradigma funcional



CONCEITO

A primeira base fundamental da **programação funcional** é a **composição de funções**. Ela determina que é possível criar uma nova função através da composição de outras.

As funções também podem ser passadas como **parâmetro** e **valores** para outras e funções e ter o resultado armazenado em uma **constante**.

Estilo de programação **declarativo**, no qual os programas descrevem os resultados desejados sem listar explicitamente os comandos ou etapas que devem ser executados.



Paradigma funcional

IMUTABILIDADE

Uma vez que um objeto ou variável sejam instanciados, não podem ser modificados. As funções não dependem de dados globais, e os acessam através de parâmetros.

FUNÇÕES PURAS

O resultado não depende dos valores anteriores da função. Apresenta sempre o mesmo resultado, sem efeitos colaterais, ou seja, não altera o estado da aplicação

GERENCIAMENTO DE MEMÓRIA

Diferente de algumas linguagens imperativas, como C, não tem alocação explícita de memória ou de variáveis.



Introdução ao Clojure



Introdução ao Clojure

Clojure é uma linguagem de programação funcional que oferece abstrações para lidar com concorrência, manipulação de dados e programação paralela, criada por Rich Hickey em 2007.

Clojure pode ser compilado para ser executado sobre a JVM (Máquina virtual do Java), mas temos a possibilidade de compilar para Javascript usando o ClojureScript e ClojureCLR que compila para a plataforma .NET.



Rich Hickey

Criador do Clojure

Introdução ao Clojure

A principal inspiração para o Clojure veio da linguagem Lisp, sendo considerado um dialeto de Lisp, pois compartilha muitas das características e princípios fundamentais dessa linguagem.

O objetivo de Hickey foi combinar as características da programação funcional com a eficiência e robustez da plataforma JVM.

Embora tenha sido inspirado pelo Lisp, o Clojure também adicionou suas próprias características e abordagens.



Rich Hickey

Criador do Clojure

Características

AMBIENTE DE EXECUÇÃO

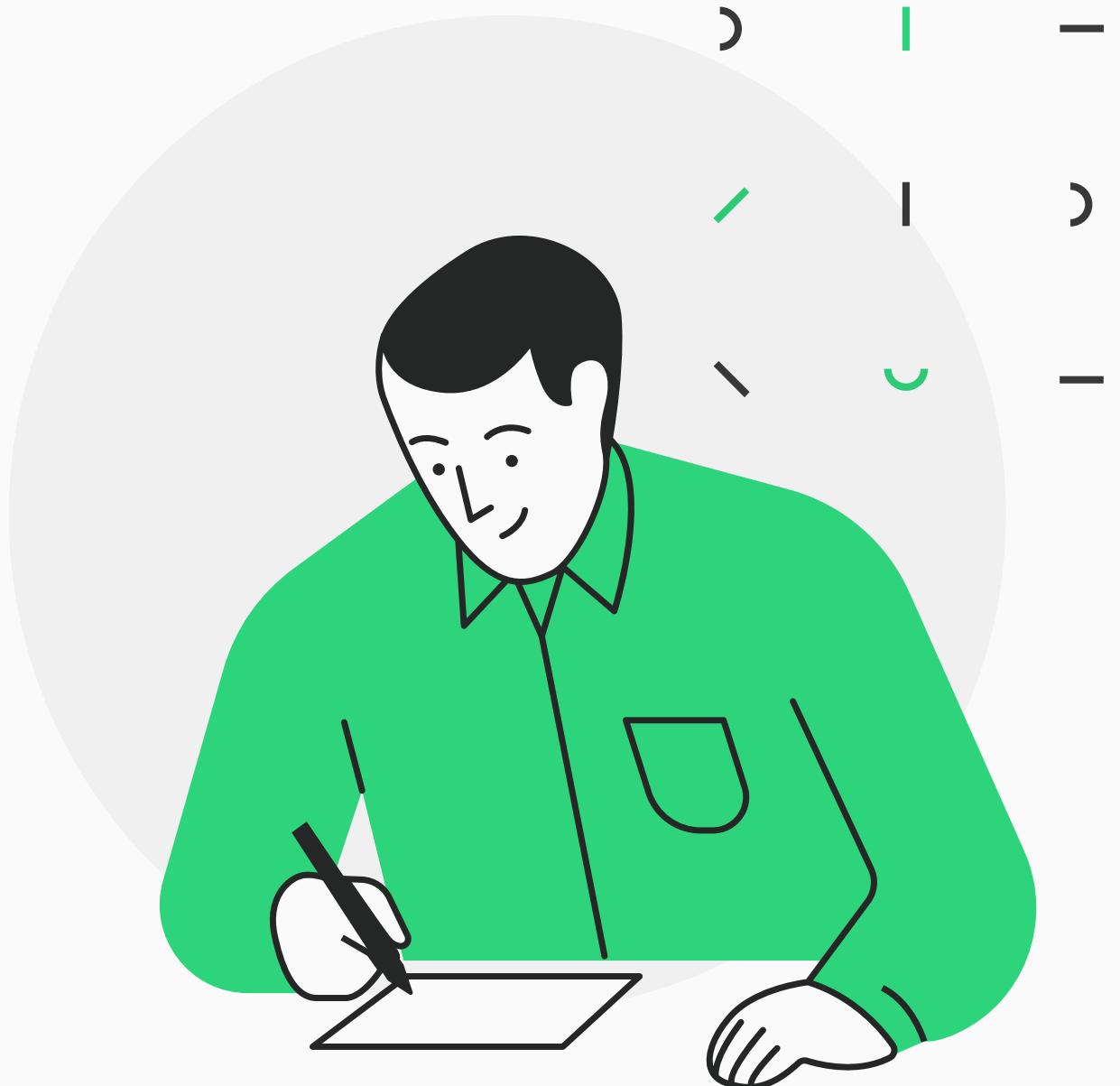
Pode ser executado na Máquina Virtual do Java (JVM), em navegadores da web e na plataforma .NET, dependendo do ambiente ou objetivo para a qual ele foi compilado.

SINTAXE

A sintaxe é baseada em parênteses como no Lisp, mas os parênteses podem chamar funções e também criar listas e outras estruturas de dados.

TIPAGEM

O Clojure é uma linguagem dinamicamente tipada, o que significa que as variáveis não têm tipos fixos e podem ser reatribuídas a diferentes tipos de dados



Características

IMUTABILIDADE DOS DADOS

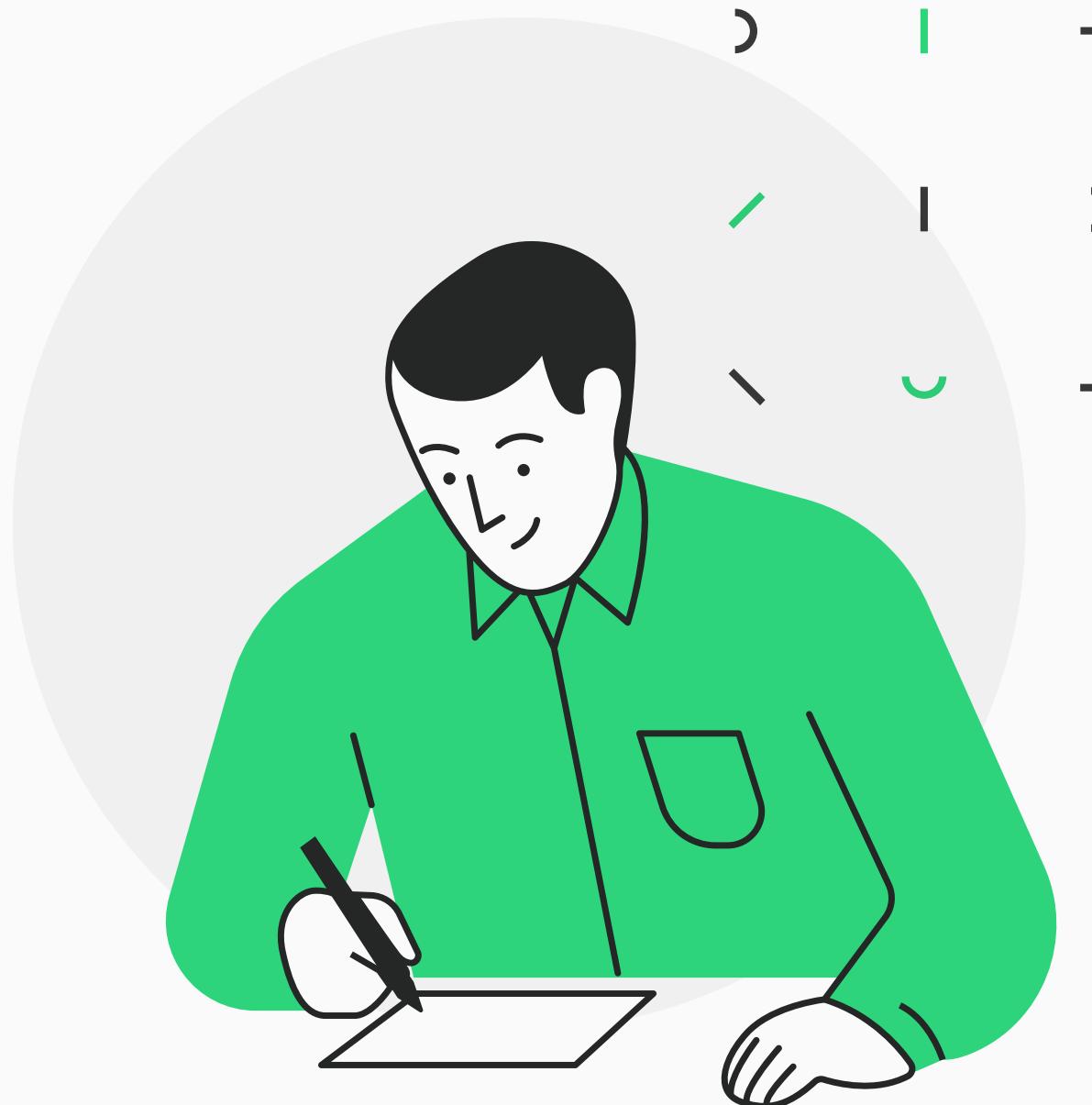
O Clojure incentiva a criação de novas estruturas de dados em vez de modificar as existentes, garantindo a ausência de efeitos colaterais indesejados.

PROGRAMAÇÃO FUNCIONAL PURA

As funções são tratadas como valores de primeira classe e podem ser passadas como argumentos para outras funções, retornadas como resultados ou armazenadas em estruturas de dados.

INTEROPERABILIDADE COM JAVA

O Clojure foi projetado para ser altamente interoperável com o ecossistema Java. Você pode usar diretamente bibliotecas Java existentes e chamar código Java a partir do Clojure

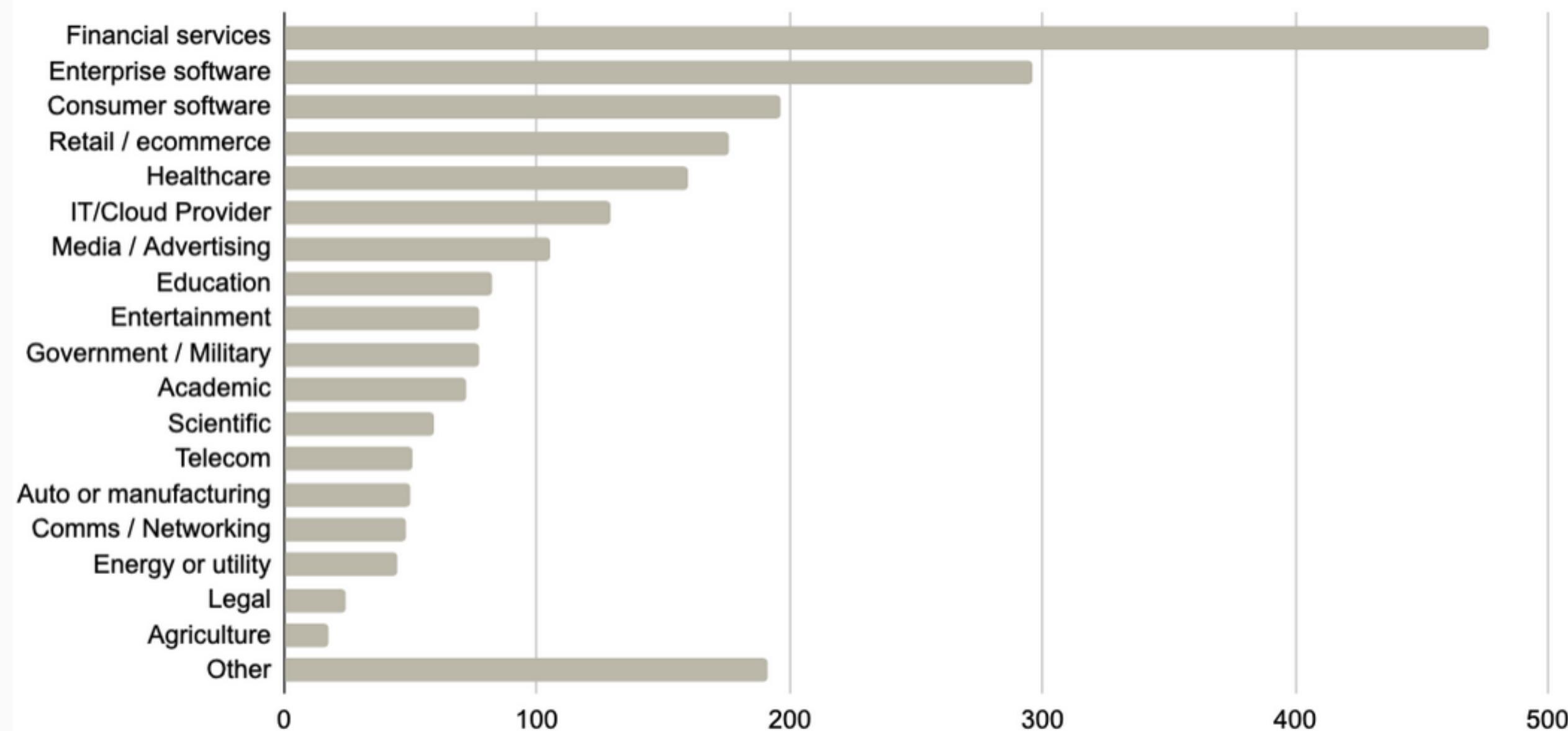


Linha do tempo

2005	2007	2009	2012	2023-06-27
Rich Hickey começa a trabalhar no Clojure	Clojure 1.0 é lançado Estruturas de dados imutáveis Sintaxe Lisp Programação funcional REPL	Clojure 1.2 é lançado Java interop Multithreading Ferramentas de debug	Clojure 1.2 é lançado Inferência de tipo Melhorias macro Novas bibliotecas	Clojure 1.16 é lançado Compatibilidade Java 17 Melhorias do compilador Novas bibliotecas

Desde o seu lançamento, o Clojure ganhou uma base de usuários dedicada e uma comunidade ativa.

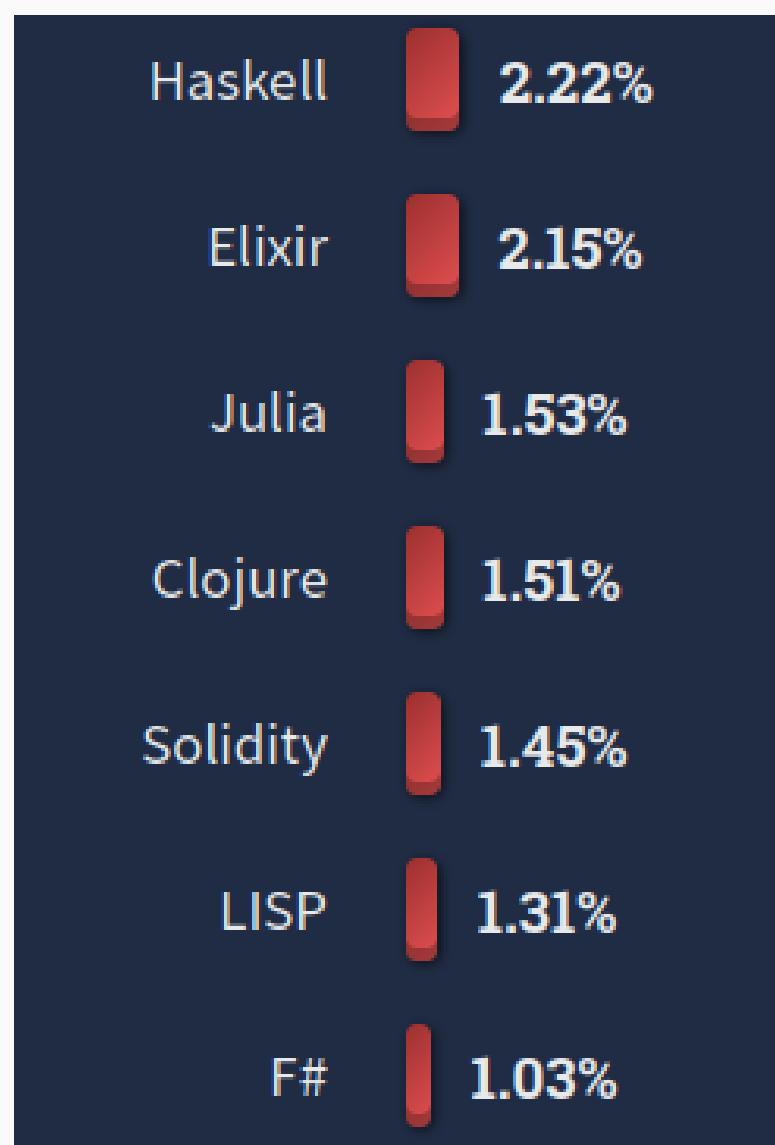
Clojure tem sido usado por várias empresas e projetos, desde startups até grandes corporações, em áreas como web, ciência de dados e programação concorrente.



Survey 2022

<https://survey.stackoverflow.co/2022/#technology-most-popular-technologies>

Tecnologias mais populares



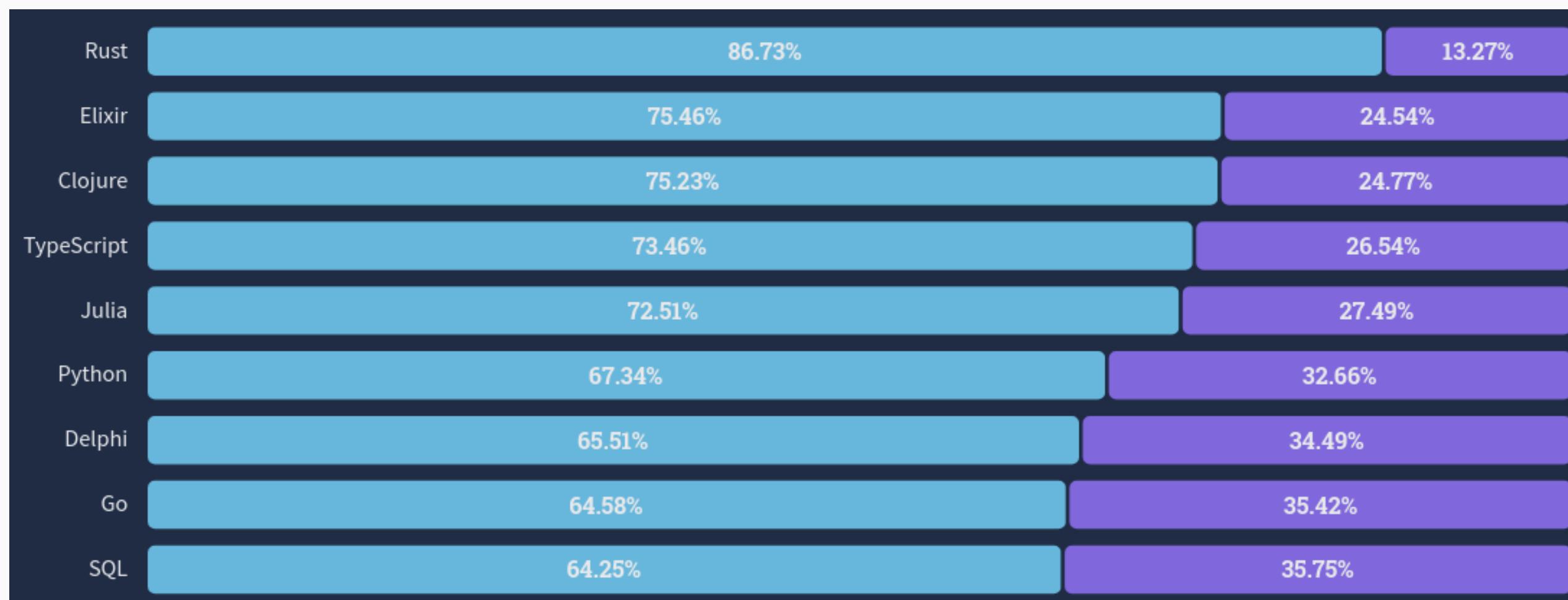
32ª posição

71.547 respostas

Survey 2022

<https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted>

Mais amado, mais temido



3^a posição
75.23% amado, 24.77% temido

71.467 respostas

Survey 2022

<https://survey.stackoverflow.co/2022/#section-top-paying-technologies-top-paying-technologies>

Tecnologias mais bem pagas



1^a posição

37.960 respostas



Sintaxe e estrutura, Java interop

Sintaxe e estrutura



POR SER BASEADO NO LISP, É BASEADO NA SINTAXE COM PARENTESES
E TUDO É UMA LISTA!

```
5  (defn salve [] "Salve!")
6  (println(salve))
```

TIPAGEM DINÂMICA

```
1  (defn nomeIdade [nome idade] (str "Meu nome e: " nome "Minha idade e: " idade))
2
3  (println(nomeIdade "Bruno" 23))
```

```
62
63  (for [i (range 10)] (println "nao aceito fiado, volte amanca " i))
```

Sintaxe e estrutura

```
15  (defn maiorQueZero [num] (if (> num 0) "verdadeiro" "falso"))
16  (defn maiorQueZero [num] (> num 0))
17  (println maiorQueZero 2)
```

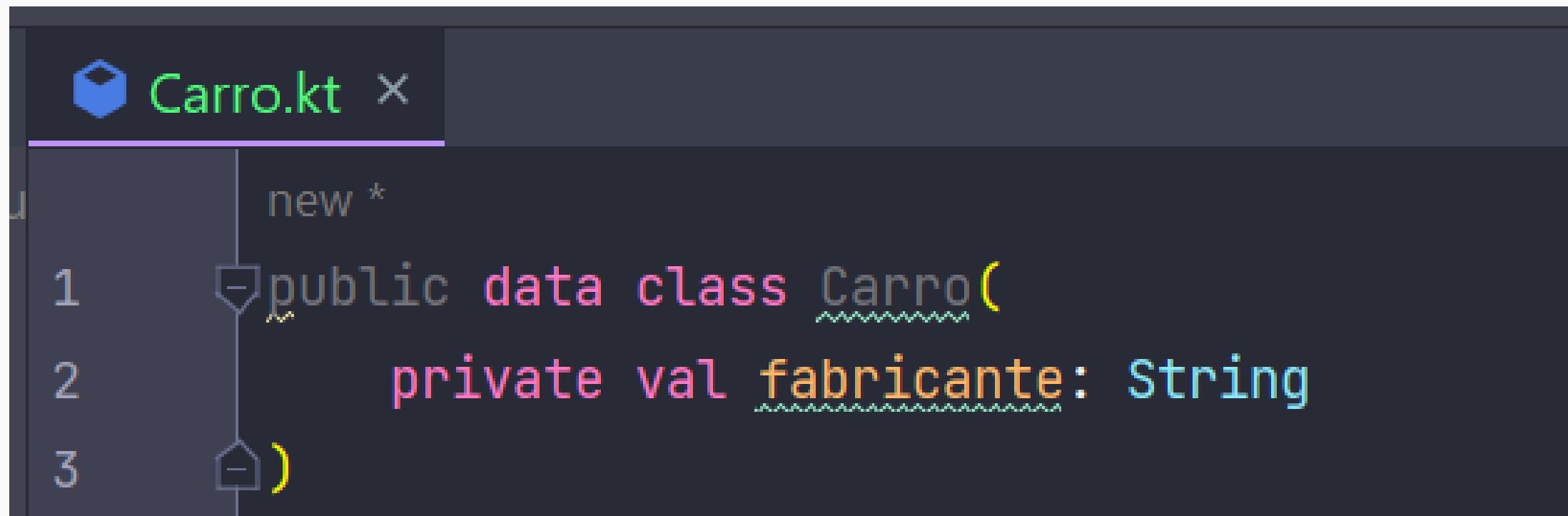
```
19  (def produtos [
20    {:nome "Uva" :quantidade 5}
21    {:nome "Banana" :quantidade 2}
22    {:nome "Laranja" :quantidade 3}
23    {:nome "Melancia" :quantidade 10}
24    {:nome "Aipim" :quantidade 0}
25    {:nome "Abacaxi" :quantidade 0}
26    {:nome "Pera" :quantidade 20}
27  ])
28  (defn estaDisponivel [produto] (> (:quantidade produto) 0))
29  (defn produtosDisponiveis [produto] (filter estaDisponivel produtos))
30  ;; OU
31  (filter estaDisponivel produtos)
32  ;; ({:nome "Uva", :quantidade 5} {:nome "Banana", :quantidade 2} {:nome "Laranja", :quantidade 3} {:nome "Melancia", :quantidade 10} {:nome "Pera", :quantidade 20})
```

Java interop

```
project.clj
1  (defproject java-interop "0.1.0-SNAPSHOT"
2    :description "FIXME: write description"
3    :url "http://example.com/FIXME"
4    :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception-2.0"
5              :url "https://www.eclipse.org/legal/epl-2.0/"}
6    :dependencies [[org.clojure/clojure "1.10.1"]
7                  [com.google.code.gson/gson "2.8.6"]]
8    :java-source-paths ["java_src"]
9    :main ^:skip-aot java-interop.core
10   :target-path "target/%s"
11   :profiles {:uberjar {:aot :all
12                         :jvm-opts ["-Dclojure.compiler.direct-linking=true"]}})
```

```
java_src > Car.java > ...
1  package carro;      Car.java is a non-p...
2
3  import com.google.gson.Gson;
4
5  public class Carro {
6
7    private String fabricante;
8
9    public Car(String fabricante) {
10      this.fabricante = fabricante;
11    }
12
13    public String getFabricante() {
14      return this.fabricante;
15    }
16
17    public String toJson() {
18      Gson gson = new Gson();
19      return gson.toJson(this);
20    }
21
22}
```

Java interop



```
1 package carro;    Car.java is a non-project +  
2  
3 import com.google.gson.Gson;  
4  
5 public class Carro {  
6  
7     private String fabricante;  
8  
9     public Car(String fabricante) {  
10         this.fabricante = fabricante;  
11     }  
12  
13     public getFabricante() {  
14         return this.fabricante;  
15     }  
16  
17     public setFabricante(String fabricante) {  
18         this.fabricante = fabricante;  
19     }  
20  
21     // public String toJson() {  
22     //     Gson gson = new Gson();  
23     //     return gson.toJson(this);  
24     // }  
25 }
```

Java interop

```
new *  
1  public class Usuario(  
2      private var firstName: String,  
3      private val lastName: String,  
4      private val email: String,  
5      private val password: String,  
6      private val phone: String,  
7  )
```

```
1  public class Usuario {  
2      private String firstName;  
3      private String lastName;  
4      private String email;  
5      private String password;  
6      private String phone;  
7      new *  
8      public Usuario(String firstName, String lastName, String email, String password, String phone) {  
9          this.firstName = firstName;  
10         this.lastName = lastName;  
11         this.email = email;  
12         this.password = password;  
13         this.phone = phone;  
14     }  
15     new *  
16     public String getFirstName() {  
17         return firstName;  
18     }  
19     new *  
20     public void setFirstName(String firstName) {  
21         this.firstName = firstName;  
22     }  
23     new *  
24     public String getLastname() {  
25         return lastName;  
26     }  
27     new *  
28     public void setLastName(String lastName) {  
29         this.lastName = lastName;  
30     }  
31     new *  
32     public String getEmail() {  
33         return email;  
34     }
```

Java interop

```
1  (ns java-interop.core
2    (:gen-class)
3    (:import (car Car)))
4
5
6  (.toJson (new Car "Toyota"))
7
8  (.getFabricante (new Car "Toyota")))
9
```

```
java_src > Car.java > ...
1  package carro;    Car.java is a non-p:
2
3  import com.google.gson.Gson;
4
5  public class Carro {
6
7    private String fabricante;
8
9    public Car(String fabricante) {
10      this.fabricante = fabricante;
11    }
12
13    public String getFabricante() {
14      return this.fabricante;
15    }
16
17    public String toJson() {
18      Gson gson = new Gson();
19      return gson.toJson(this);
20    }
21
22}
```

Java interop

```
36  (let [d (java.util.Date.)]
37    (. d getTime)) ; => 1349819873183
38
39  (doto (java.util.Stack.)
40    (.push 42)
41    (.push 13)
42    (.push 7)) ; => [42, 13, 7]>
43
```

A MELHOR PARTE ?

```
(defroutes app-routes
  (context "/documents" [] (defroutes documents-routes
    (GET "/" [] (get-all-documents))
    (POST "/" {body :body} (create-new-document body)))
    (context "/:id" [id] (defroutes document-routes
      (GET "/" [] (get-document id))
      (PUT "/" {body :body} (update-document id body))
      (DELETE "/" [] (delete-document id))))))
  (route/not-found "Not Found")))
```

Imutabilidade, estruturas de dados persistentes, concorrência e paralelismo

Estruturas de dados persistentes

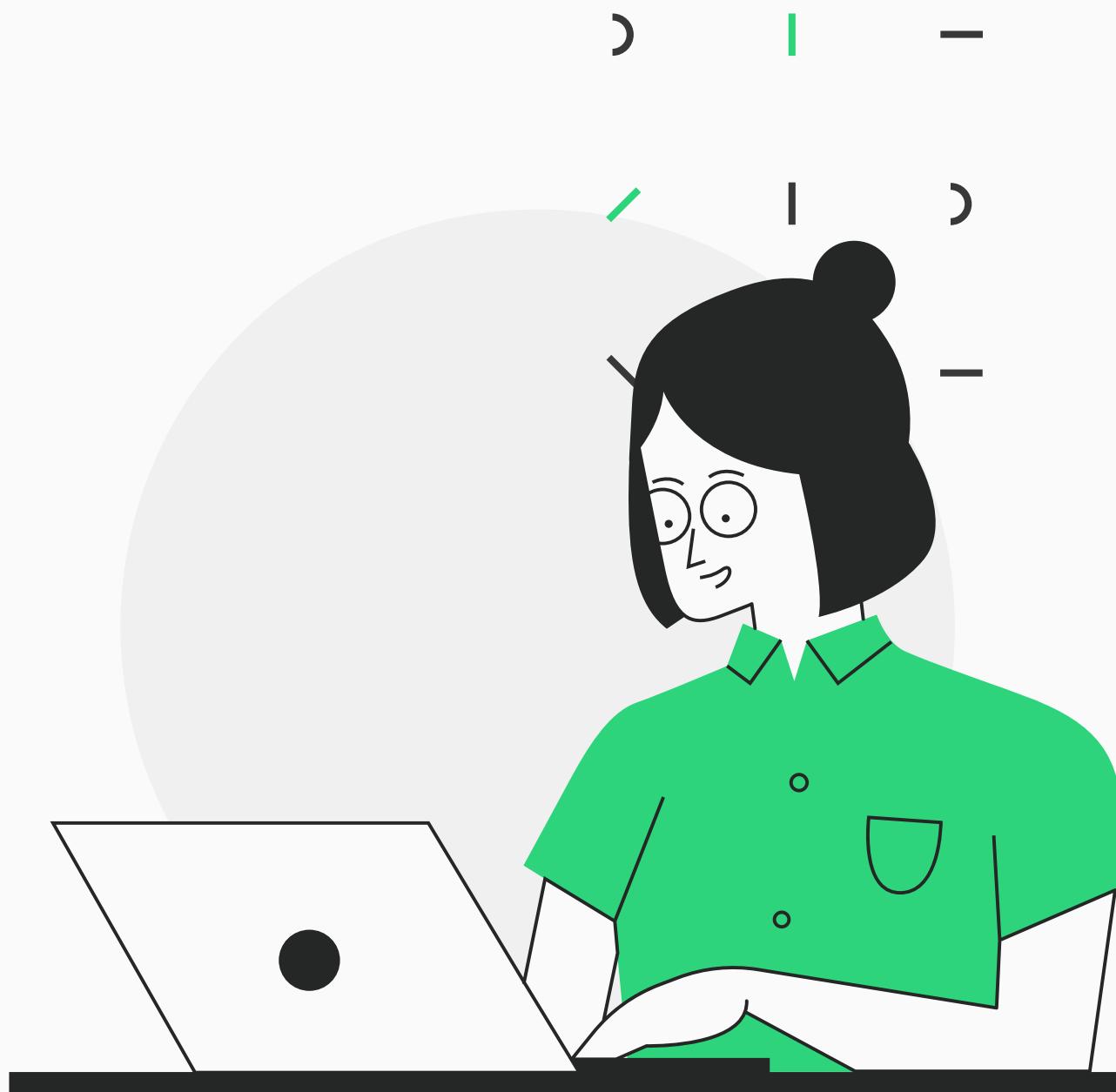
```
(def a [1 2 3])

(defn dobro [x]
  (* x 2))

(defn triplo [x]
  (* x 3))

(prinln a) ; (1 2 3)
(prinln (map dobro a)) ; (2 4 6)
(prinln (map triplo a)) ; (3 6 9)
(prinln a) ; (1 2 3)
```

Concorrência e paralelismo



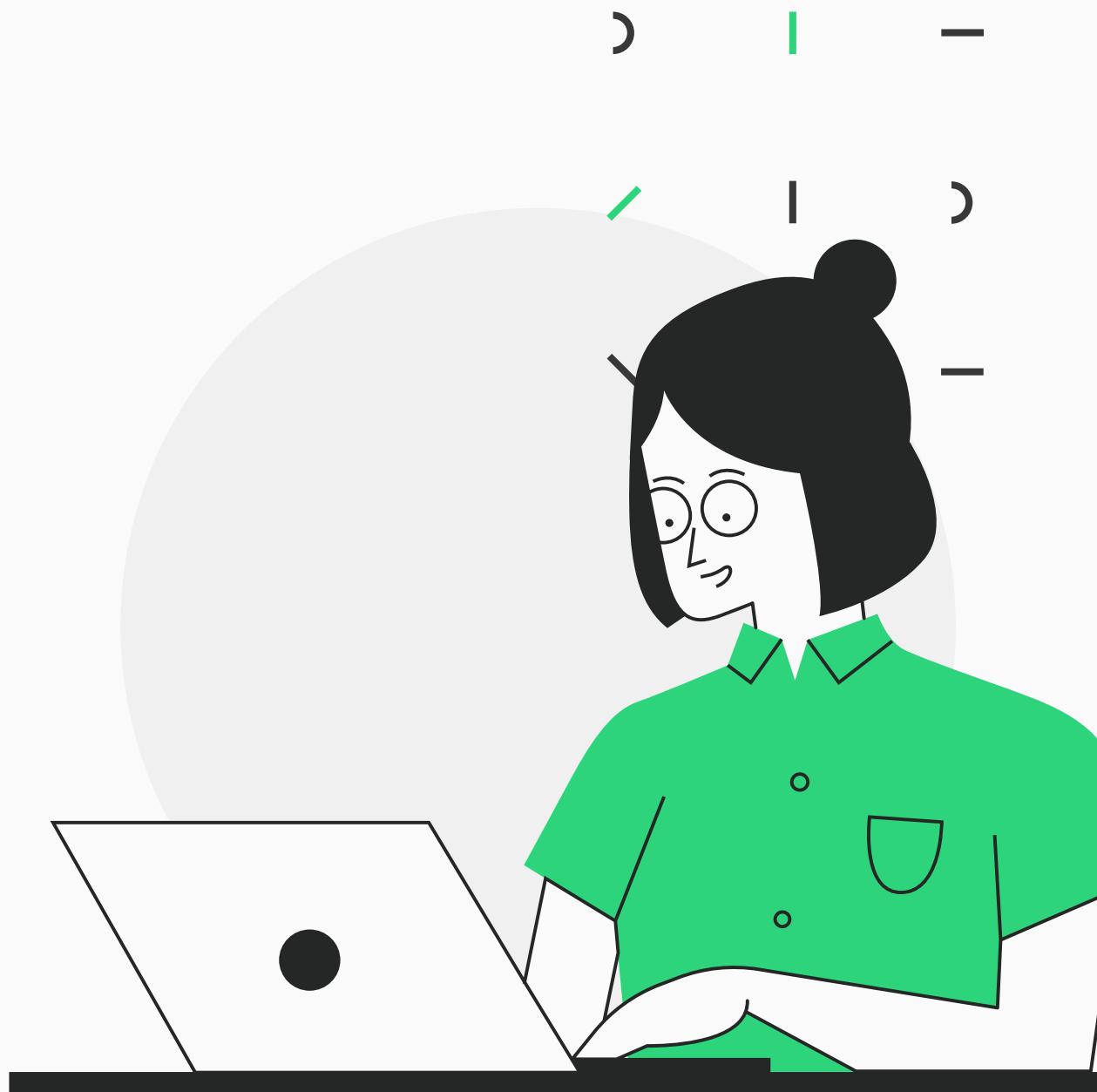
EXEMPLO

Situação inicial:
Saldo=R\$10,00

Recebeu uma transferência de 5 reias:
Saldo = 15,00

Fez uma transferência de 2 reias:
Saldo = 13,00

Concorrência e paralelismo



**EXISTEM VÁRIAS FORMAS DE
RESOLVER O PROBLEMA!**

Otimização das threads do Java

```
(def a [1 2 3])

(defn dobro [x]
  (* x 2))

(defn triplo [x]
  (* x 3))

(defn quadruplo [x]
  (* x 4))

(->> a
  (map dobro)
  (map triplo)
  (map quadruplo))
```

```
int dobro (int x) {
    return x * 2;
}

int triplo (int x) {
    return x * 3;
}

int quadruplo (int x) {
    return x * 4;
}

int main() {
    int a[3] = {1, 2, 3};

    for (int i=0; i<3; i++) {
        a[i] = quadruplo(triplo(dobro(a[i])));
    }
}
```

Alternativa: Atoms

```
(def my-atom (atom 0))
(println @my-atom) ;; 0

(swap! my-atom inc)
(println @my-atom) ;; 1

(swap! my-atom (fn [n] (* (+ n n) 2)))
(println @my-atom) ;; 4

(reset! my-atom 0)
(println @my-atom) ;; 0
```

Função pura X Função não pura

```
(defn soma [a b]
  (+ a b))

(println (soma 2 3)) ; 5
(println (soma 2 3)) ; 5
println (soma 2 3)) ; 5
```

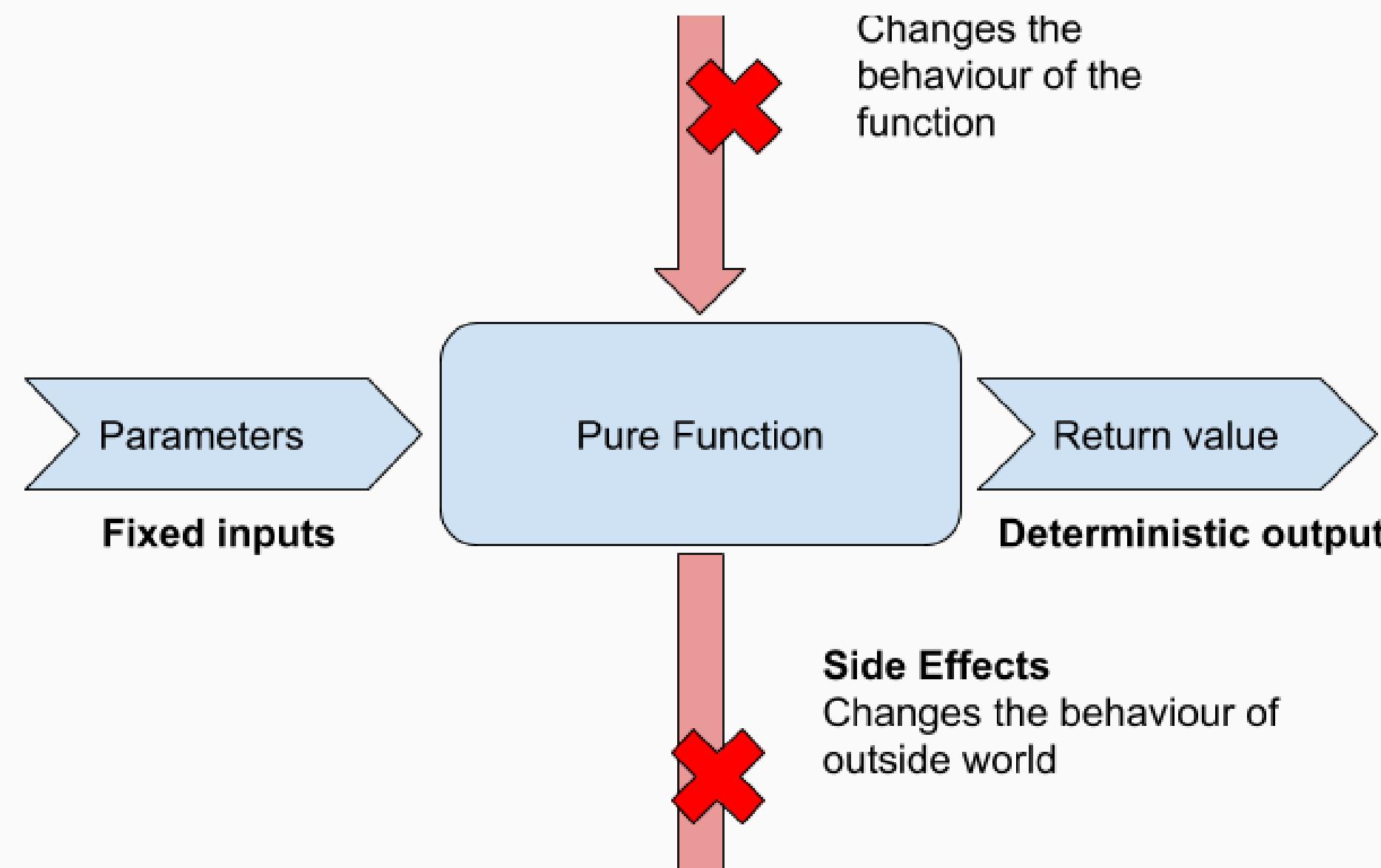
```
class Main {

    public static int valor = 0;

    public static int soma (int a) {
        valor += a;
        return valor;
    }

    public static void main(String[] args) {
        System.out.println(soma(3)); // 3
        System.out.println(soma(3)); // 6
        System.out.println(soma(3)); // 9
    }
}
```

Funções puras



Vantagens

- OTIMIZAÇÃO DAS THREADS
 - MAIS PERFORMANCE
 - MAIOR LEGIBILIDADE
- FACILITA ABSTRAÇÕES: "CAIXA PRETA"
- ALTAMENTE TESTÁVEL
- MUITA SEGURANÇA COM RELAÇÃO AO COMPORTAMENTO DO CÓDIGO
- ISOLAMENTO DA LÓGICA DE NEGÓCIO

Isolamento da lógica de negócio

```
;; Lógica de negócio

(defn dobro [x]
  (* x 2))

(defn triplo [x]
  (* x 3))

(defn quadruplo [x]
  (* x 4))
```

```
;; Interações com o mundo externo

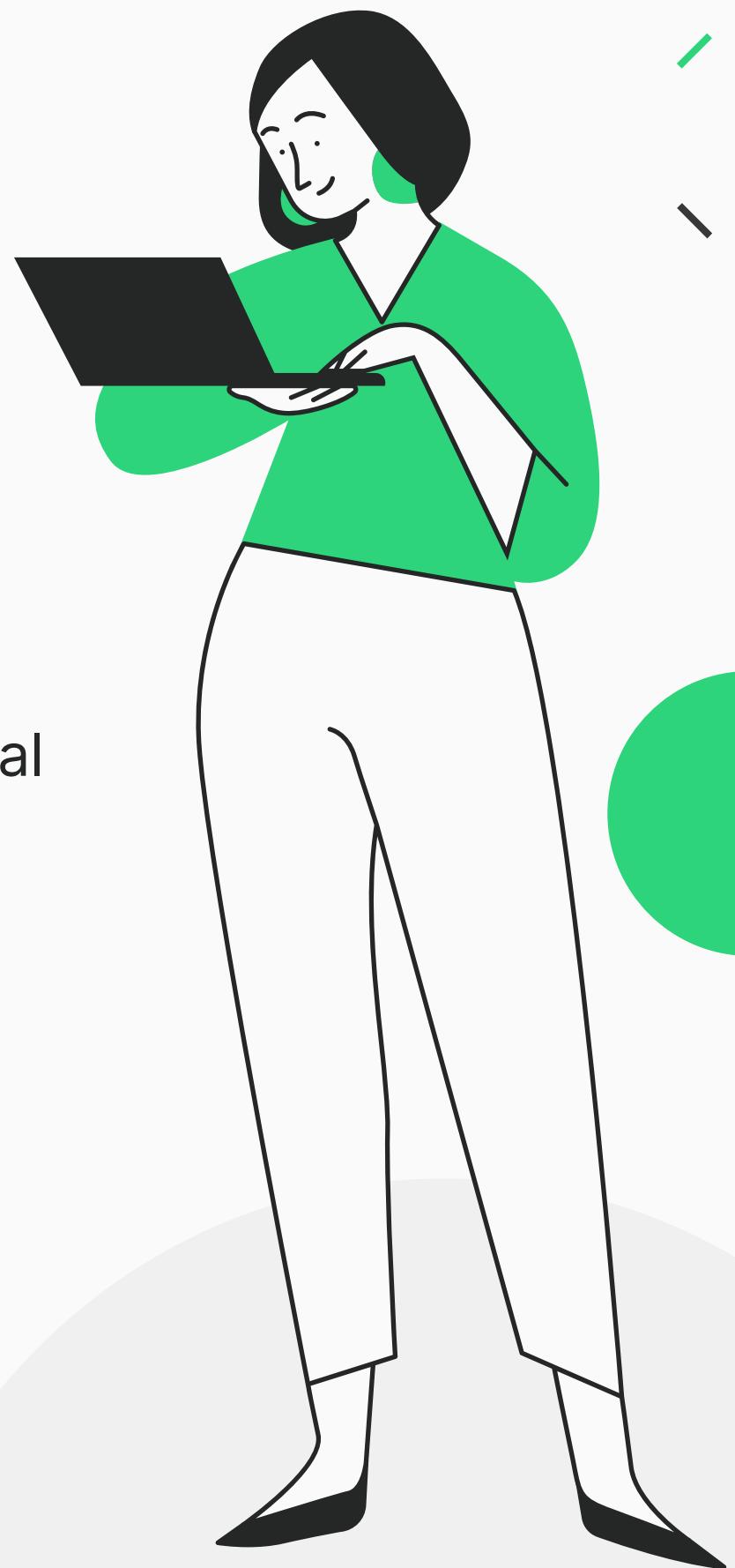
(defn salvar-no-banco-de-dados! []
  ; do something
  )

(defn fazer-requisicao-http! []
  ; do something
  )
```

Macros e metaprogramação

Macros e metaprogramação

Em Clojure, a **metaprogramação** é um recurso poderoso que permite manipular o código-fonte do programa em tempo de execução. A principal ferramenta para a metaprogramação em Clojure são os **macros**.

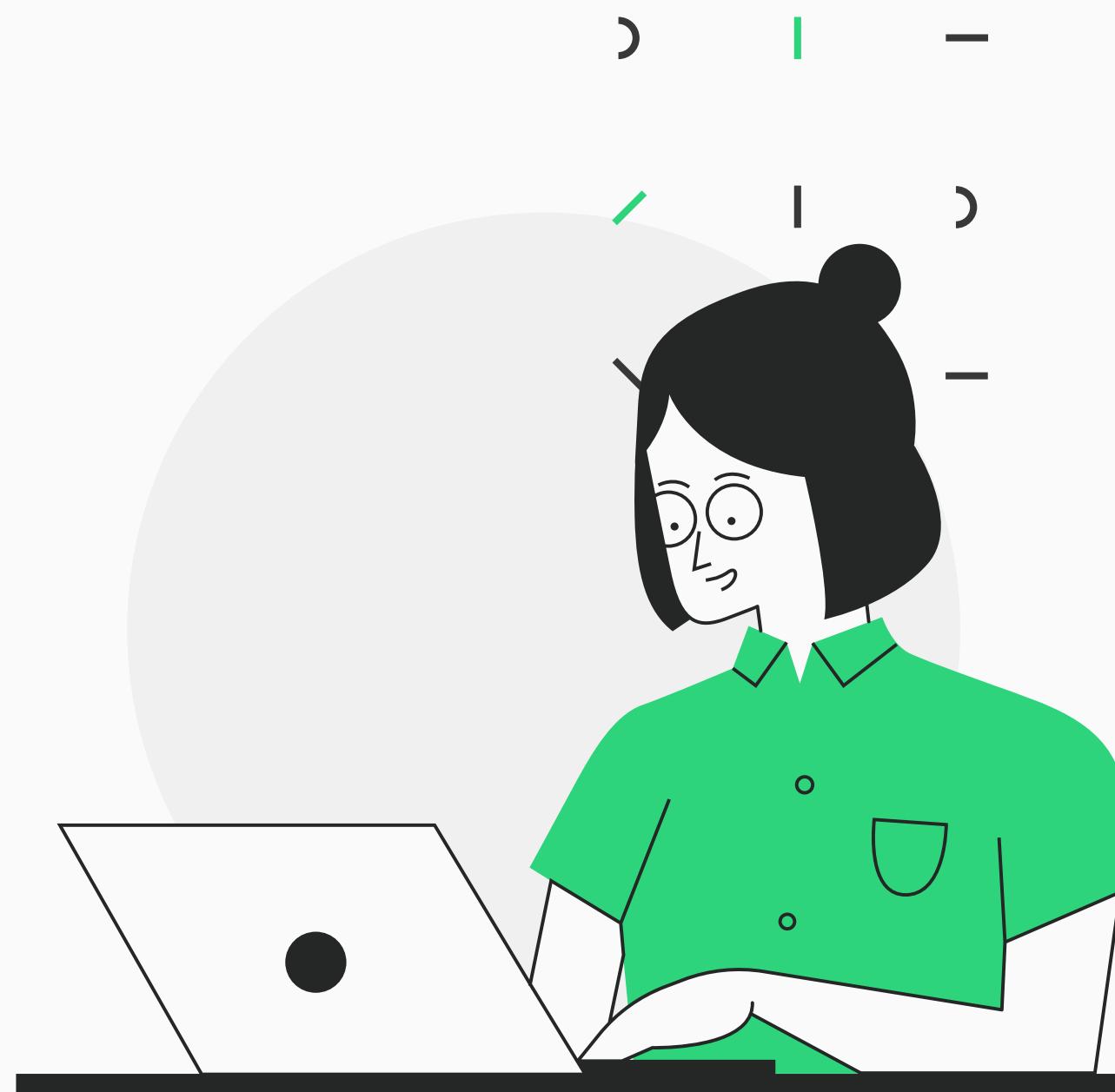


Macros e metaprogramação

```
1 (defmacro dobra [valor]
  `(* 2 ~valor))
3
4 (let [x 5]
5  (println (dobra x)))
```

Ferramentas e ecossistema

Ferramentas e ecossistema



LEININGEN

É um sistema de construção e gerenciamento de dependências para projetos Clojure. Ele facilita a criação de novos projetos, a inclusão de dependências e a execução de tarefas comuns, como compilar, testar e rodar o código.

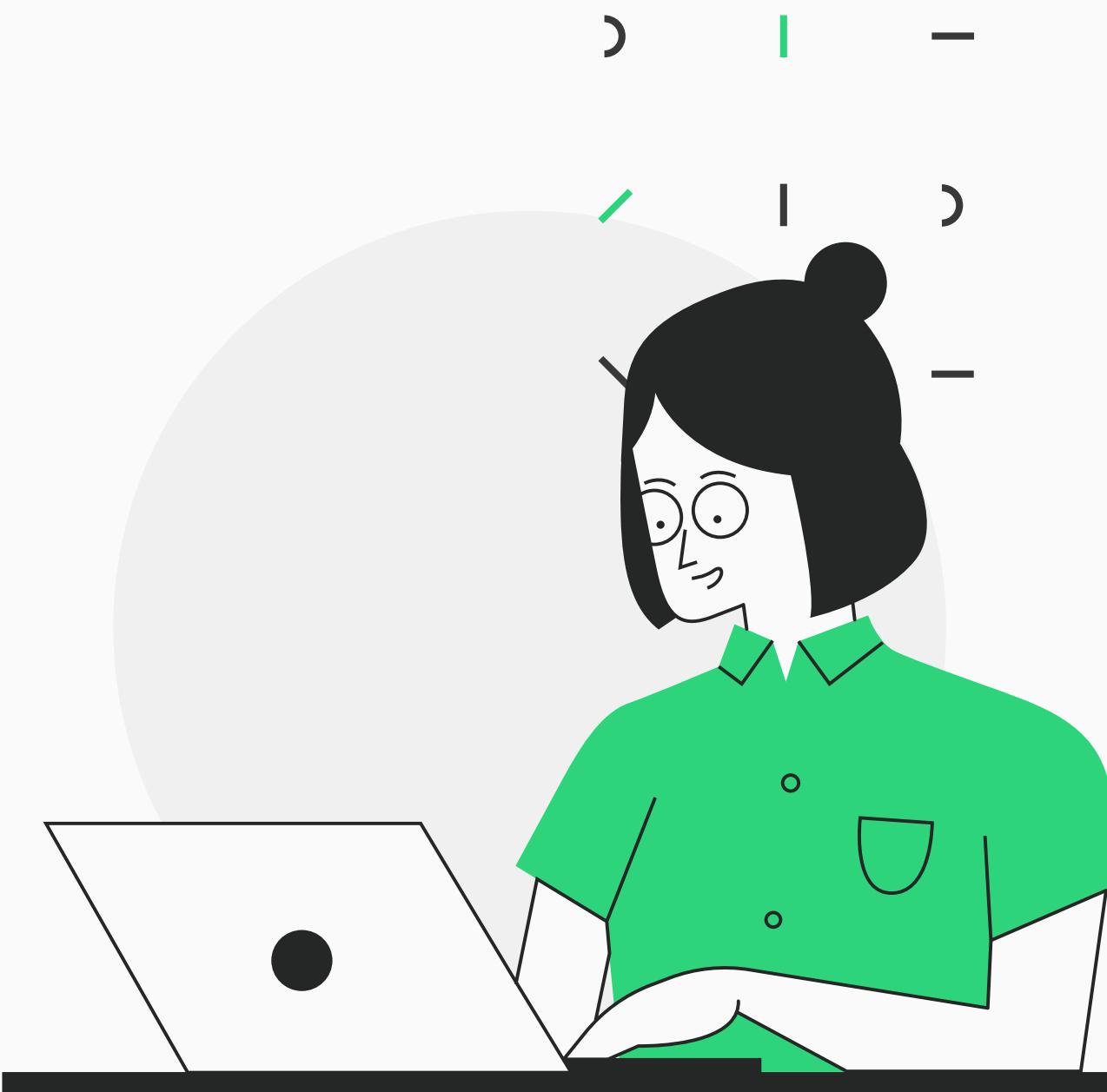
CLOJURESCRIPT

É um dialeto do Clojure que compila para JavaScript. Ele permite que você escreva aplicativos front-end em Clojure que podem ser executados nos navegadores.

CLOJURECLR

É uma implementação do Clojure para o ambiente Common Language Runtime (CLR) do .NET Framework. Ele permite que você escreva código Clojure que pode ser executado em plataformas .NET.

Ferramentas e ecossistema



BABASHKA

Babashka é um interpretador nativo de Clojure para scripts. Seu principal objetivo é permitir utilizar o Clojure em locais onde você normalmente usaria o bash.

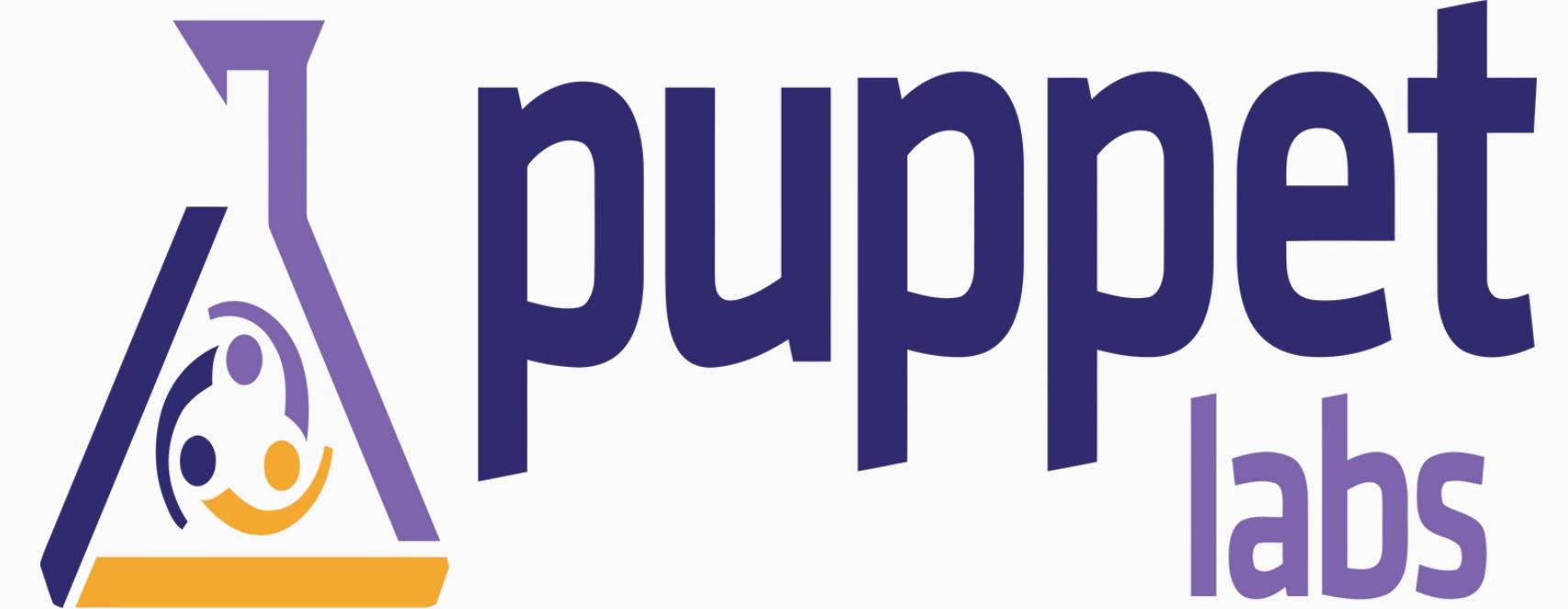
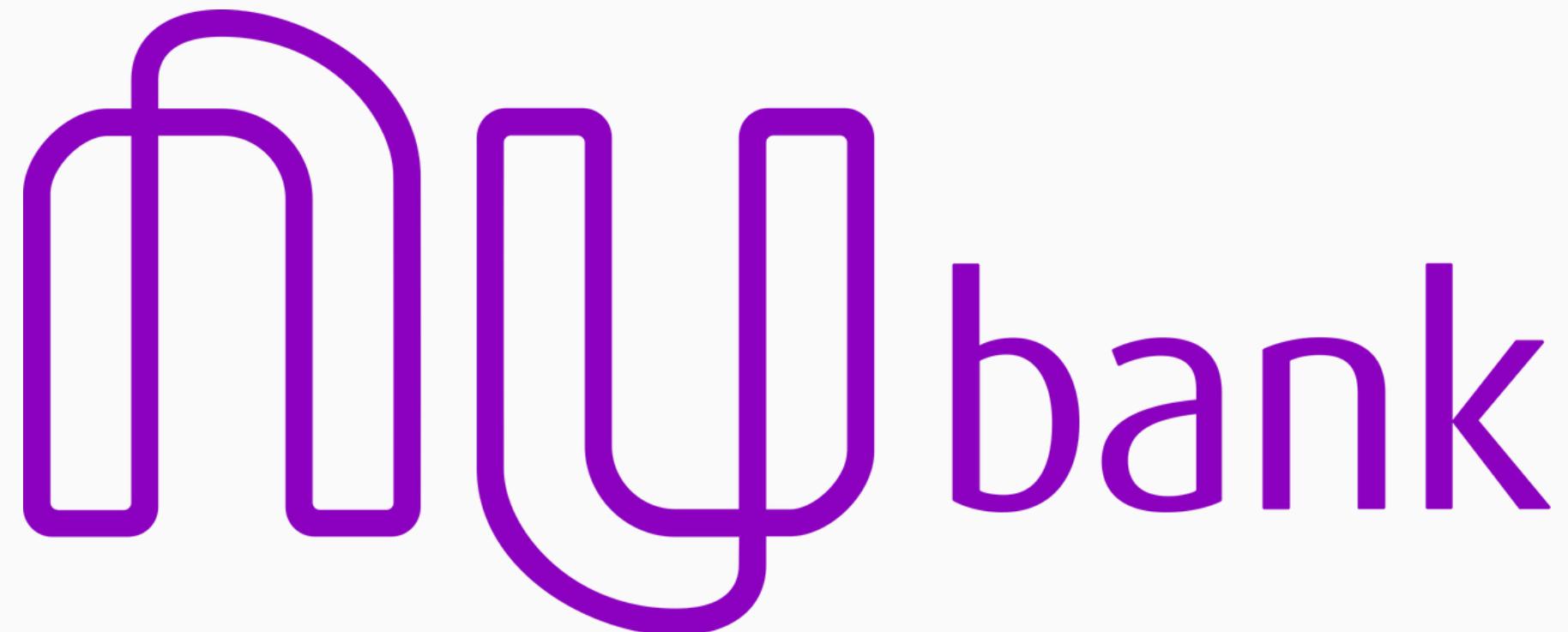
CLJSTYLE

Ferramenta de formatação de Código.

CLJ-KONDO

Ferramenta de análise estática (erros/incosistências - linter)

Empresas que usam Clojure



▲ throwawaylabs on April 23, 2019 | parent | context | favorite | on: Why did Clojure gain so much popularity?

I work at Walmart Labs. Clojure is super popular[1] in Walmart Labs. Clojure is fun and importantly it is a very productive language to work with. This gain in productivity comes from less state, thus easy to debug and fix, and functional approach, so the functions do what exactly what they are supposed to and there are no hidden surprises.

Less state or immutability means concurrency is easy. Easy concurrency means we can scale[2] applications with ease. I mean ease of development. It takes less work, less mental effort to take something that works for a single thread and make it work in a multithreaded and/or distributed manner over hundreds of thousands of cores. Debugging is easy too. P1 incidents for a microservices written in Clojure takes less time to debug compared to those written in other languages. So no wonder most of our mission critical services are written in Clojure.

For a complex environment where thousands of microservices are deployed on hundreds of thousands of computes, Clojure has been super successful.[3]

[1]: <https://github.com/walmartlabs?language=clojure>

[2]: <http://blog.cognitect.com/blog/2015/6/30/walmart-runs-clojur...>

[3]: https://clojure.org/community/success_stories

Produtividade, escalabilidade, debug

Why is Puppet Labs moving all their server-side applications to Clojure? In a word - [results](#). Drastic improvements in performance. Increased scalability. Ease of deployment. In several more words - the power and ecosystem of the JVM, Clojure's functional strengths of immutability and concurrency, and an active development community. Leveraging their [Trapperkeeper](#) library, an open source Clojure framework for composing large-scale applications, Puppet Labs has enabled any organization to up their game for building highly reliable long-running applications and services.

Performance, comunidade, confiabilidade

Na época em que o Nubank foi fundado, Clojure pareceu a melhor opção para os problemas que precisávamos resolver. Hoje, com mais de 15 milhões de clientes, todas as áreas do Nubank usam Clojure e mais de 90% dos **microsserviços** são escritos nessa linguagem.

“Criamos diversas bibliotecas internas em Clojure”, conta Bruno. “Se precisamos fazer cálculos bancários, de juros ou de negócios, por exemplo, usamos nossas próprias bibliotecas”, complementa.

FIM

