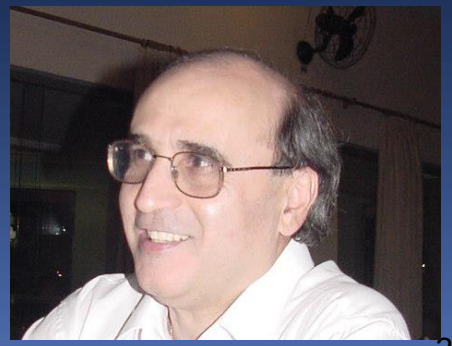
THE CLOJURE LANGUAGE

Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@prof.uscs.edu.br
aparecidovfreitas@gmail.com

Prof. Aparecido V. de Freitas

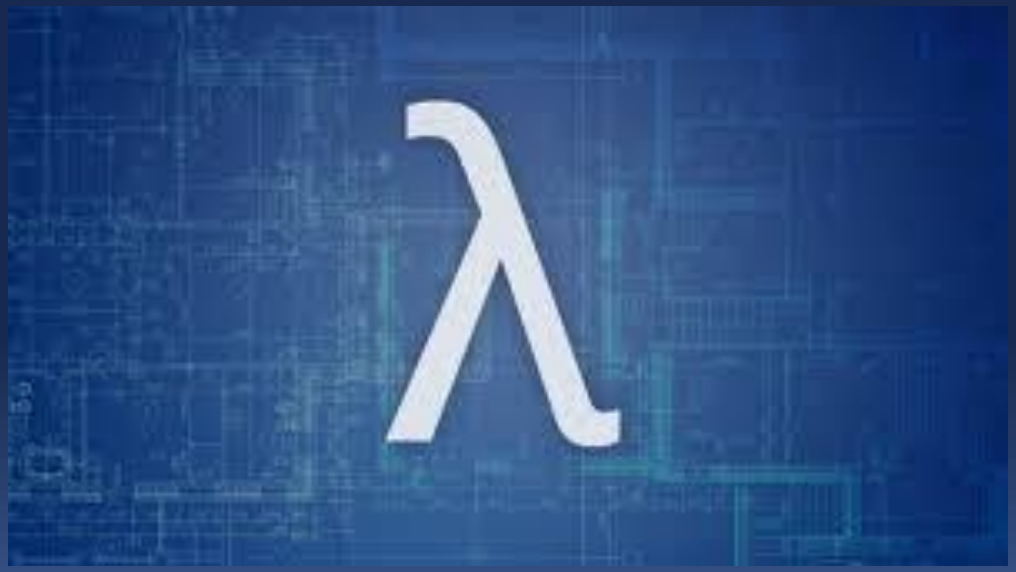
- **Doutor** em Engenharia da Computação pela **EPUSP** – Escola Politécnica da USP
- **Mestre** em Engenharia da Computação pela **EPUSP** – Escola Politécnica da USP
- Especialização em Engenharia de Software pela EPUSP – Escola Politécnica da USP
- **Engenharia** Plena pela Escola de Engenharia **Mauá**
- Bacharel em **Matemática** pela Fundação Santo André
- Atuou durante 15 anos como Analista e Supervisor de **TI** na área de TI da **Volkswagen** do Brasil
- Especialista na plataforma **IBM i** (desde 1993)
- Experiência na plataforma **IBM Mainframe** (15 anos)
- Professor da **USCS** desde a primeira turma do curso de Ciência da Computação (1989)
- Professor do Curso de Engenharia de Computação da Escola de Engenharia Mauá
- Ex-Gestor dos cursos de Computação da USCS há 13 anos (2000 a 2013)
- Ex-Professor do curso de Ciência da Computação da Universidade **Metodista**
- Ex-Professor do Curso de Matemática – Ênfase Software – **Fundação Santo André**
- Consultor e Instrutor em empresas de TI – Qualitsys Consultoria de Informática Ltda
- Certificação Internacional em Engenharia de **Requisitos** – **IREB** – CPRE
- Certificação internacional em **Testes** de Software – **ISTQB** – CTFL
- Certificação internacional em **Testes Ágeis** de Software – **ISTQB** – CTFL-AT



Outline

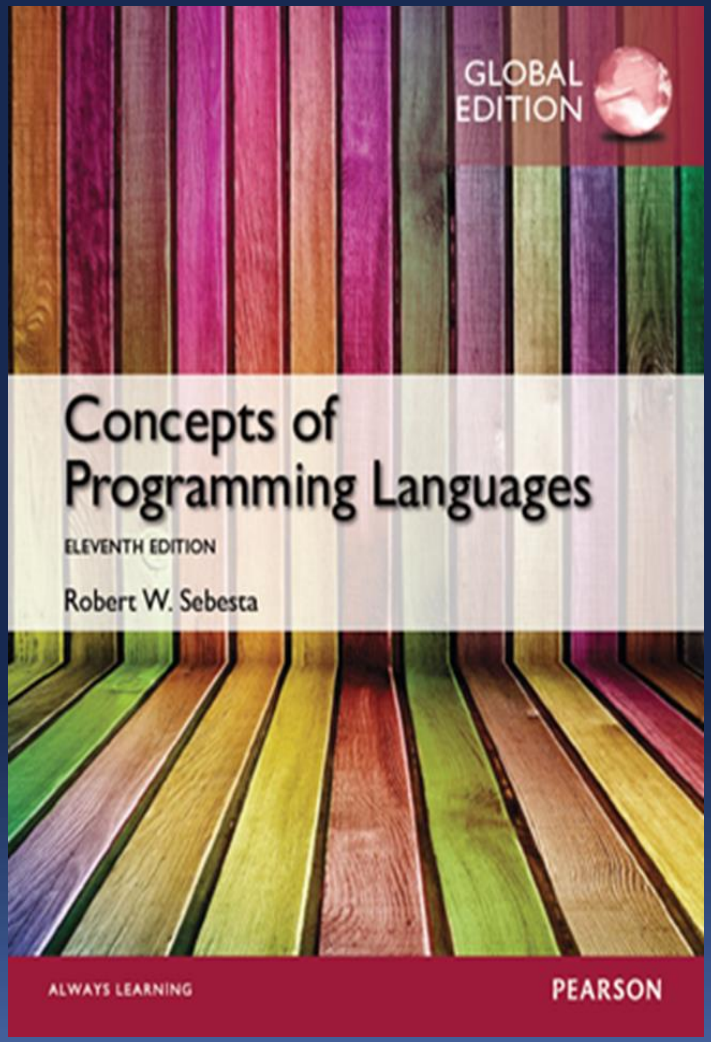
- ✓ O Paradigma Funcional
- ✓ Linguagens Funcionais
- ✓ Clojure – Instalação - Leiningen
- ✓ Plugins para IDE's (ATOM)
- ✓ Desenvolvimento **REPL**
- ✓ Estruturas de Dados
- ✓ Interoperabilidade com Java
- ✓ API's em Clojure - Pedestal
- ✓ Interação com Banco de Dados (MySQL - JDBC)
- ✓ ClojureScript

Clojure é uma Linguagem Puramente Funcional



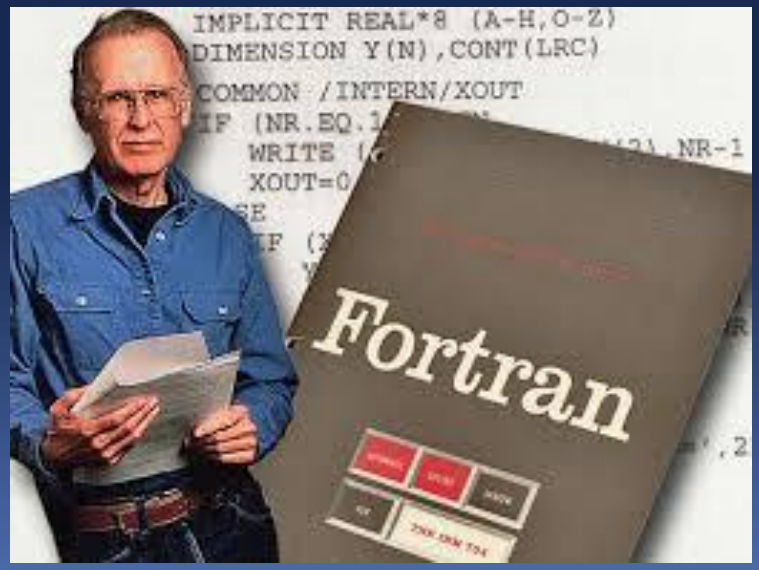
Recomendo ...

- **R. Sebesta – Concepts Of Programming Languages**

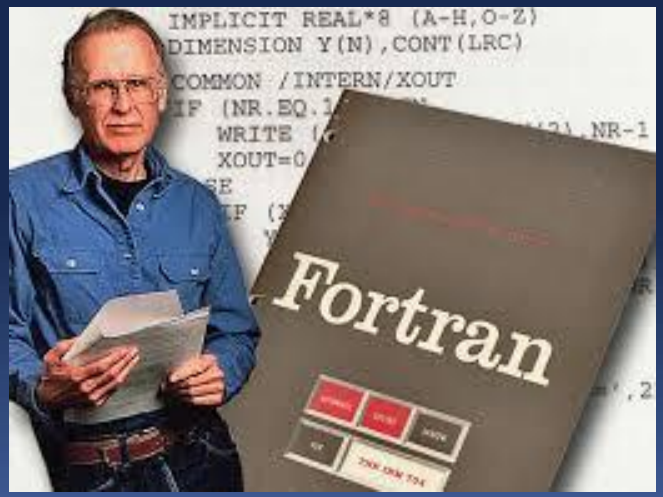


O Paradigma Funcional

- ✓ Fortran foi a primeira linguagem de programação (**Imperativa**);
- ✓ O primeiro compilador Fortran foi desenvolvido por uma equipe da IBM sendo chefiada por **John Backus**, na década de 50;
- ✓ A partir da Linguagem Fortran, diversas outras linguagens foram desenvolvidas;
- ✓ Mas, em 1977 na palestra ministrada por John Backus quando ganhou o prêmio ACM Turing, ele argumentou que **linguagens funcionais** são melhores que as **imperativas**, pois podem apresentar mais **confiabilidade**, **legibilidade** e com maior probabilidade de estarem **corretas**.

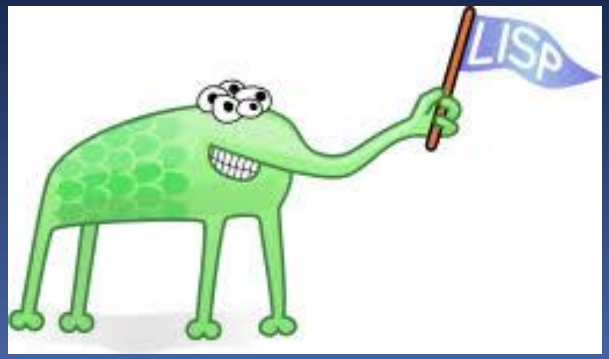


Em que se baseou Backus para afirmar que Linguagens Funcionais apresentam maior legibilidade e confiabilidade ?



Linguagens Funcionais

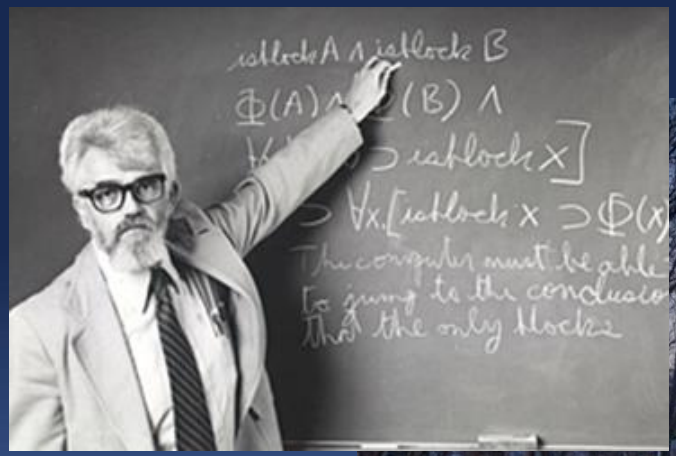
- ✓ O **argumento** de John Backus teve como base que em **Linguagens Puramente Funcionais**, o significado das expressões são independentes de contexto;
- ✓ Em **Linguagens Puramente Funcionais** nem expressões nem funções apresentam Efeitos Colaterais (**Side Effects**);
- ✓ **Backus** propôs na época uma nova Linguagem Funcional chamada **FP** (Functional Programming) para embasar seu argumento;
- ✓ A linguagem não vingou, mas abriu espaço para pesquisa do Paradigma Funcional de Programação;

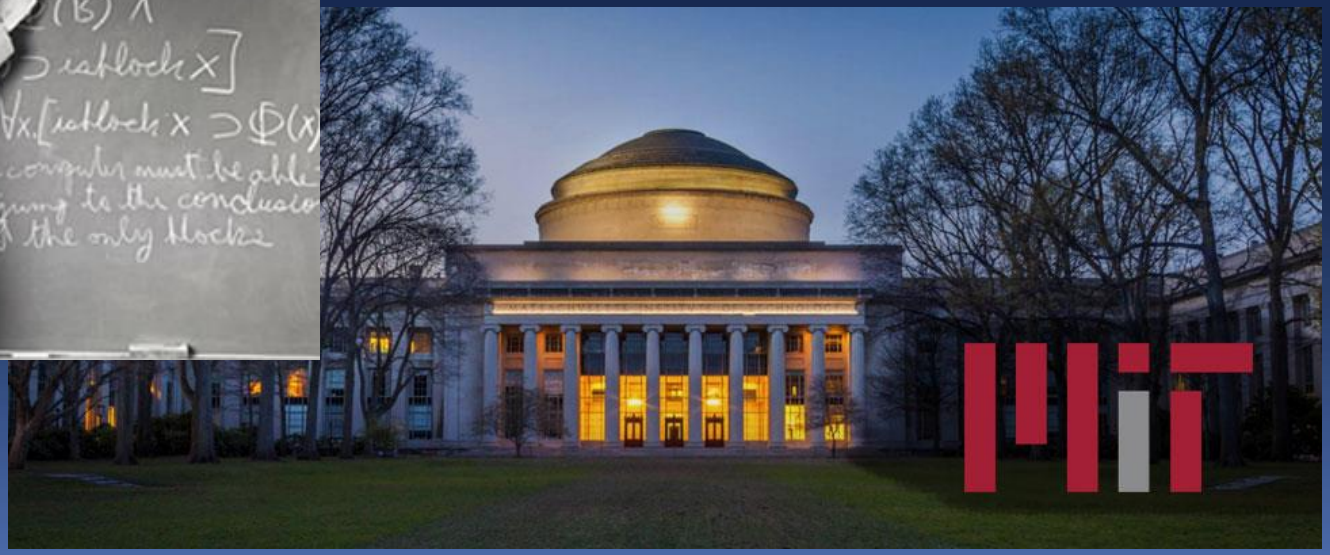


Clojure é um dialeto LISP

Linguagem de Programação Lisp

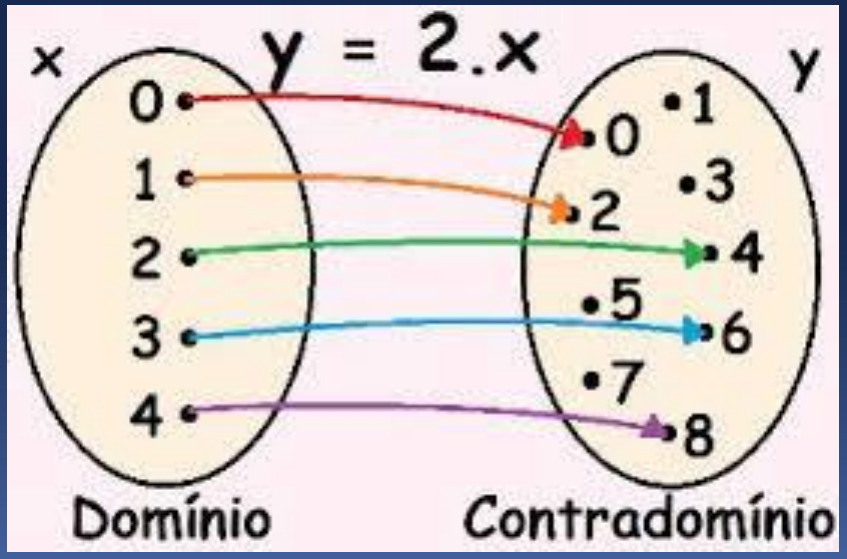
- ✓ Concebida por John McCarthy em 1959, no **MIT**;
- ✓ Focada no uso exclusivo de **funções matemáticas** como estrutura de dados;
- ✓ Tem como base formal o Cálculo Lambda de Alonzo Church;
- ✓ É ainda a mais importante linguagem representativa do Paradigma Funcional.





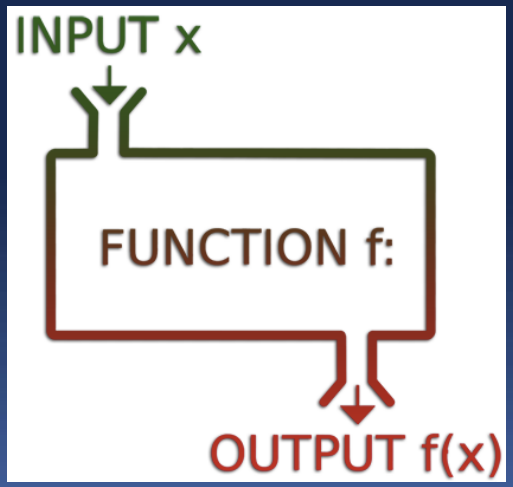
Funções Matemáticas

- ✓ Uma **função** matemática é um **mapeamento** de elementos de um conjunto, chamado conjunto **Domínio**, para outro conjunto, chamado **Contra-Domínio** (range set);
- ✓ O mapeamento é descrito por uma expressão;
- ✓ Funções são geralmente aplicadas a um elemento específico do **Domínio**, passado como argumento para a função;
- ✓ Ao se aplicar um argumento à função obtém-se um valor do **Contra-Domínio**.

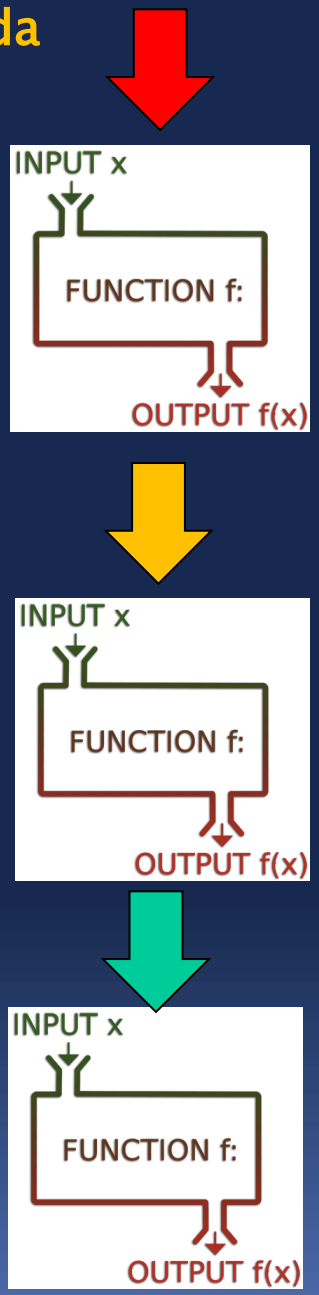


Paradigma Funcional

- ✓ Utiliza Funções matemáticas como **Higher Order Functions !!!**
- ✓ Diferentemente do Paradigma Imperativo que se baseia em **estados** em Memória
- ✓ Funções matemáticas sempre mapeiam o **mesmo** valor do Contra-Domínio para um valor do Domínio; (Não há side effects)
- ✓ Essa particularidade das funções matemáticas **não** ocorre nas linguagens imperativas, pois nestas linguagens um **subprograma** pode depender dos valores **correntes** de diversas **variáveis não-locais** ou **globais**, causando assim **side effects**.



Paradigma Funcional



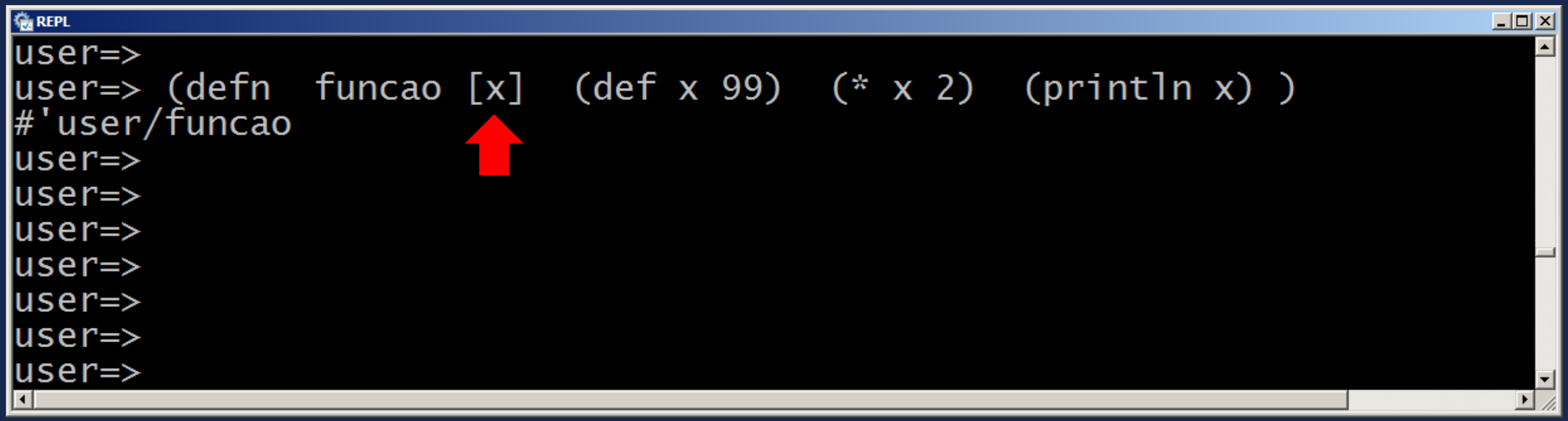
- ✓ O processamento é uma abstração de Encadeamento de Funções !!!
- ✓ O programa opera em transformação de dados !

IMUTABILIDADE favorece a CONCORRÊNCIA

Imutabilidade

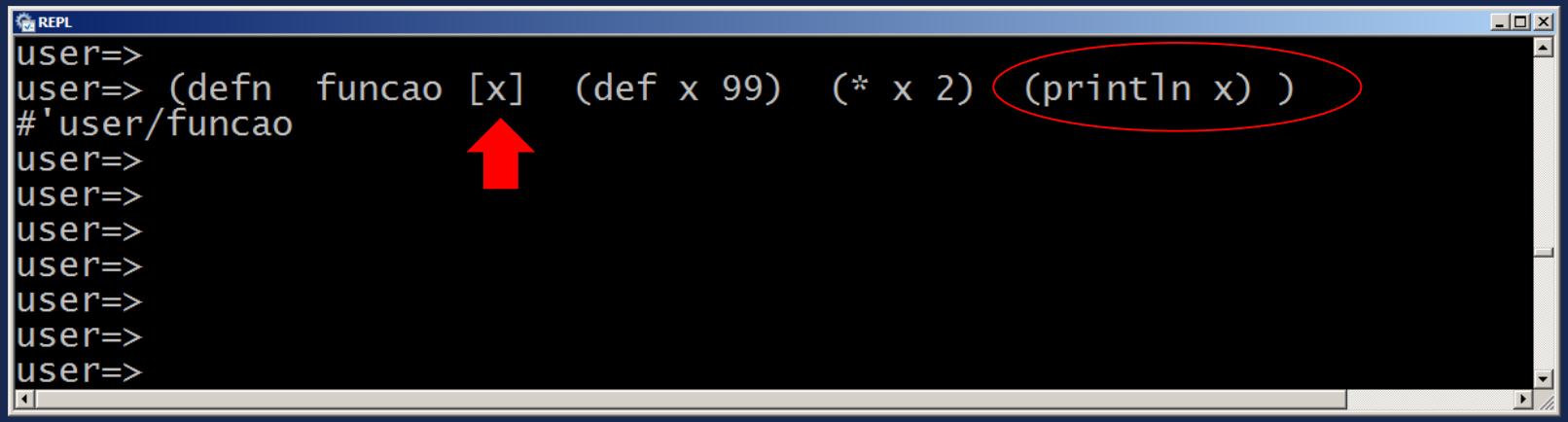
- ✓ Característica fundamental das Linguagens Puramente Funcionais !
- ✓ Processamento baseado em TRANSFORMAÇÃO de DADOS !!!
- ✓ Os **argumentos** passados à uma função são **imutáveis**
- ✓ Esse conceito é chamado **Imutabilidade**.

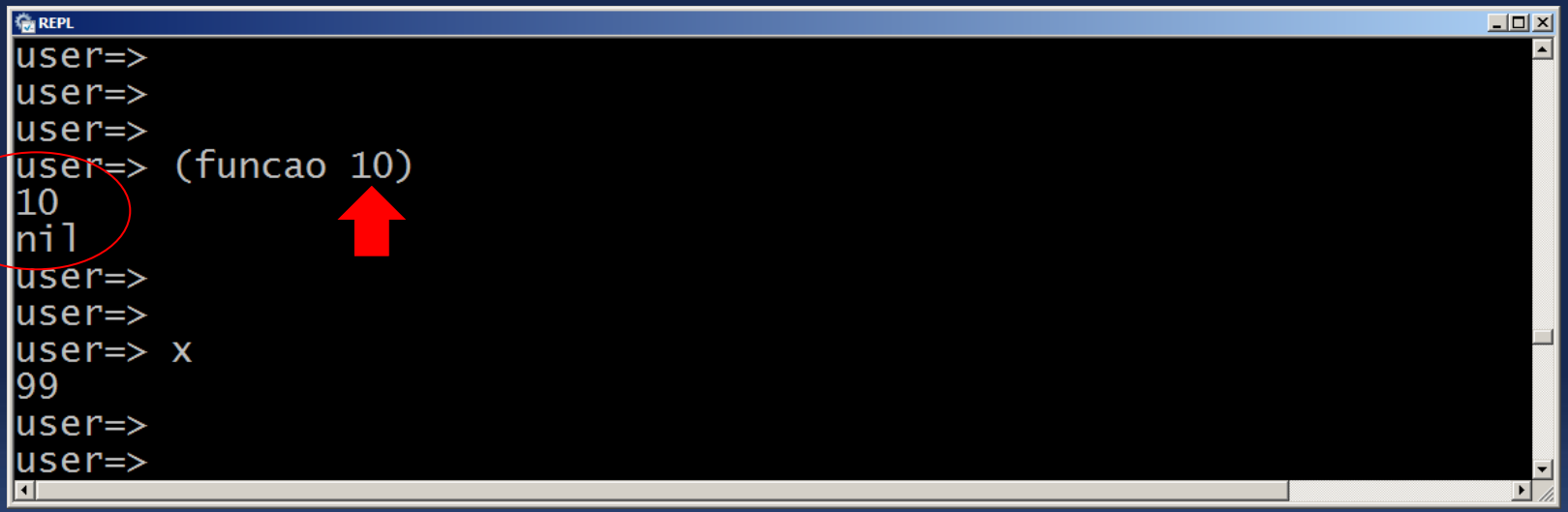
Exemplo - Clojure



- ✓ O parâmetro x é será ligado (bound) à algum argumento durante a avaliação;
- ✓ Esse argumento passado à função é **imutável**!

Exemplo - Clojure

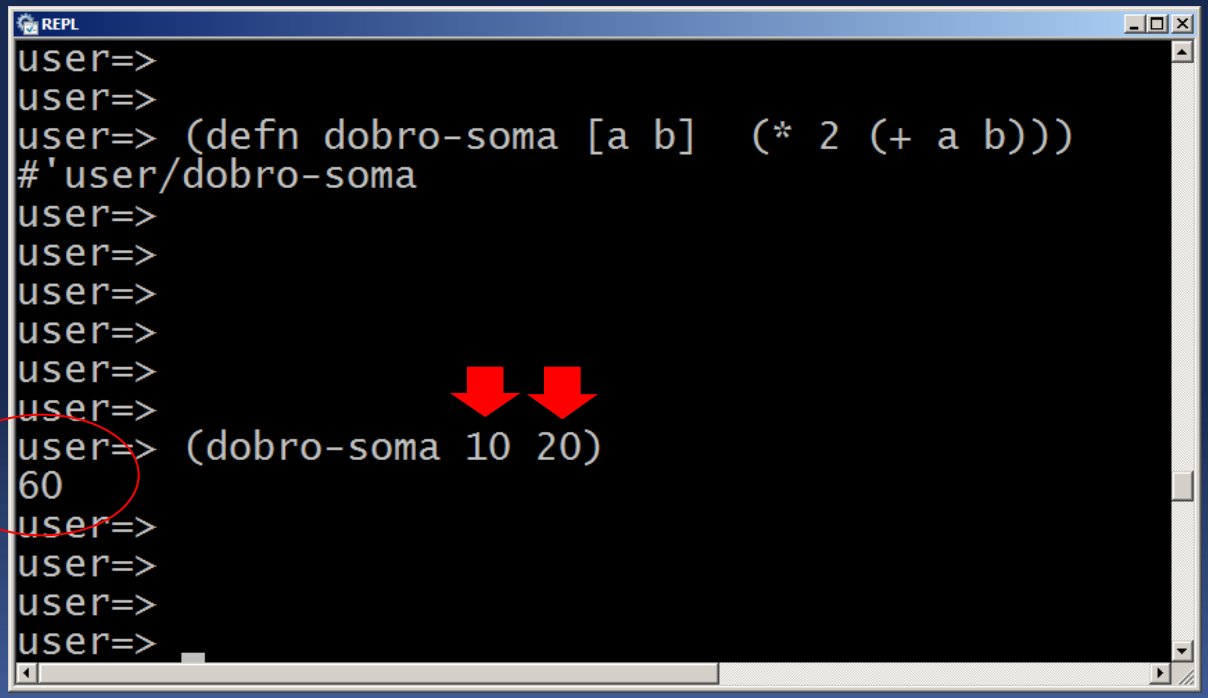




- ✓ O argumento **x** com o valor **10** é constante durante a avaliação.
- ✓ Embora tenha havido um binding anterior para **99**, a função **println** imprimiu, **por side effect**, o valor **10** (constante durante a avaliação).

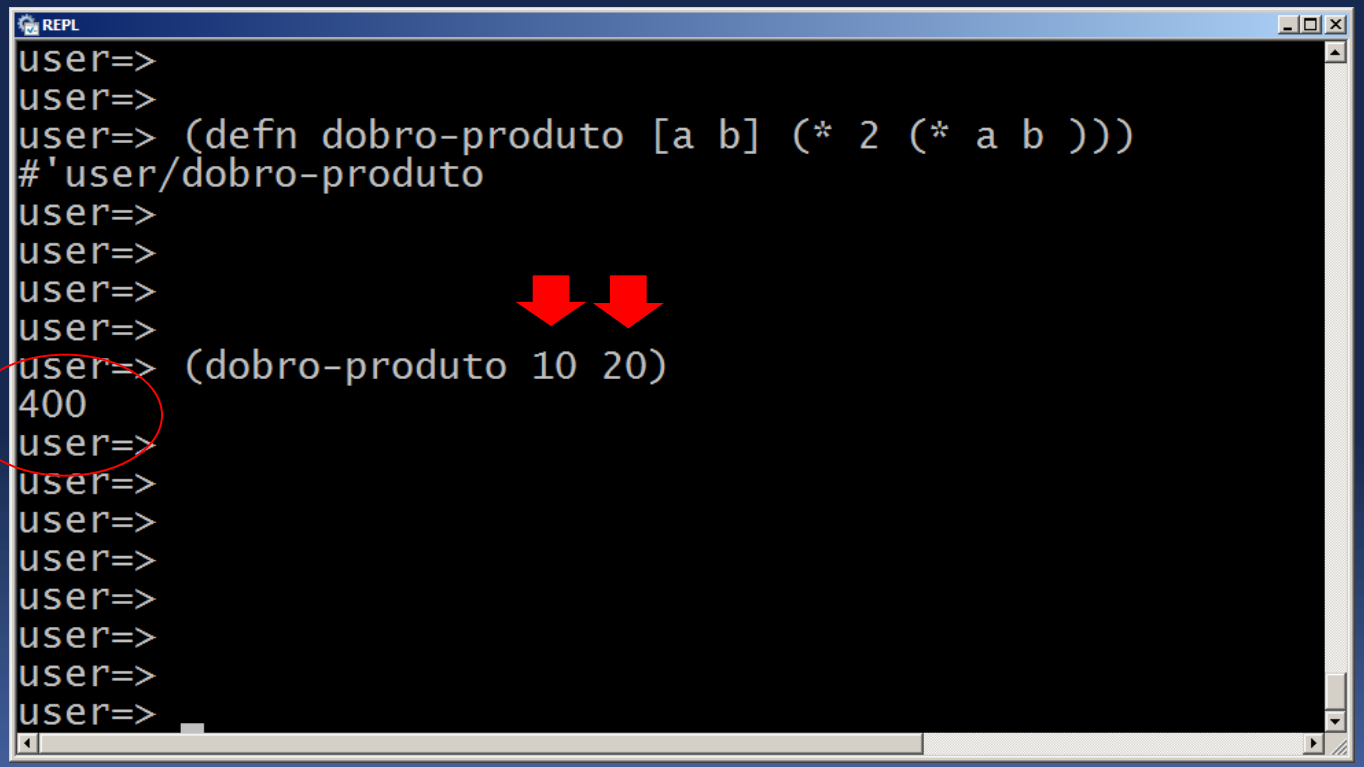
Higher Order Function - Clojure

- ✓ A função **dobro-soma** possui **2** parâmetros, **a** e **b**;
- ✓ A função retorna o dobro da soma dos **argumentos a** e **b** passados a ela;
- ✓ Por exemplo, para os argumentos **10** e **20**, a função retorna o valor **60**.



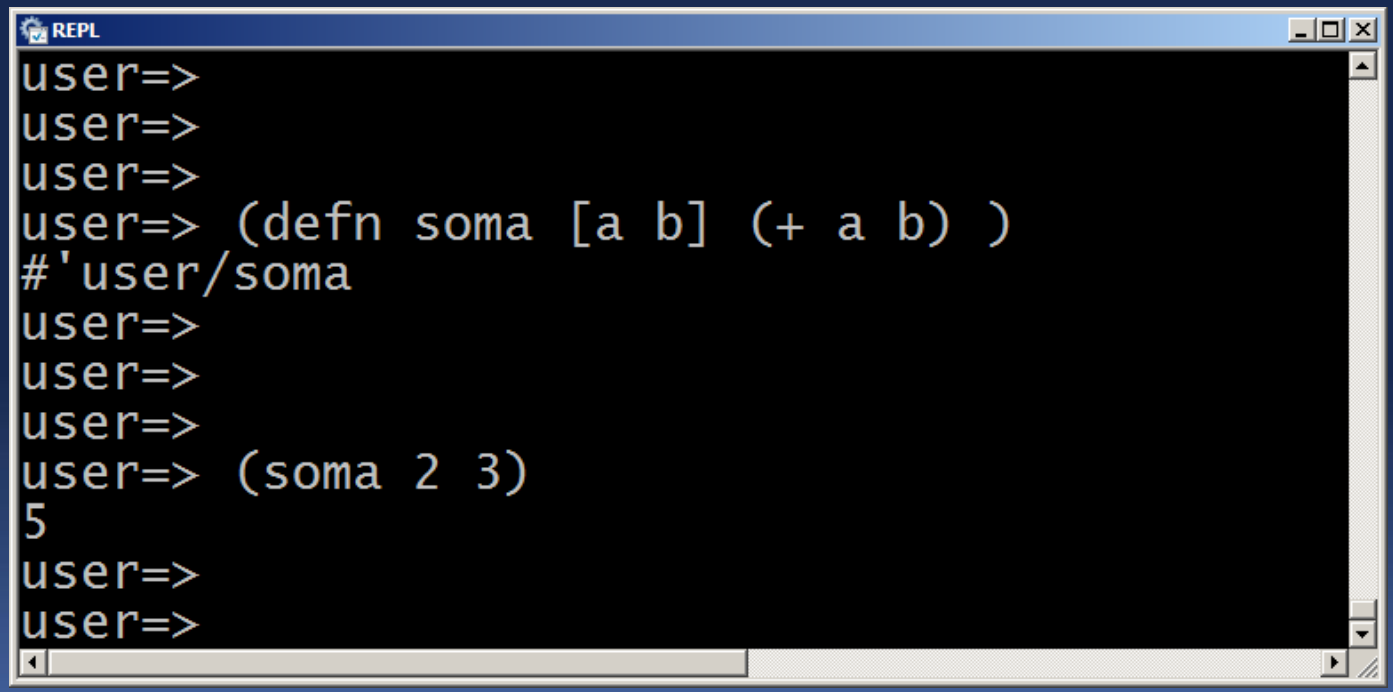
Higher Order Function - Clojure

- ✓ A função **dobro-produto** também possui **2** parâmetros, **a** e **b**;
- ✓ A função retorna o dobro do produto dos **argumentos a** e **b** passados a ela;
- ✓ Por exemplo, para os argumentos **10** e **20**, a função retorna o valor **400**



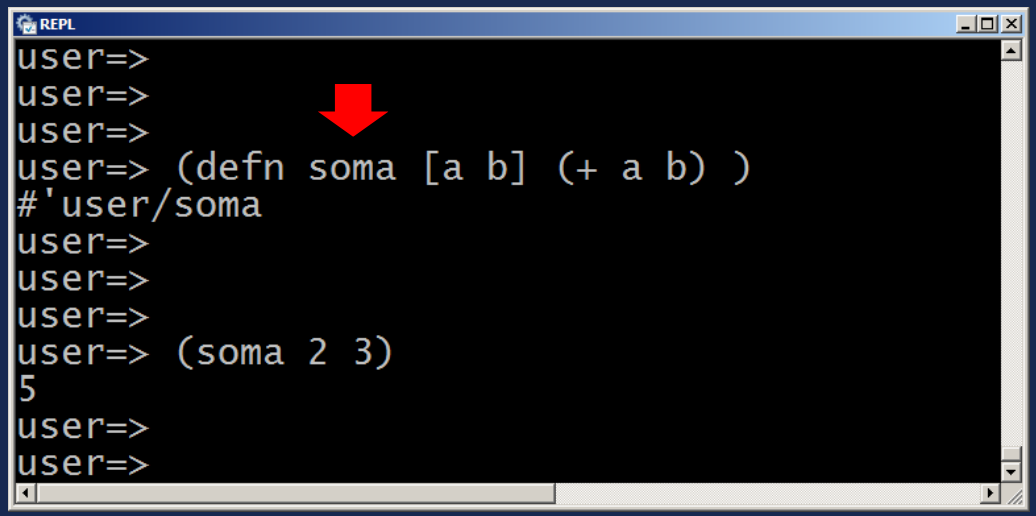
Higher Order Function - Clojure

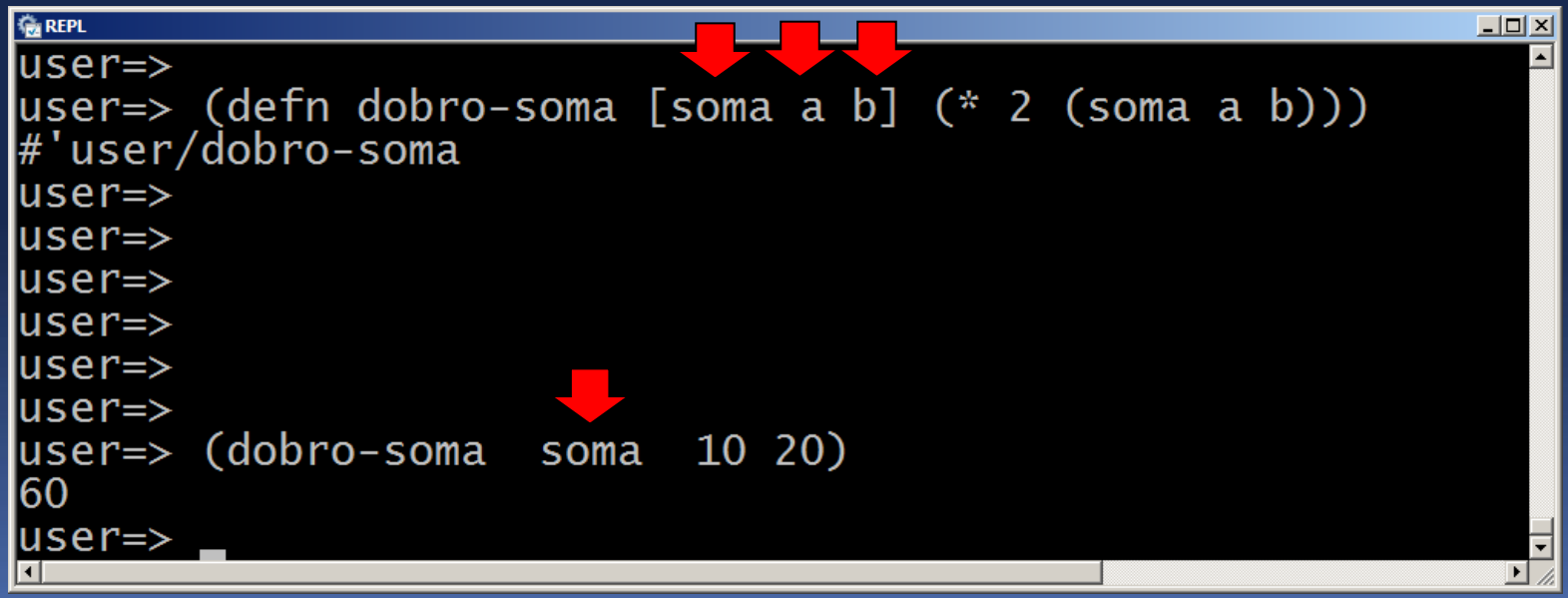
- ✓ As função **dobro-soma** e **dobro-produto,** compartilham um padrão;
- ✓ Elas somente diferem no nome na função usada para a computação de a e b;
- ✓ Poderíamos definir uma função chamada **soma** e passá-la para **dobro-soma**.



Higher Order Function - Clojure

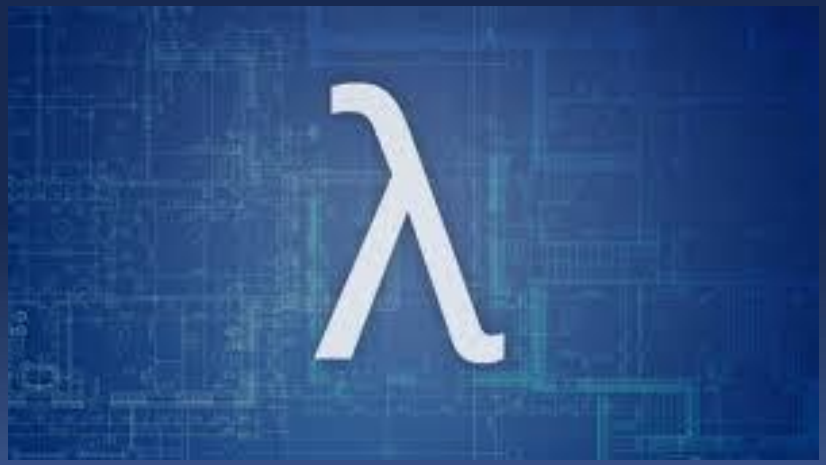
- ✓ Poderíamos definir uma função chamada **soma** e passá-la para **dobro-soma**.





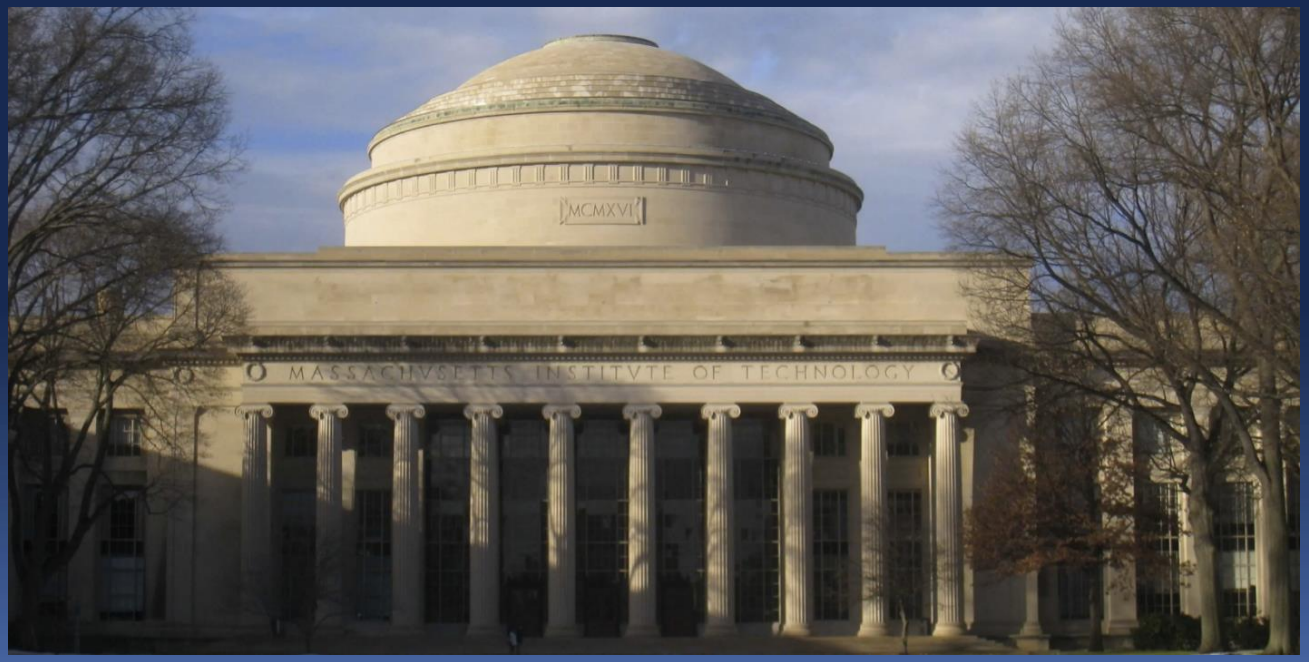
Objetivos da Programação Funcional

- ✓ Focar no emprego de funções matemáticas da forma mais intensa possível;
- ✓ Isso resulta numa abordagem totalmente diferente da Programação Imperativa;



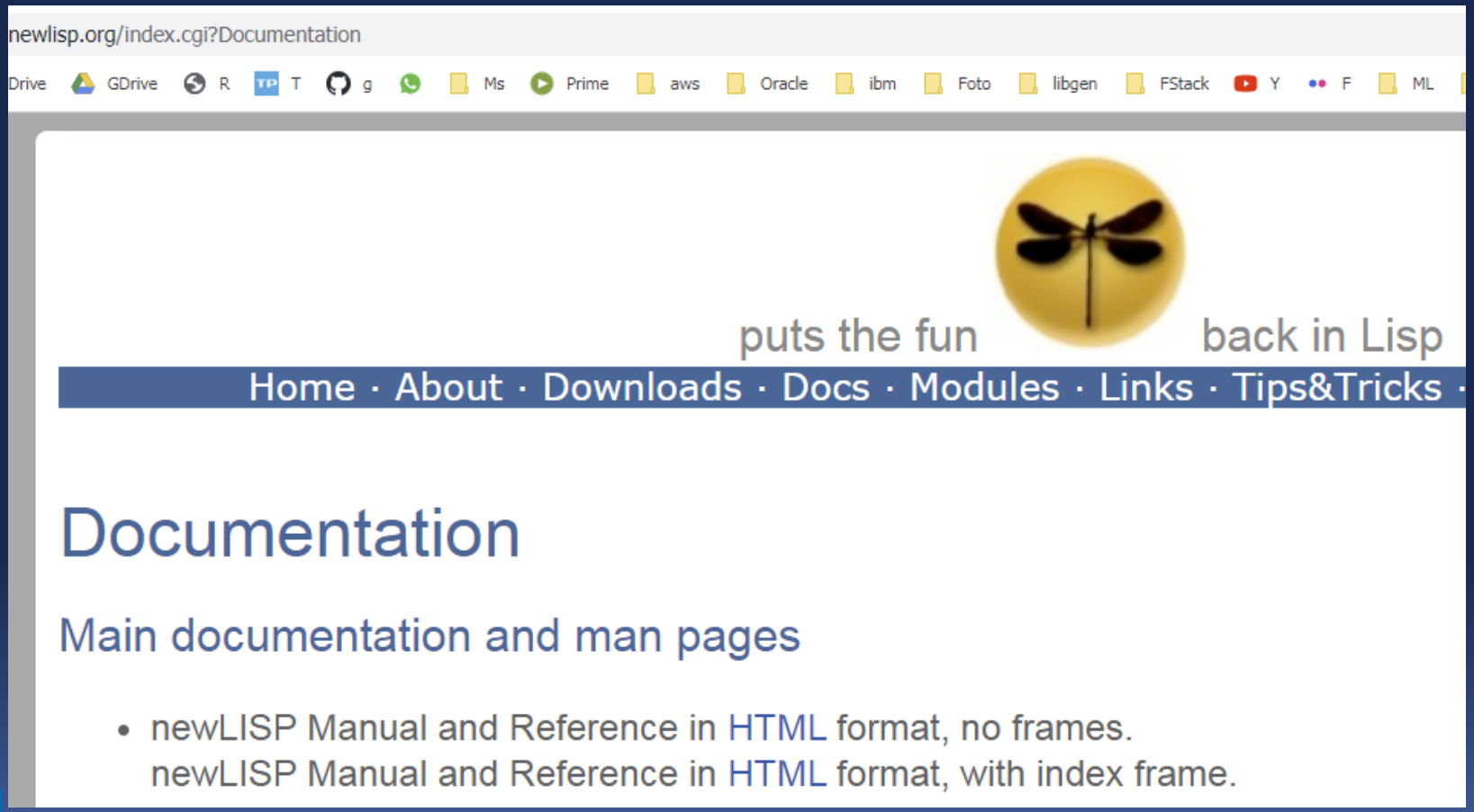
A Linguagem Lisp

- ✓ Primeira Linguagem do **Paradigma Funcional**;
- ✓ Desenvolvida no **MIT** em 1959;
- ✓ Atualmente existem diversos **dialetos** do **Lisp** que incluem algumas características das linguagens imperativas;



A Linguagem Lisp

http://www.newlisp.org/index.cgi?Documentation



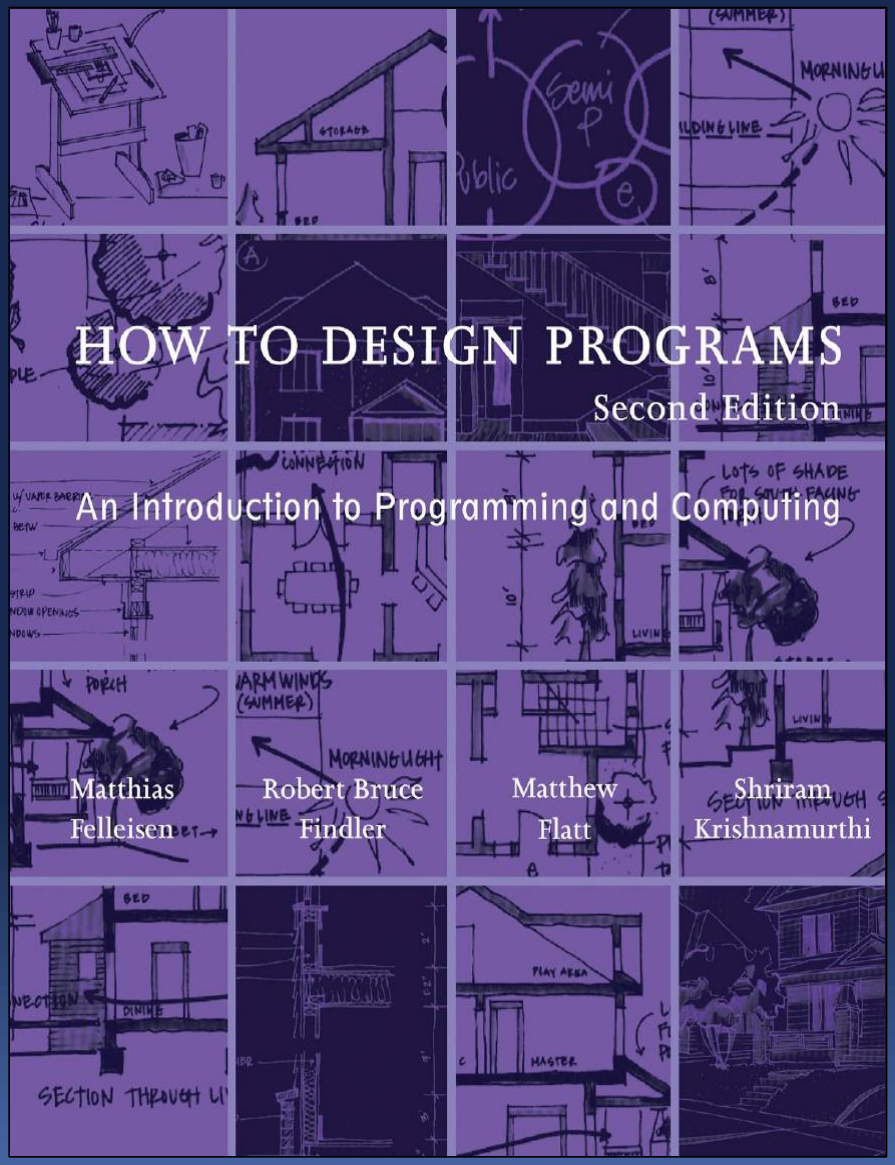
Downloading e Instalação

- ✓ O download do **newLISP** pode ser feito em http://www.newlisp.org/
- ✓ Disponível em diversas plataformas;
- ✓ O interpretador poderá ser chamado diretamente pela linha de comandos, em uma console;
- ✓ O código fonte pode ser gerado a partir de um editor de programas, como por exemplo o **Notepad++**.

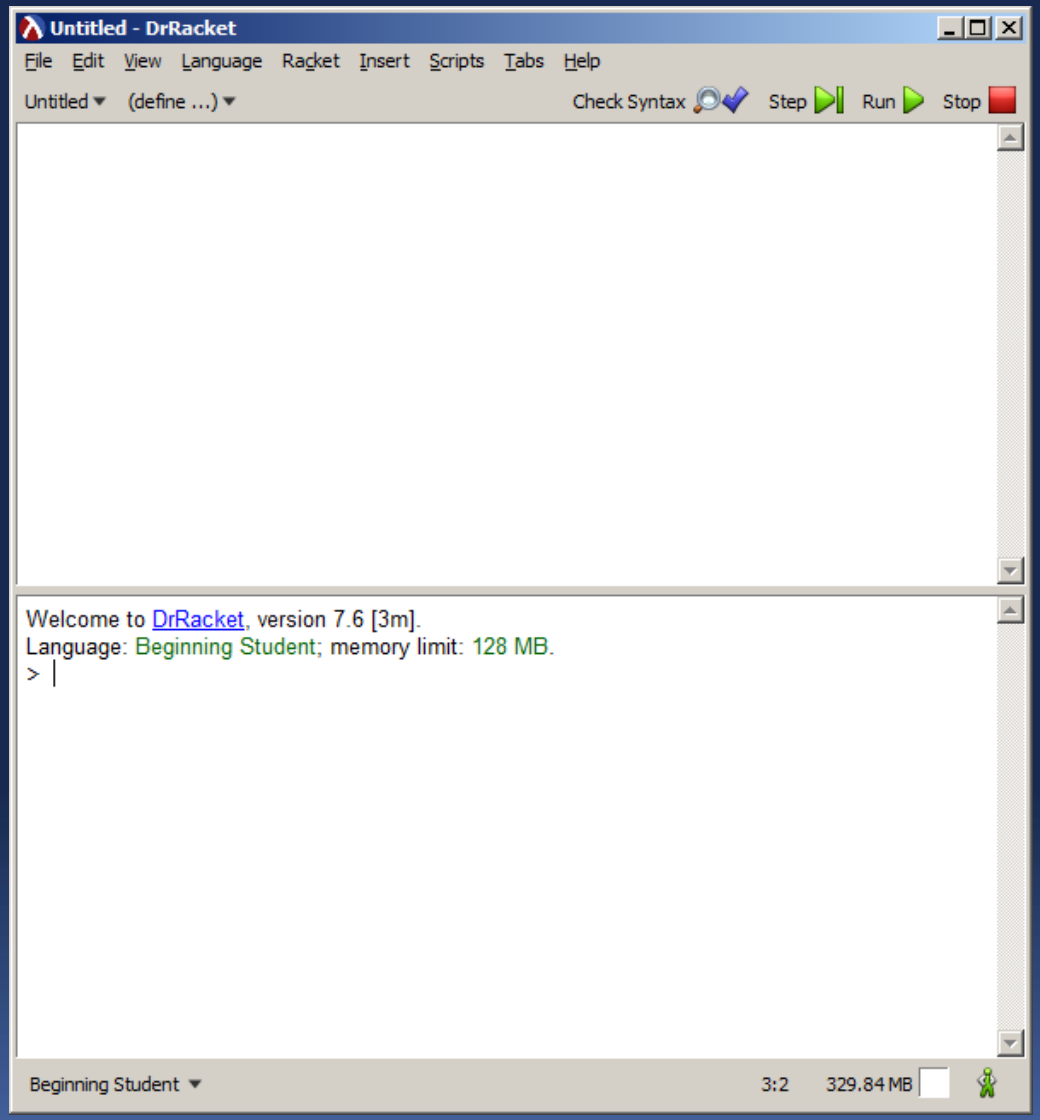
Linguagem Racket

- ✓ Dialeto **Scheme**;
- ✓ Desenvolvida no **MIT**;
- ✓ Propósito educacional e acadêmico;

Racket


HOW TO DESIGN PROGRAMS
Second Edition
An Introduction to Programming and Computing
Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi

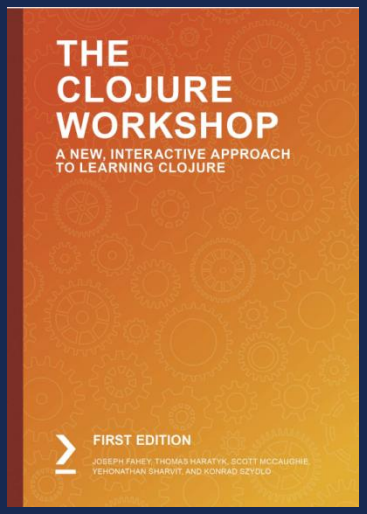
Racket

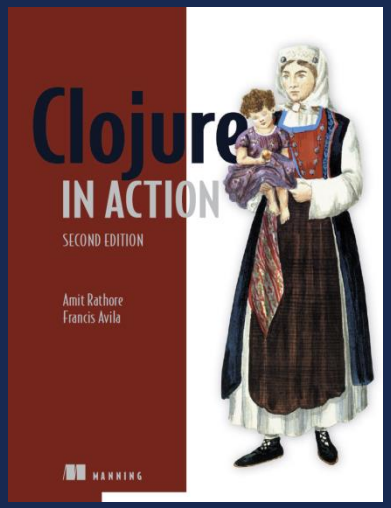

Untitled - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
Untitled (define ...)
Check Syntax Step Run Stop
Welcome to DrRacket, version 7.6 [3m].
Language: Beginning Student; memory limit: 128 MB.
>
Beginning Student
3:2
329.84 MB

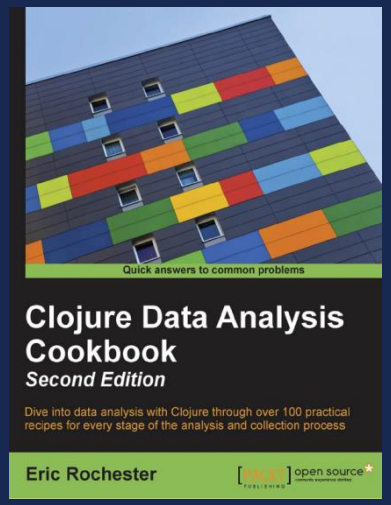
Linguagem Clojure

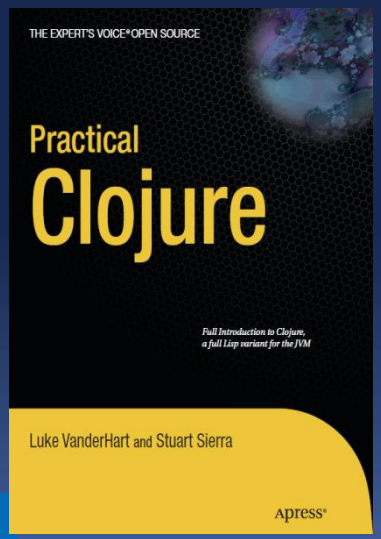
- ✓ Dialeto **Lisp**;
- ✓ Puramente Funcional;
- ✓ Criada por Rich Hickey;
- ✓ Executada na **Máquina Virtual Java** (JVM);
- ✓ Versões alternativas para .NET (Clojure CLR) e JavaScript (ClojureScript);
- ✓ Utilizada em grandes corporações (Walmart, Netflix, **Cisco**, Amazon, etc)
- ✓ Mantido **Nubank** **(desenvolvido pela** Cognitec)

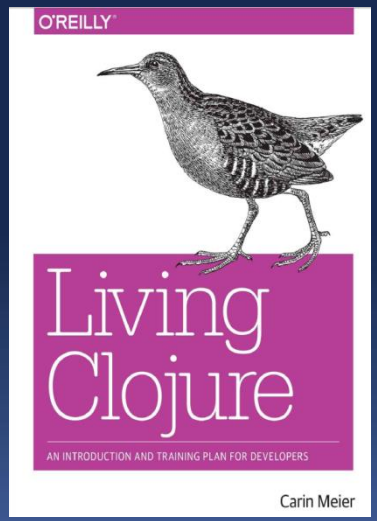
Bibliografia

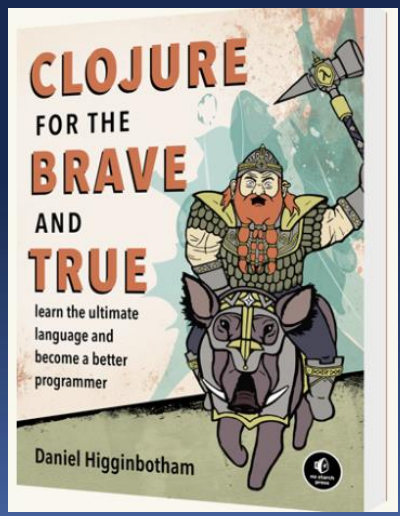








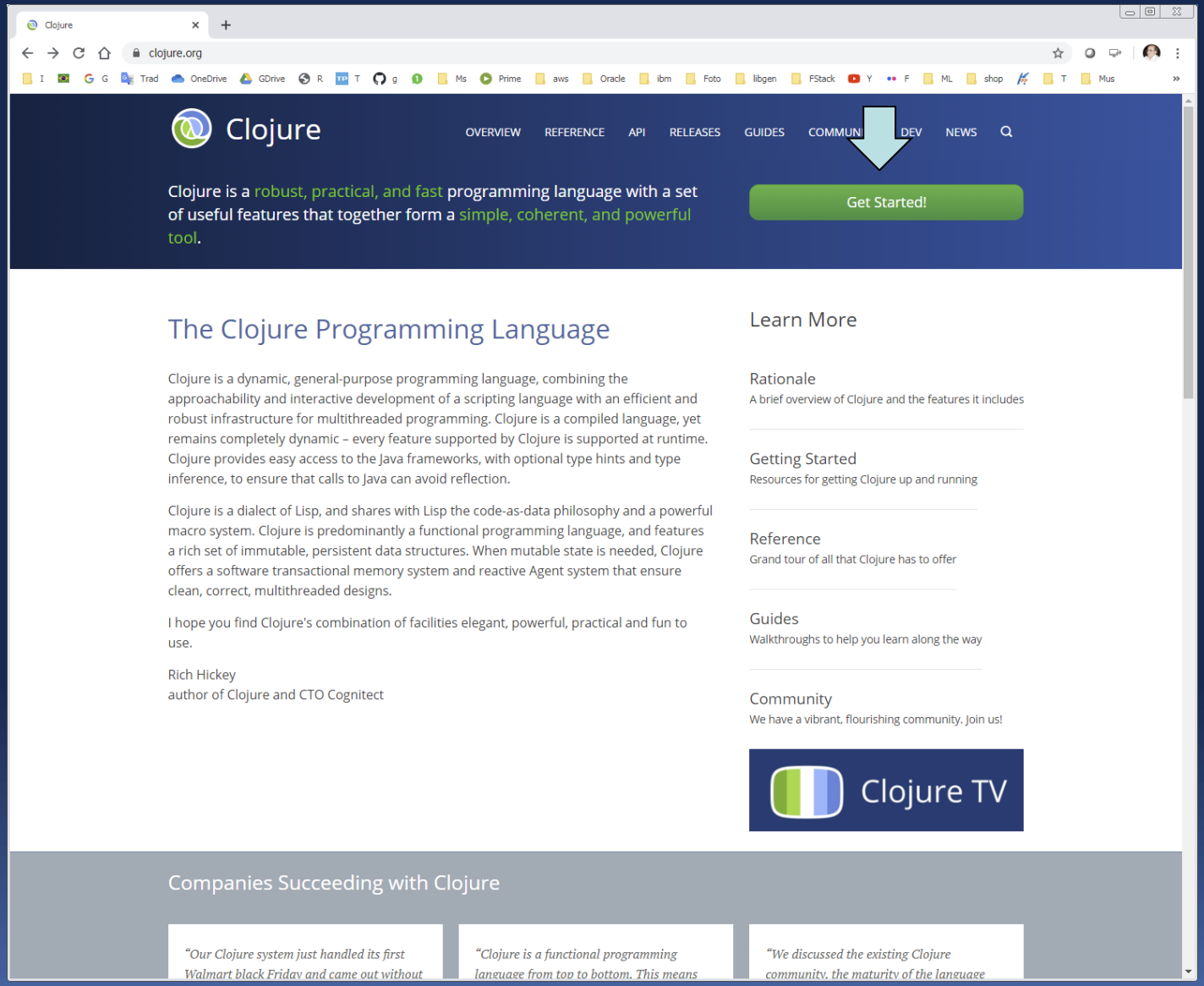




Antes de iniciar

- ✓ Antes de iniciar a instalação do Clojure, esteja certo de que **JDK** está instalado;
- ✓ Clojure é implementado em **Java** e roda na **JVM** (Java Virtual Machine);
- ✓ Clojure é desenhado para ser uma **hosted** language, enquanto que uma outra implementação, chamada **ClojureScript**, roda em qualquer runtime **JavaScript**, por exemplo, um web browser ou Node.js;
- ✓ Clojure **não** requer uma versão particular da Máquina Virtual Java;
- ✓ Mas, recomenda-se que a versão **Java 8** seja instalada;
- ✓ Instruções de instalação em https://clojure.org/
- ✓ É possível interagir códigos **Clojure** com libraries (**packages**) escritos em Java (API'S escritas em Java);

https://clojure.org/



Instruções de Instalação

Local build

Download and build Clojure from source (requires Git, Java, and Maven):

```
git clone https://github.com/clojure/clojure.git
cd clojure
mvn -Plocal -Dmaven.test.skip=true package
```

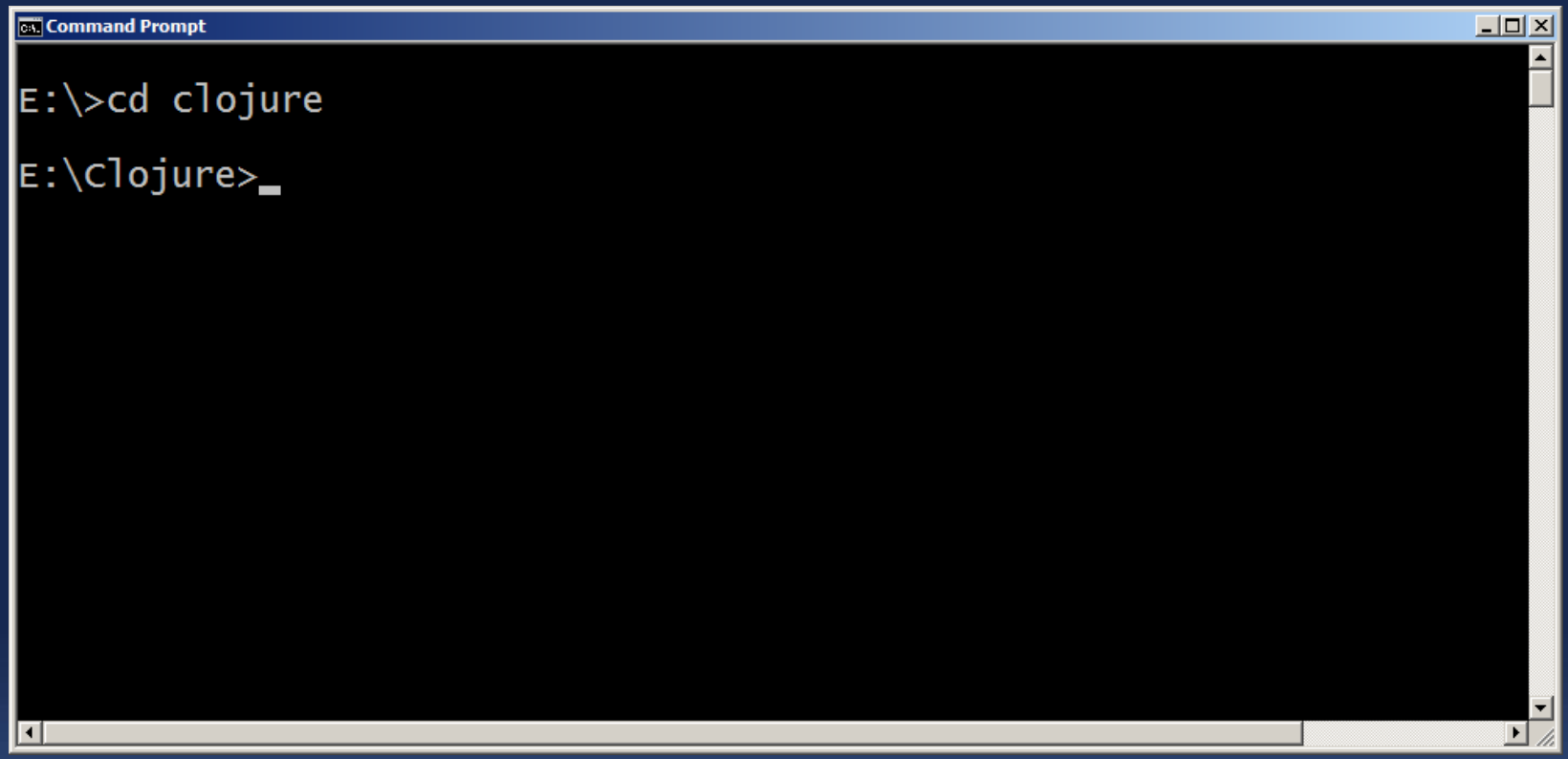
Then start the REPL with the local jar:

```
java -jar clojure.jar
```

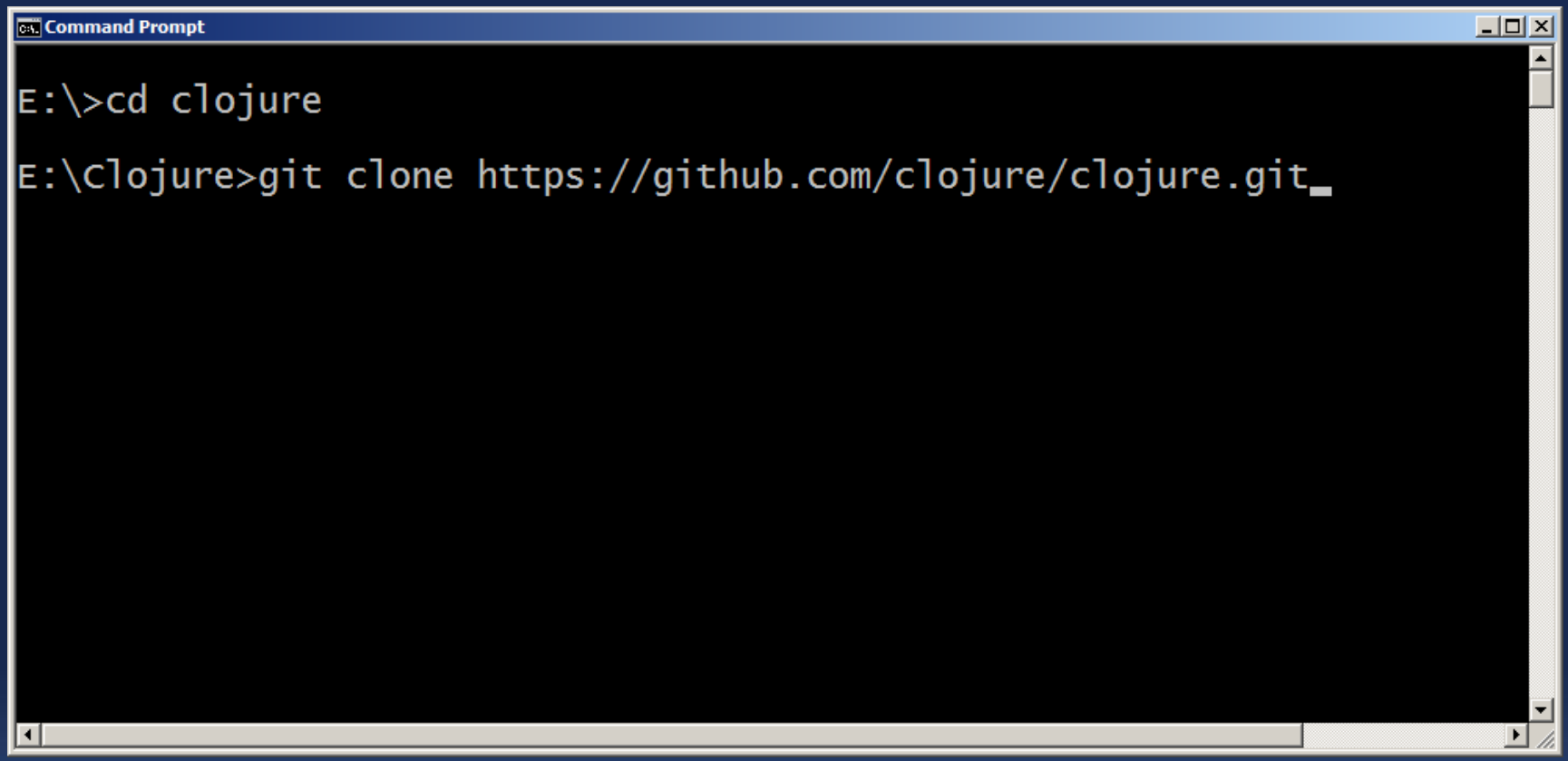
Try Clojure online

repl.it provides a browser-based Clojure repl for interactive exploration.

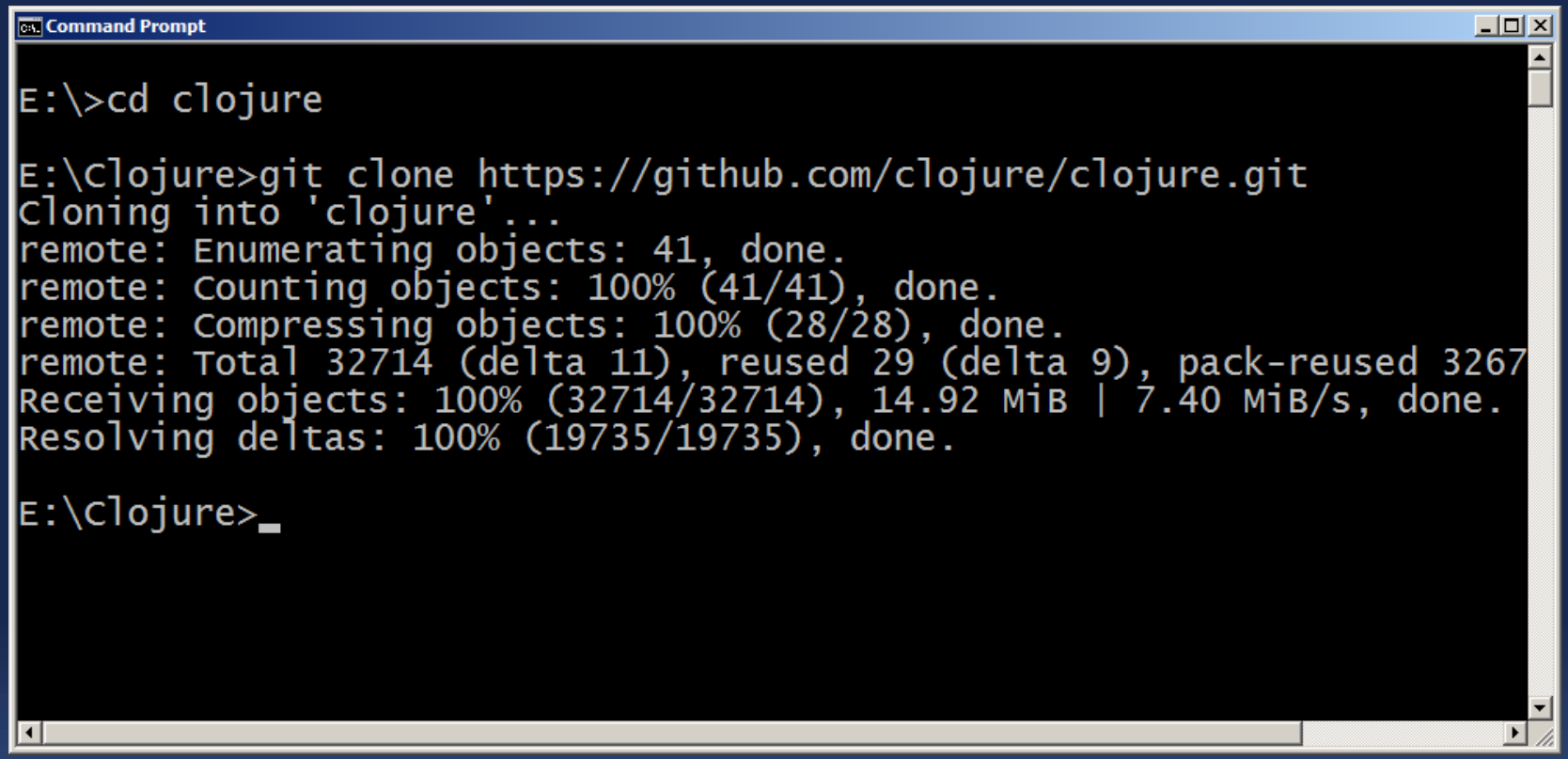
Criar pasta E:\clojure



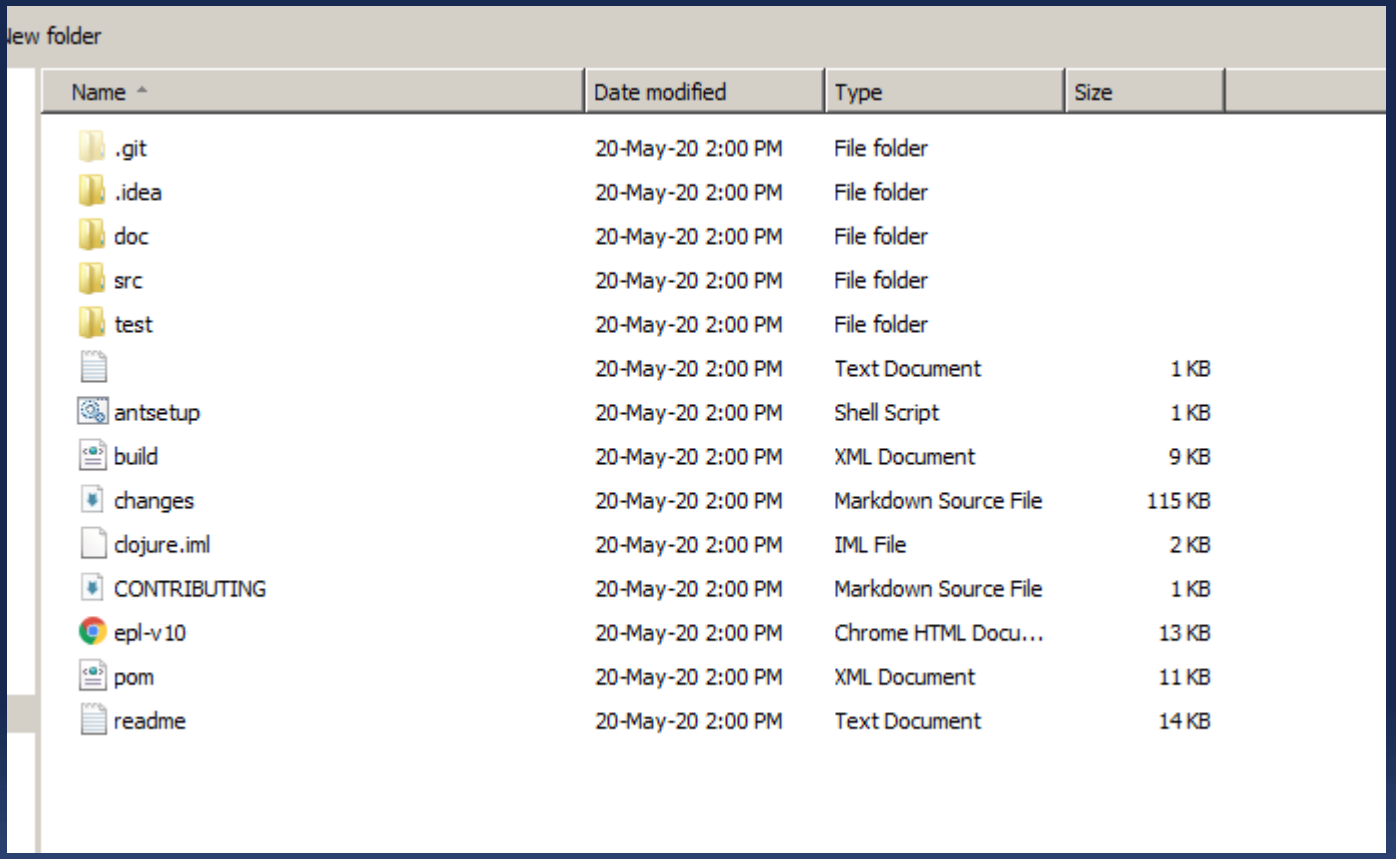
Baixando clojure



Baixando clojure

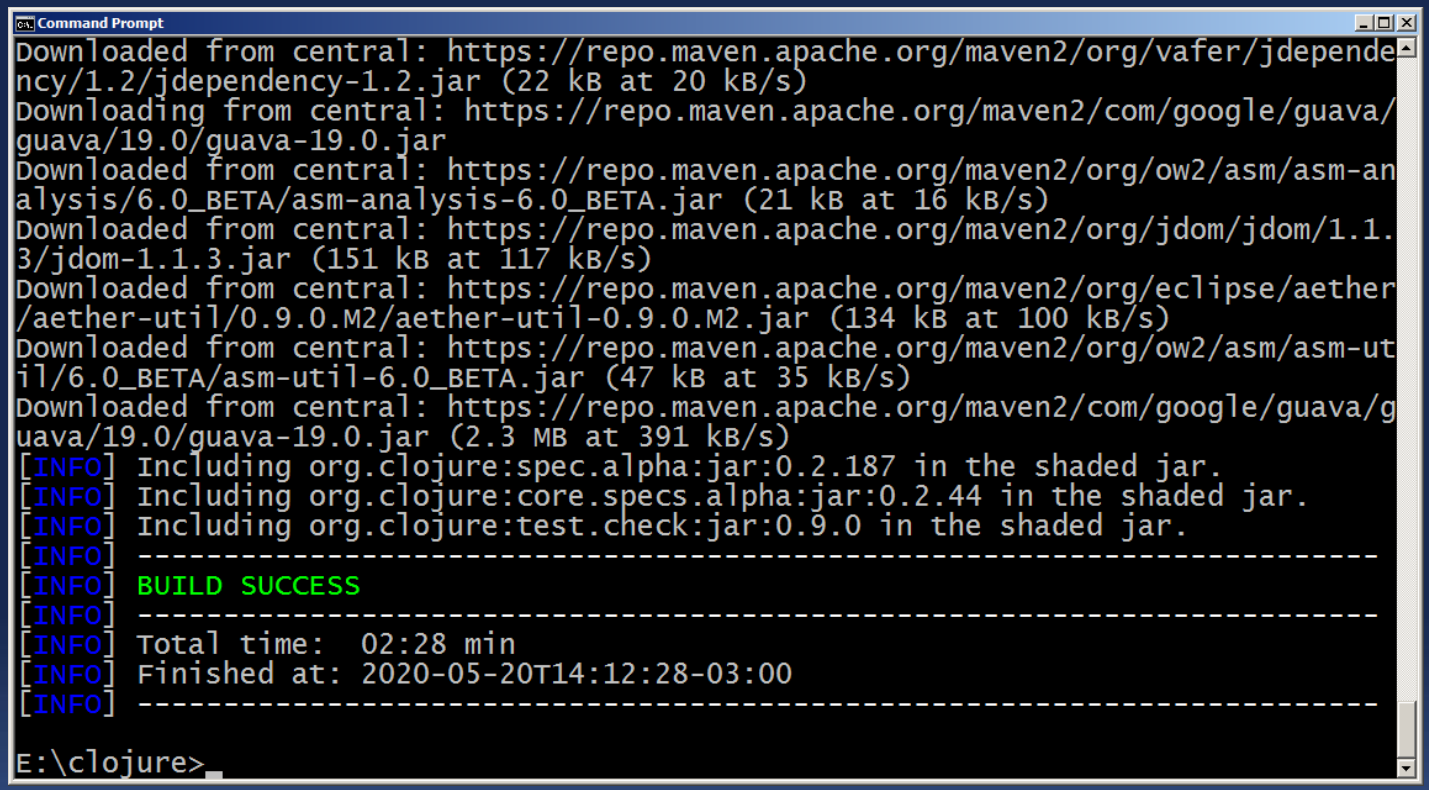


Pasta Clojure



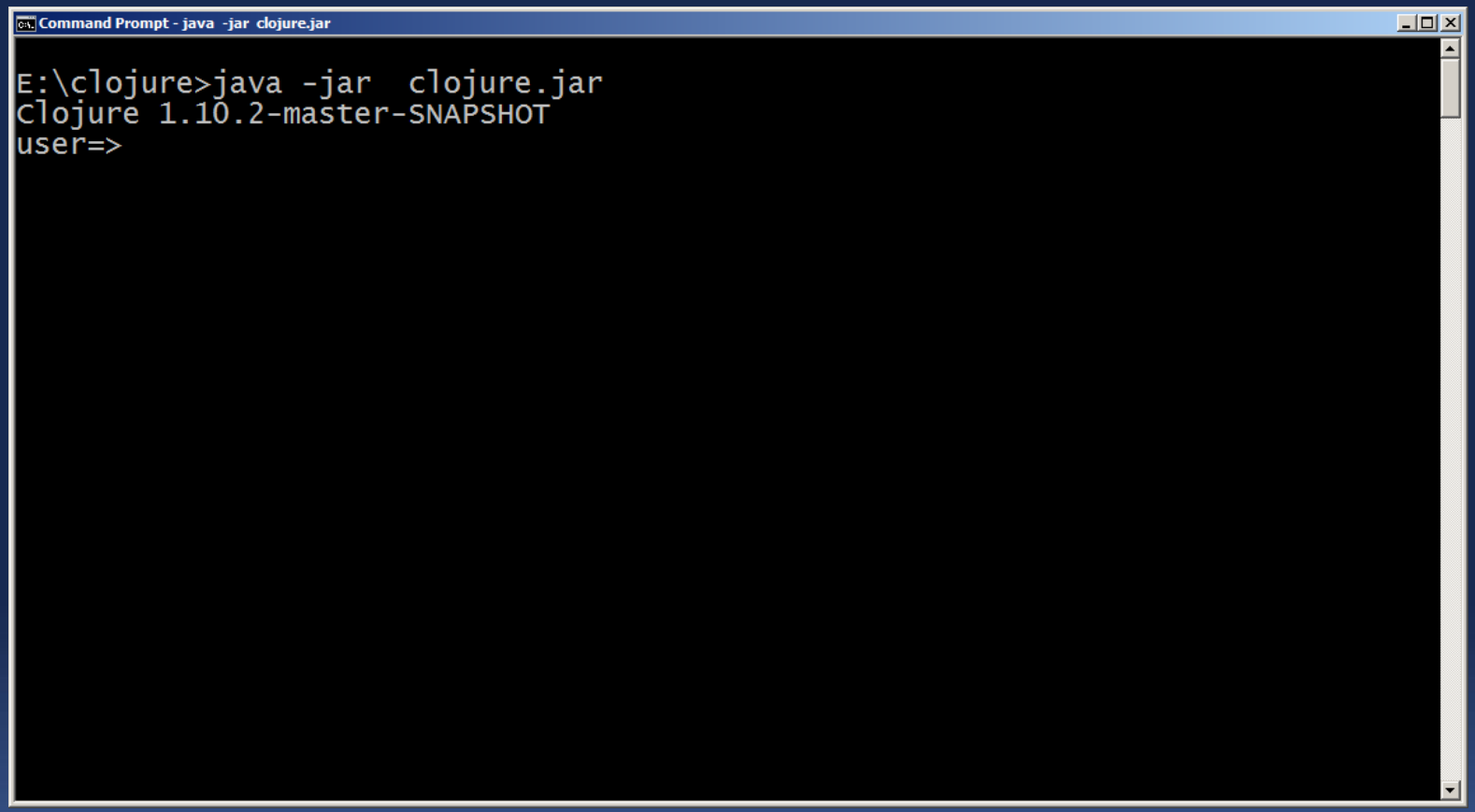
Building Clojure com Maven

Mvn –Plocal –Dmaven.test.skip=true package



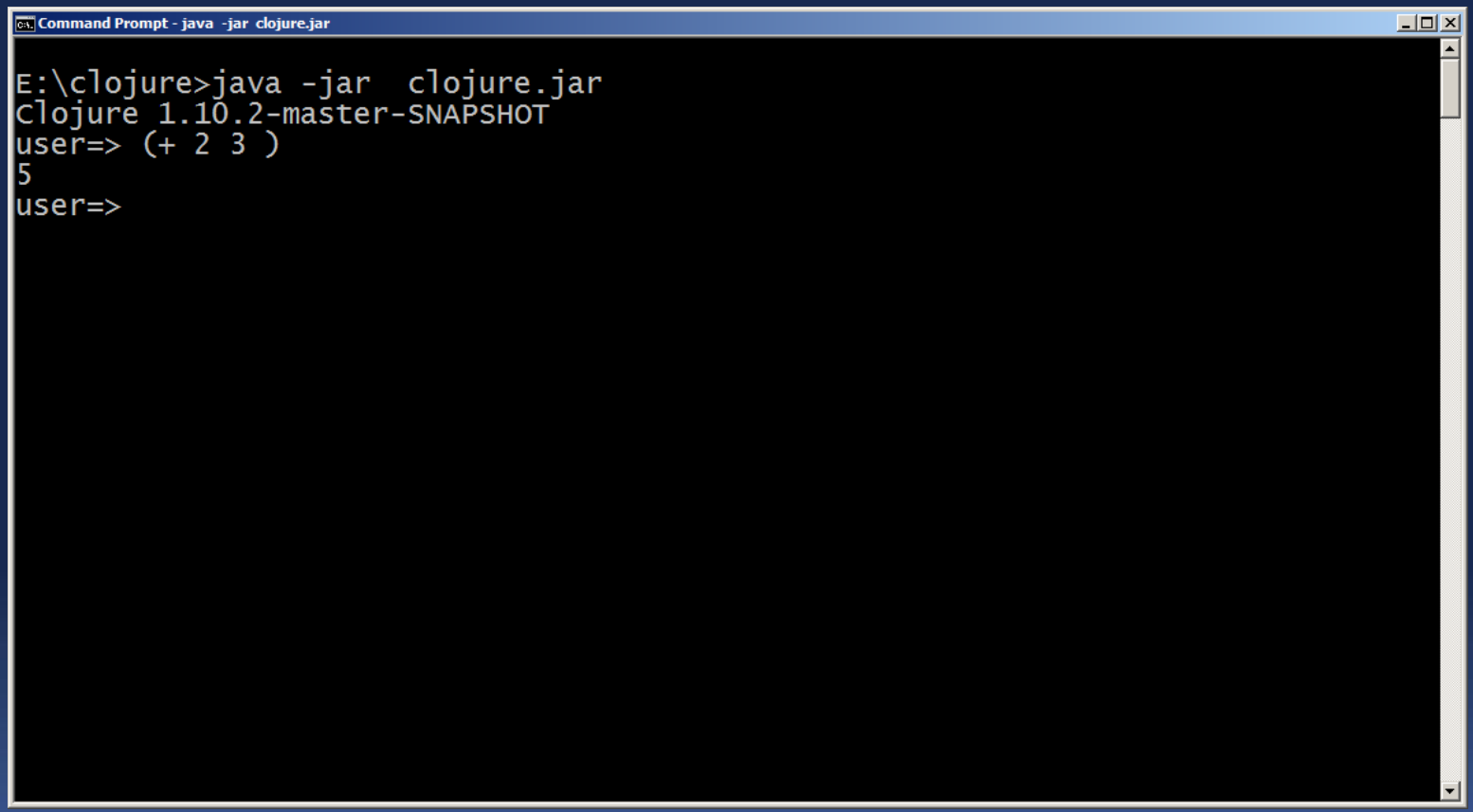
Iniciando REPL

java -jar clojure.jar



Testando REPL

java -jar clojure.jar



Clojure installer e CLI tools

Clojure installer e CLI tools

Clojure installer and CLI tools

Clojure provides command line tools that can be used to start a Clojure repl, use Clojure and Java libraries, and start Clojure programs. See the changelog for version information.

After following these installation instructions, you should be able to use the `clj` or `clojure` command to start a Clojure repl.

Installation on Windows

An early release version of clj on Windows is available at clj on Windows. Please provide feedback at https://clojure.atlassian.net/projects/TDEPS.

clj on Windows

clj on Windows

Alex Miller edited this page 5 days ago · 37 revisions

Currently, `clj` on Windows is in an alpha state. Please try it and provide feedback in the TDEPS jira or on #clj-on-windows room on Clojurians slack.

Install

Make sure PowerShell 5 (or later, include PowerShell Core) and .NET Core SDK 2.1+ or .NET Framework 4.5+ (or later) are installed. Then run:

```
Invoke-Expression (New-Object System.Net.WebClient).DownloadString('https://download.clojure.org/install/win-install-1.10.1.561.ps1')
```

clj on Windows

Alternatively, download the latest version of the installer and run the downloaded copy:

- https://download.clojure.org/install/win-install-1.10.1.561.ps1

When you run the installer, you will be prompted with several possible install locations:

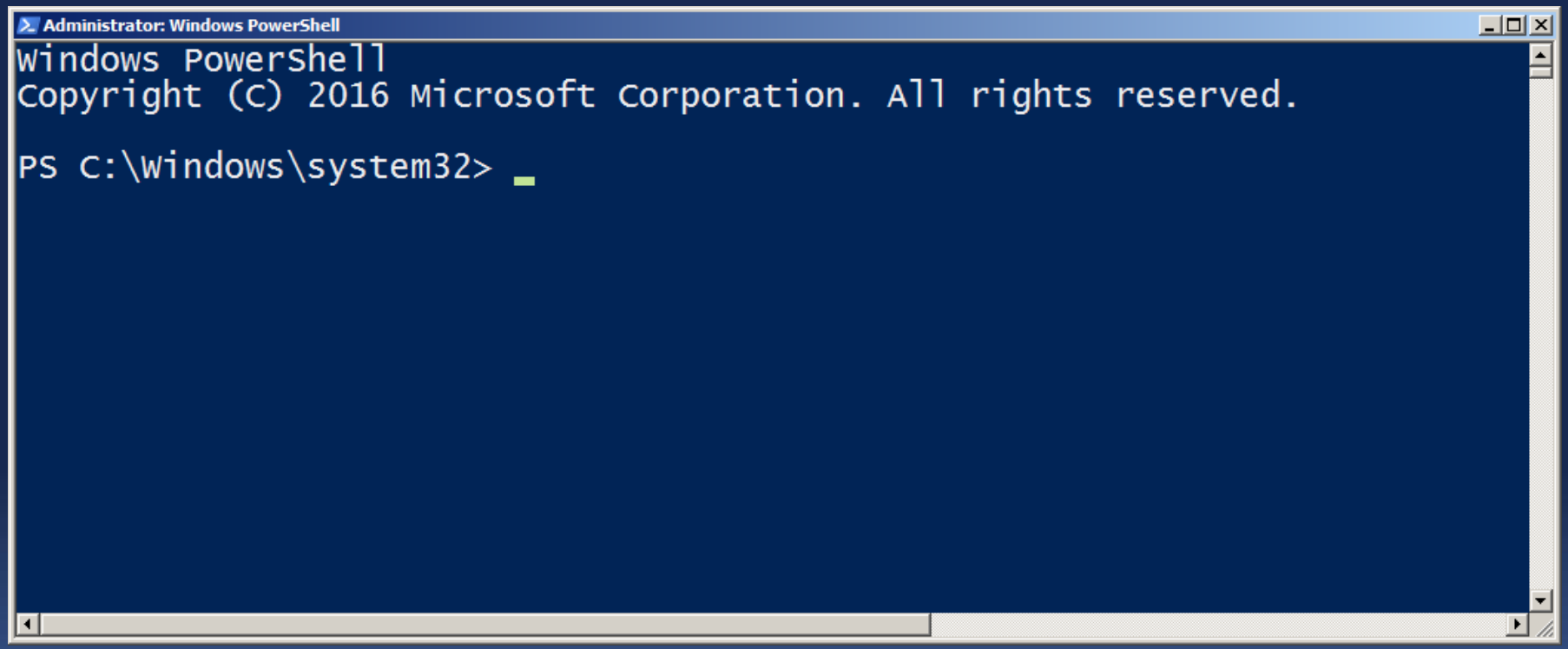
```
PS Y:\Downloads> .\win-install-1.10.1.561.ps1
Downloading Clojure tools
WARNING: Clojure will install as a module in your PowerShell module path.

Possible install locations:
  1) \\Drive\Home\Documents\WindowsPowerShell\Modules
  2) C:\Program Files\WindowsPowerShell\Modules
  3) C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules\
Enter number of preferred install location: 1

Cleaning up existing install
Installing PowerShell module
Removing download
Clojure now installed. Use "clj -h" for help.
```

When choosing which location to install consider these tradeoffs:

- #1 can be installed without admin privileges but will create a directory in Documents
- #2 and 3 should probably be run only if you have admin privileges

clj on Windows

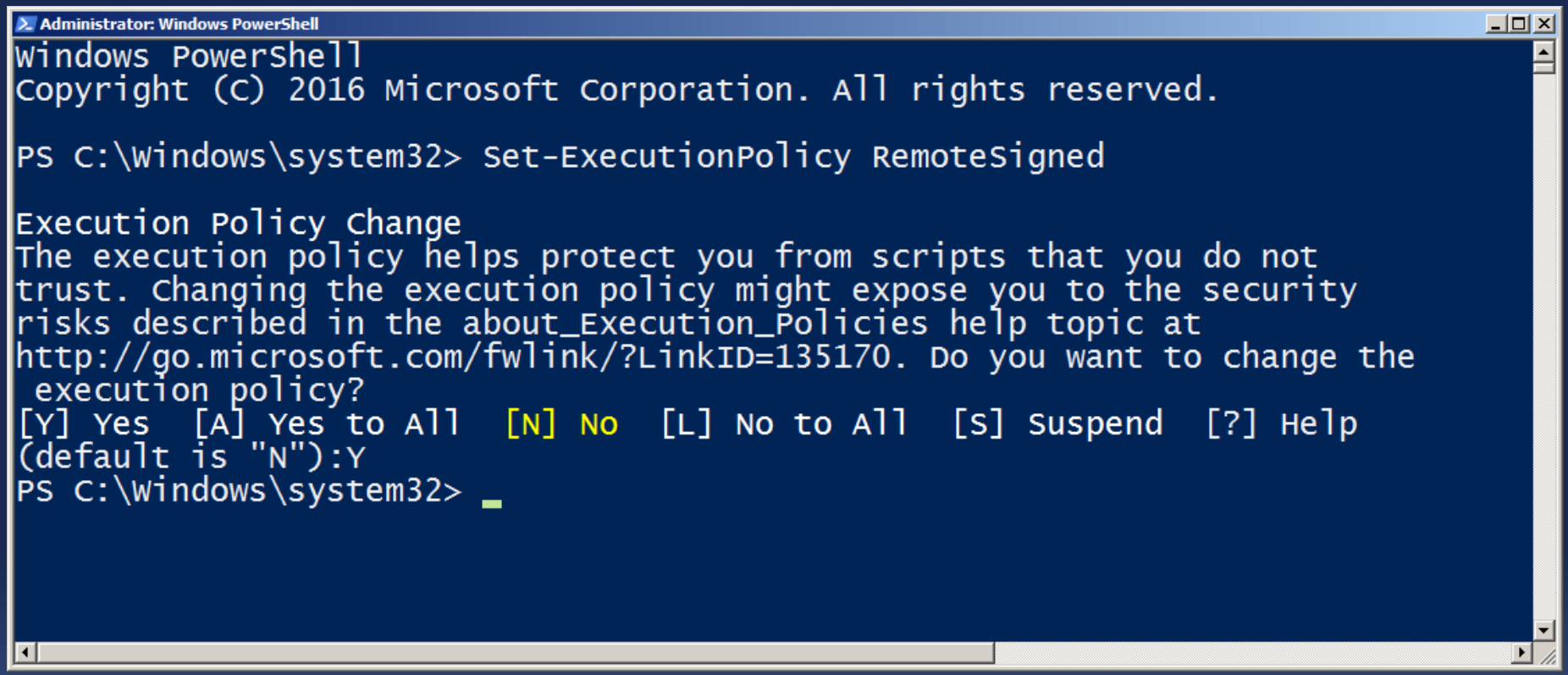
- ✓ A instalação será feita com Powershell versão 5
- ✓ Esteja certo de que **PowerShell 5** ou superior esteja instalado em sua máquina;



Configurações de segurança - Powershell

Habilitar PowerShell para execução

- ✓ No prompt do PowerShell: **Set-ExecutionPolicy RemoteSigned**



Habilitar PowerShell para execução

- ✓ No prompt do PowerShell: (em modo administrador)

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

Fix for PowerShell Script Not Digitally Signed

When you run a .ps1 PowerShell script you might get the message saying "**.ps1 is not digitally signed. The script will not execute on the system.**"

To fix it you have to run the command below to run Set-ExecutionPolicy and change the Execution Policy setting.

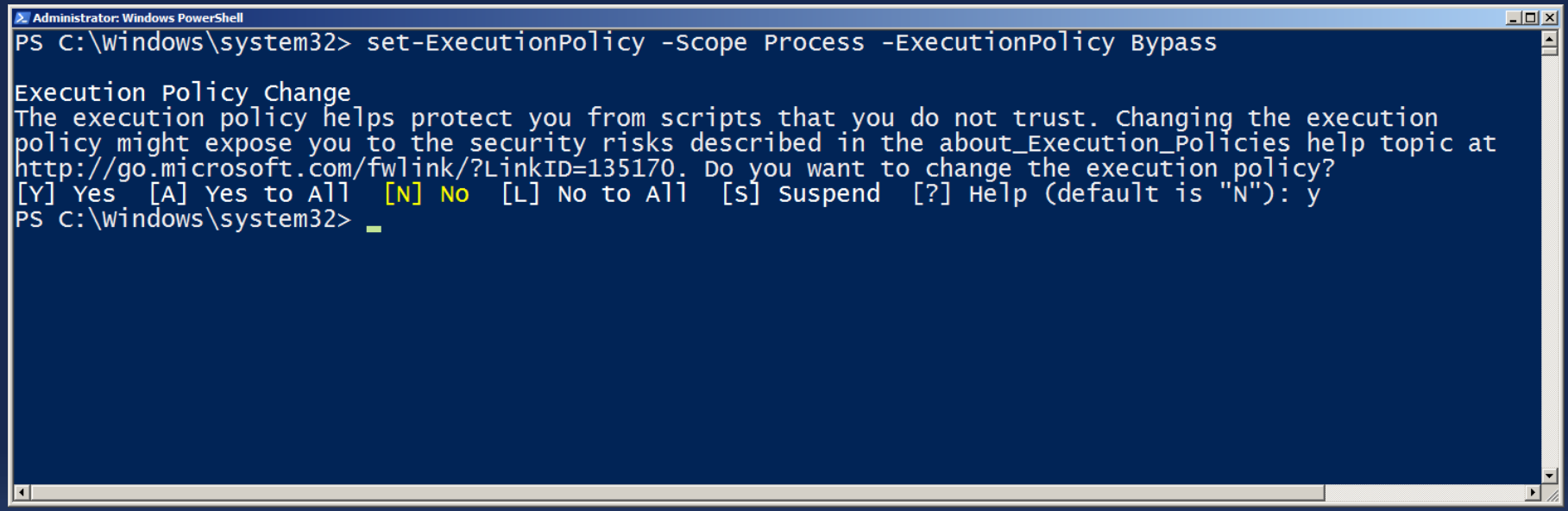
```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

This command sets the execution policy to bypass for only the current PowerShell session after the window is closed, the next PowerShell session will open running with the default execution policy. "Bypass" means nothing is blocked and no warnings, prompts, or messages will be displayed.

Habilitar PowerShell para execução

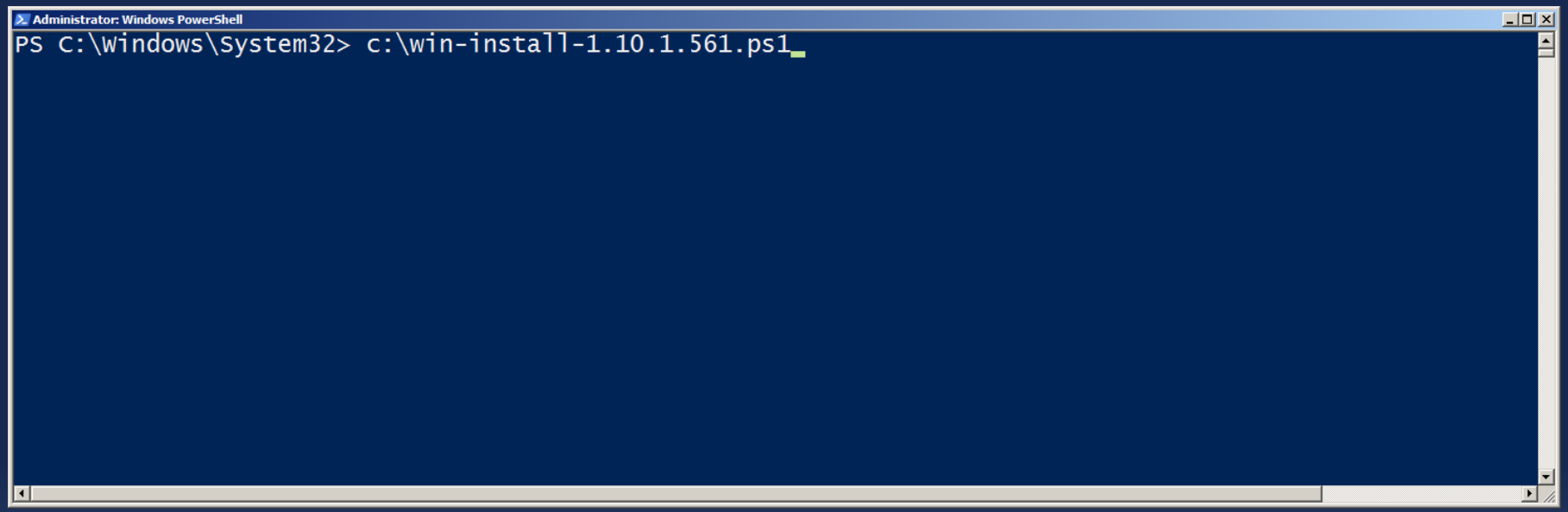
- ✓ No prompt do PowerShell: (em modo administrador)

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```



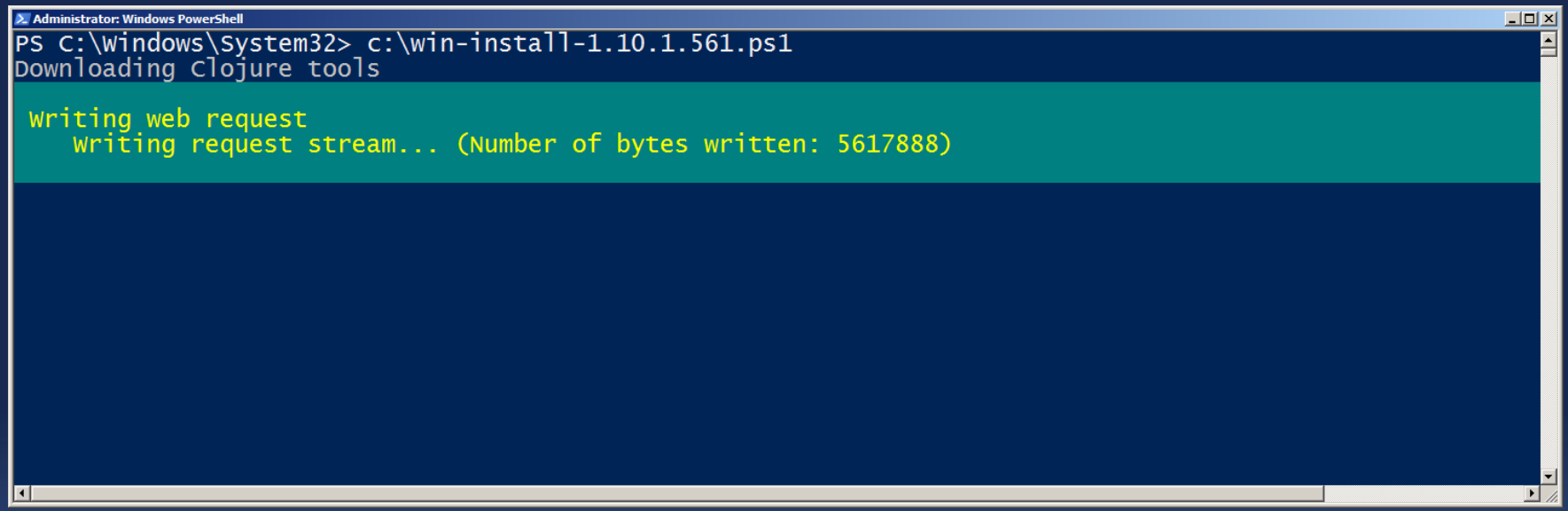
clj on Windows

✓Após baixar o instalador do clojure (**win-install-1.10.1.561.ps1**), executá-lo sob o Powershell, em modo administrador.



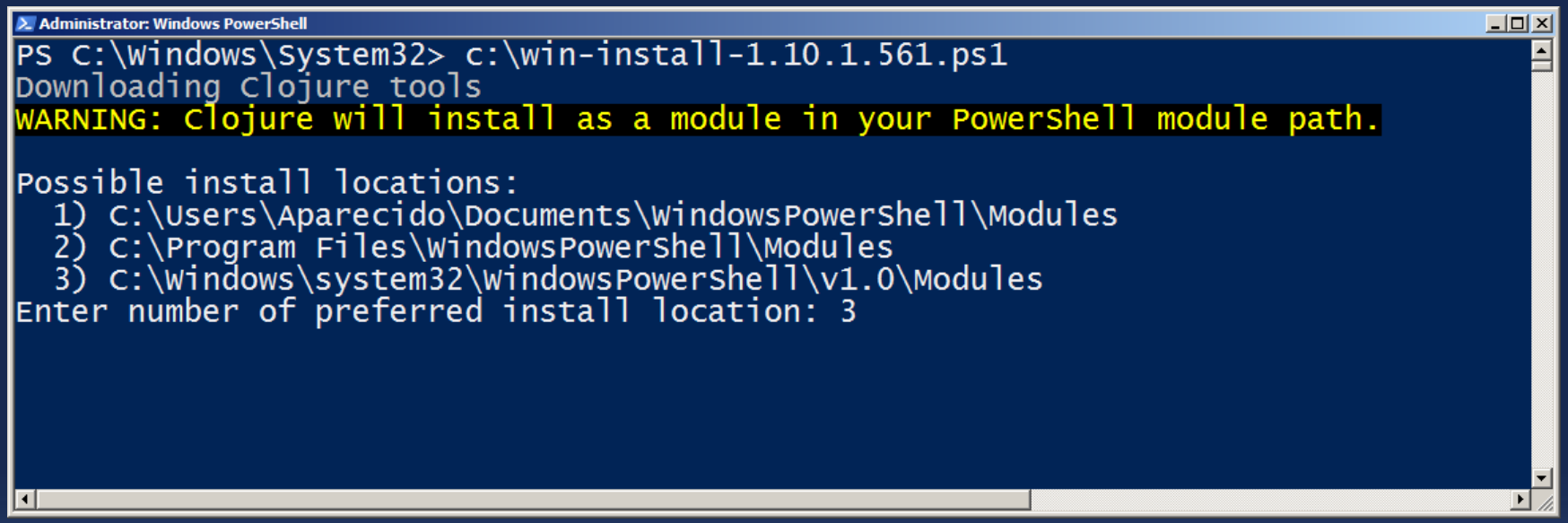
clj on Windows

✓Após baixar o instalador do clojure (**win-install-1.10.1.561.ps1**), executá-lo sob o Powershell, em modo administrador.



clj on Windows

- ✓ Definir opção 3 => c:\Windows\system32/WindowsPowerShell.

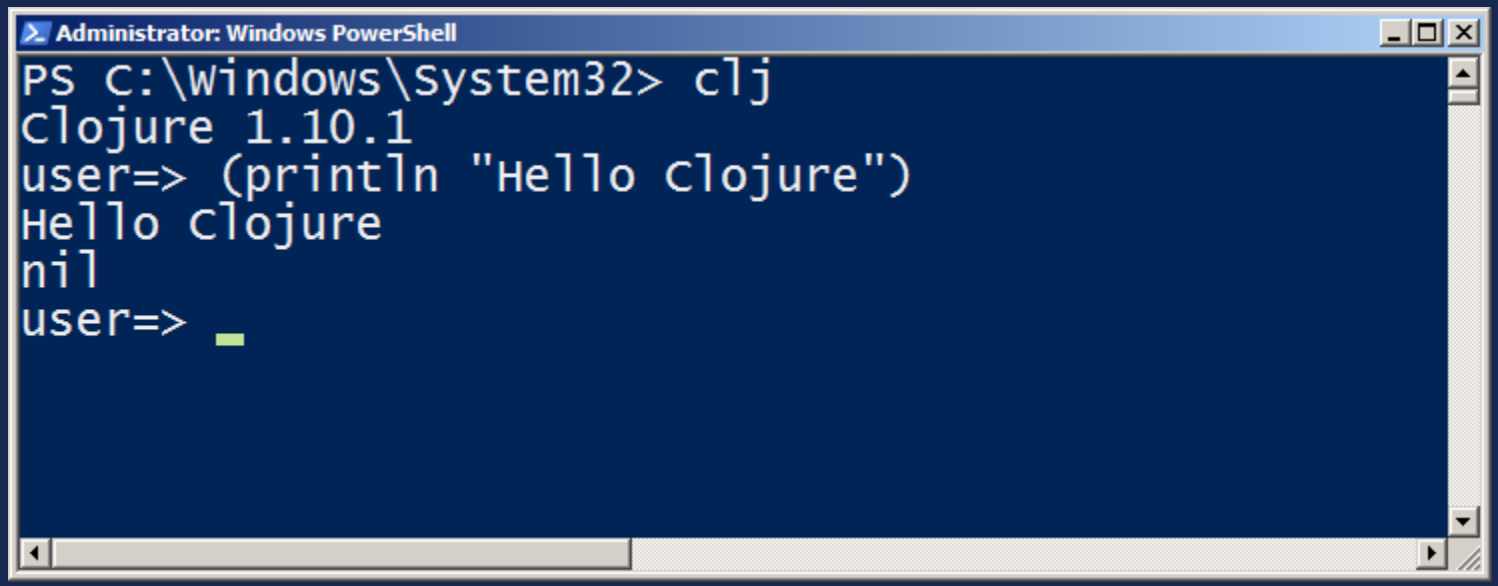


clj on Windows

- ✓ Clojure now installed. Use "clj-h" for help.

```
Administrator: Windows PowerShell
Downloading Clojure tools
WARNING: Clojure will install as a module in your PowerShell module path.

Possible install locations:
  1) C:\Users\Aparecido\Documents\WindowsPowerShell\Modules
  2) C:\Program Files\WindowsPowerShell\Modules
  3) C:\Windows\system32\WindowsPowerShell\v1.0\Modules
Enter number of preferred install location: 3

Cleaning up existing install
Installing PowerShell module
Removing download
Clojure now installed. Use "clj -h" for help.
PS C:\Windows\System32>
```

clj on Windows



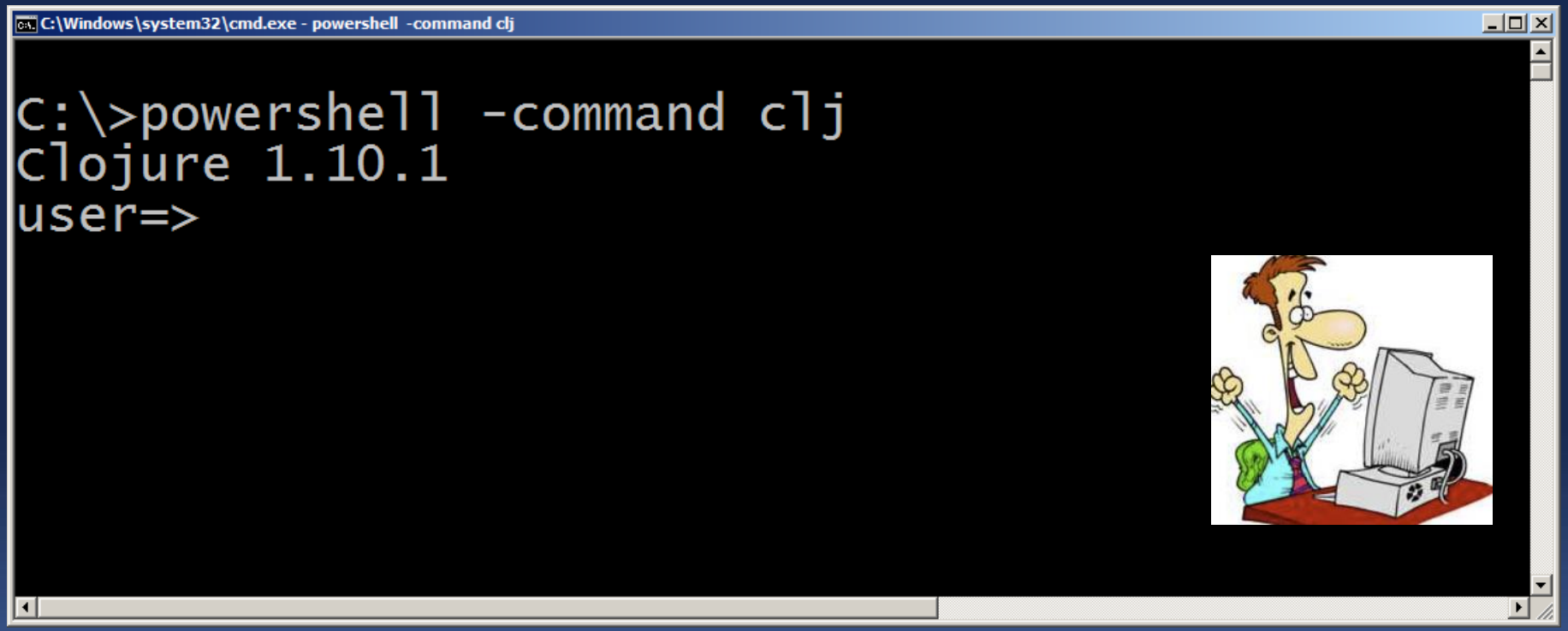
Executando clj

- ✓ Sob **BASH**: powershell -command clj

```
MINGW64:/c/Users/Aparecido
Aparecido@Aparecido-PC MINGW64 ~
$ (powershell -command   clj)
Clojure 1.10.1
user=>
```

Ativando clj

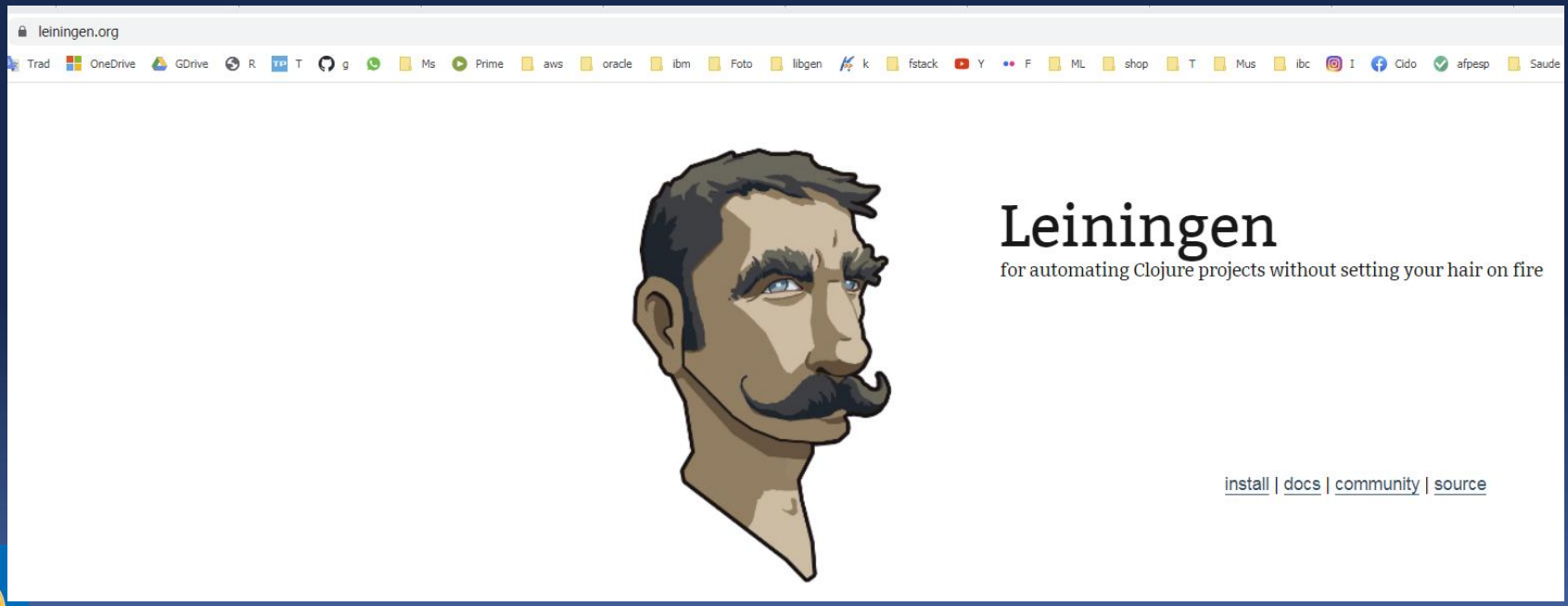
- ✓ Sob prompt do Windows: powershell -command clj



Instalação com Leiningen

Leiningen – www.leiningen.org

- ✓ Leiningen é a forma mais simples de se usar a Linguagem **Clojure**;
- ✓ Ferramenta para se trabalhar com projetos Clojure;
- ✓ Foco na automação do projeto e configuração declarativa, permite que se foque na criação do código;



Leiningen - Install

Install

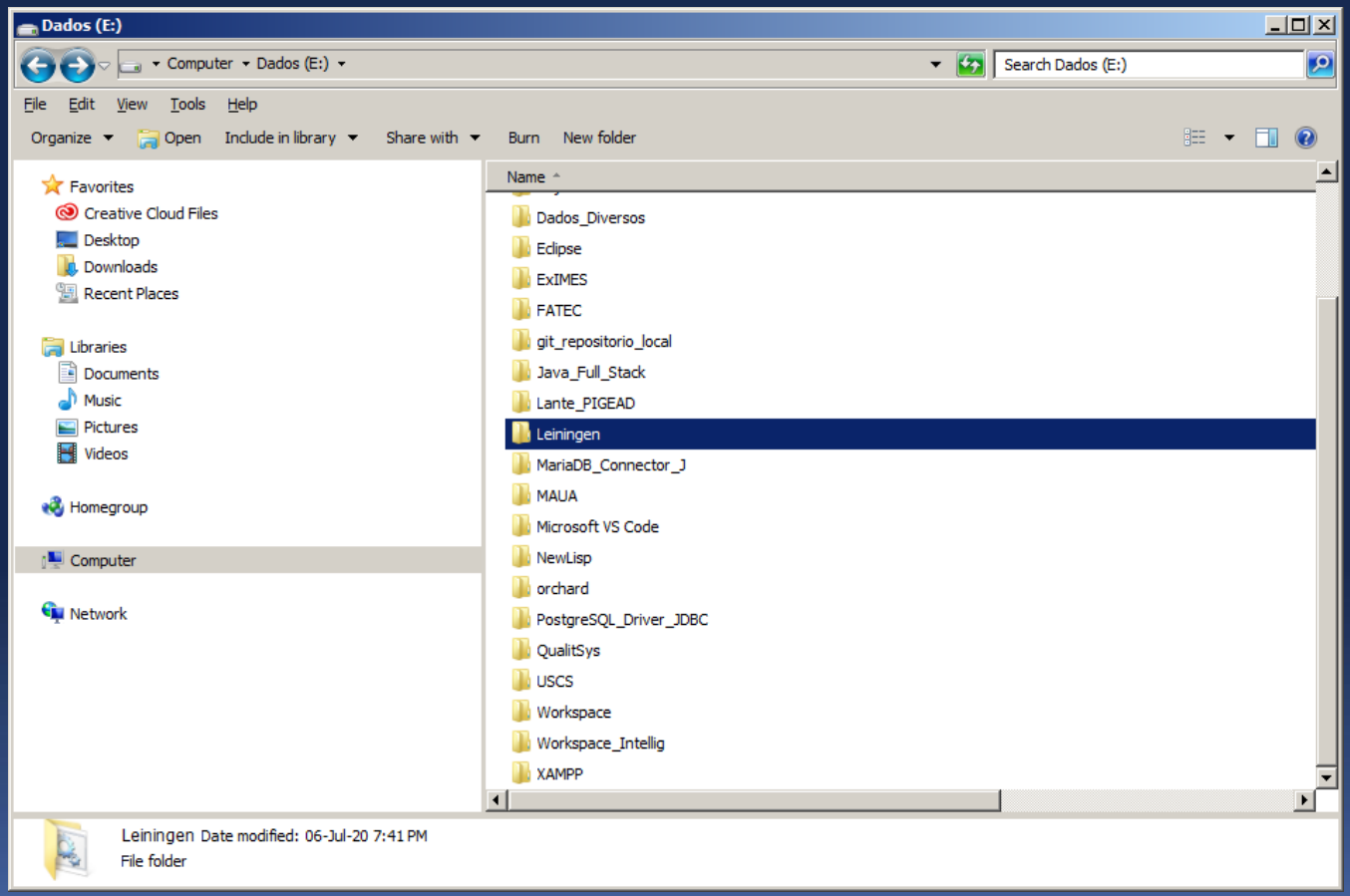
Leiningen and Clojure require Java. OpenJDK version 8 is recommended at this time.

1. Download the `lein` script (or on Windows `lein.bat`)
2. Place it on your `$PATH` where your shell can find it (eg. `~/bin`)
3. Set it to be executable (`chmod a+x ~/bin/lein`)
4. Run it (`lein`) and it will download the self-install package

You can check your package manager as well..

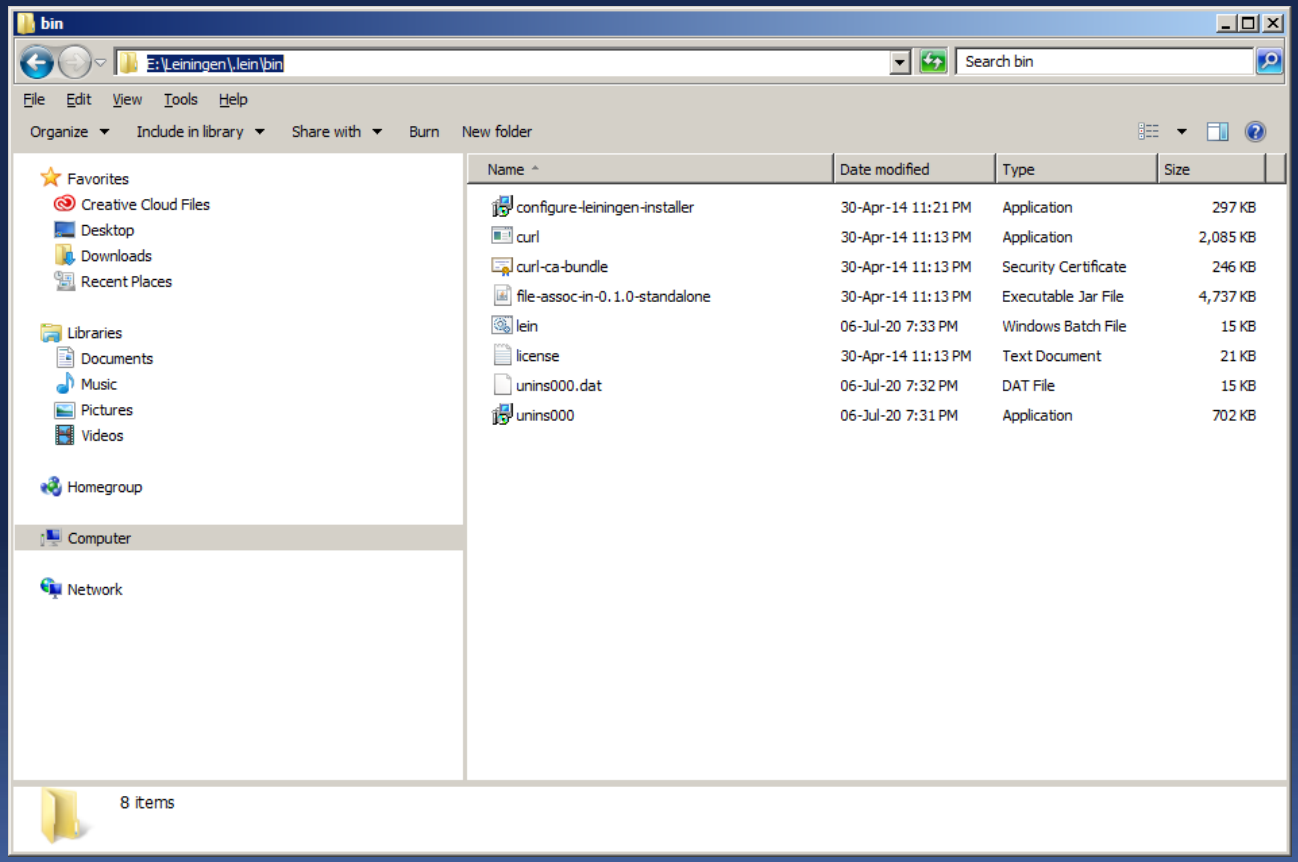
Leiningen - Install

- ✓ Criar pasta para Leiningen, download e execução do instalador



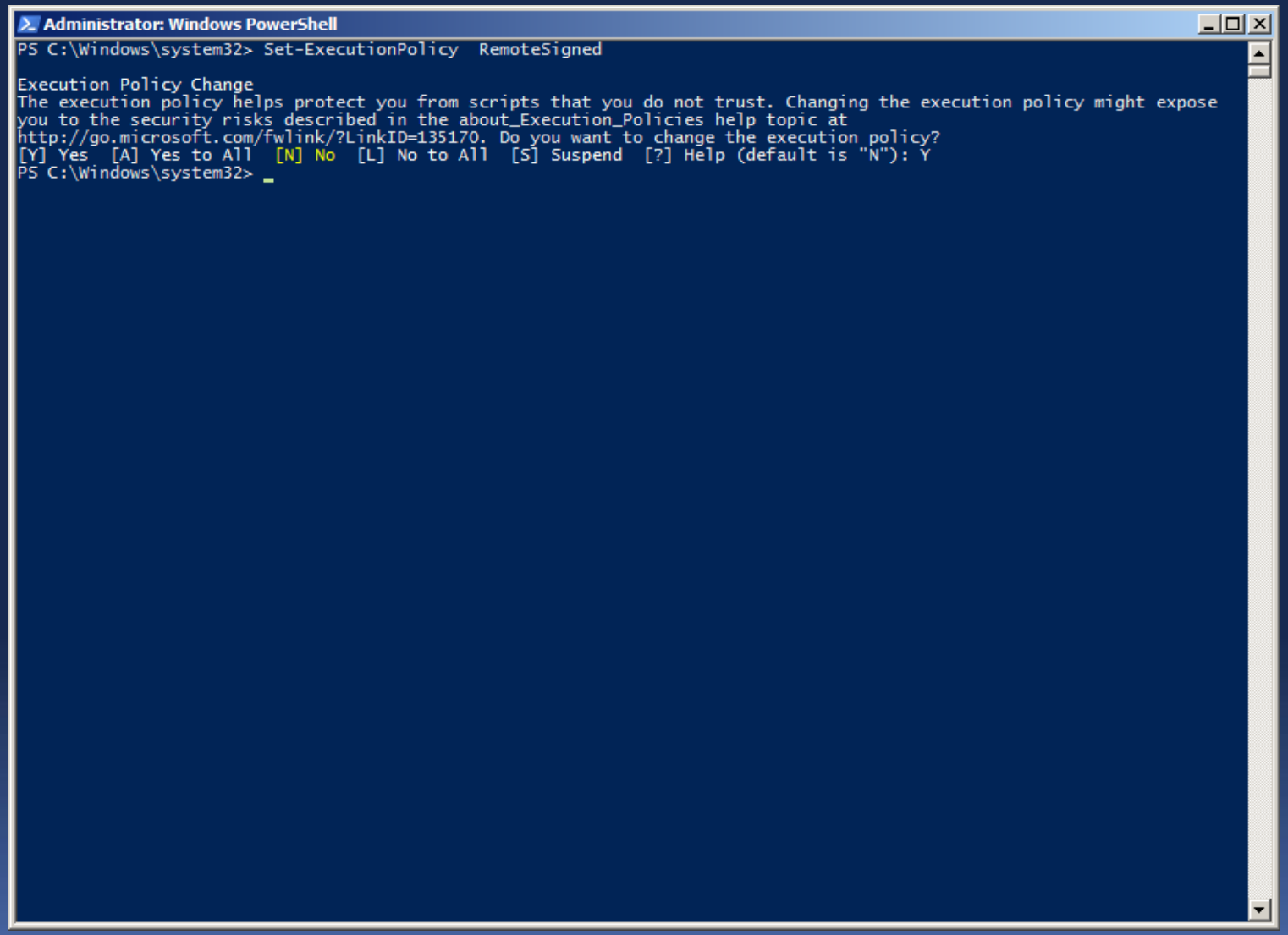
Leiningen - Install

- ✓ Criar pasta para **Leiningen**, download e execução do instalador;
- ✓ Executar o bat file: **lein.bat** (sob Powershell)



Habilitar PowerShell para execução

- ✓ No prompt do PowerShell: **Set-ExecutionPolicy RemoteSigned**



Habilitar PowerShell para execução

- ✓ No prompt do PowerShell:

  Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

Fix for PowerShell Script Not Digitally Signed

Caio Moreno Follow

Jul 15, 2019 · 1 min read

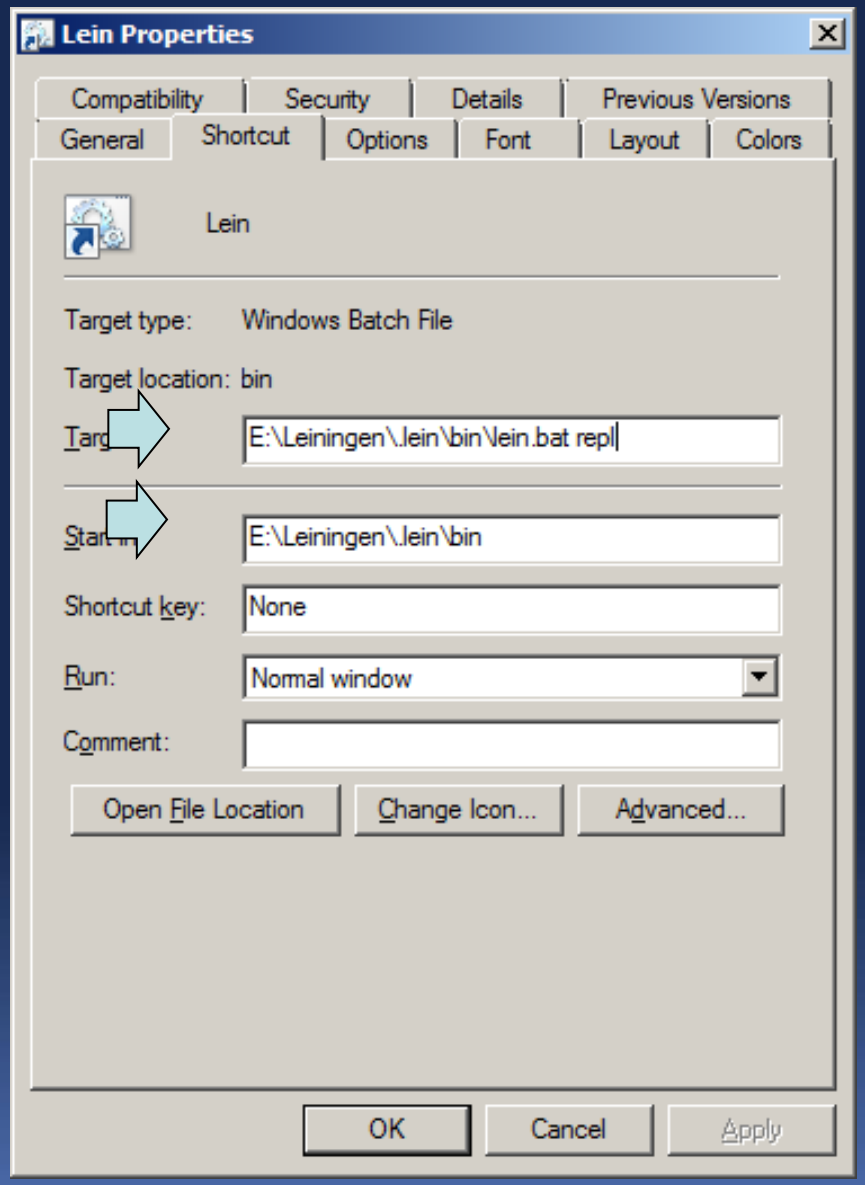
When you run a .ps1 PowerShell script you might get the message saying "**.ps1 is not digitally signed. The script will not execute on the system.**"

To fix it you have to run the command below to run Set-ExecutionPolicy and change the Execution Policy setting.

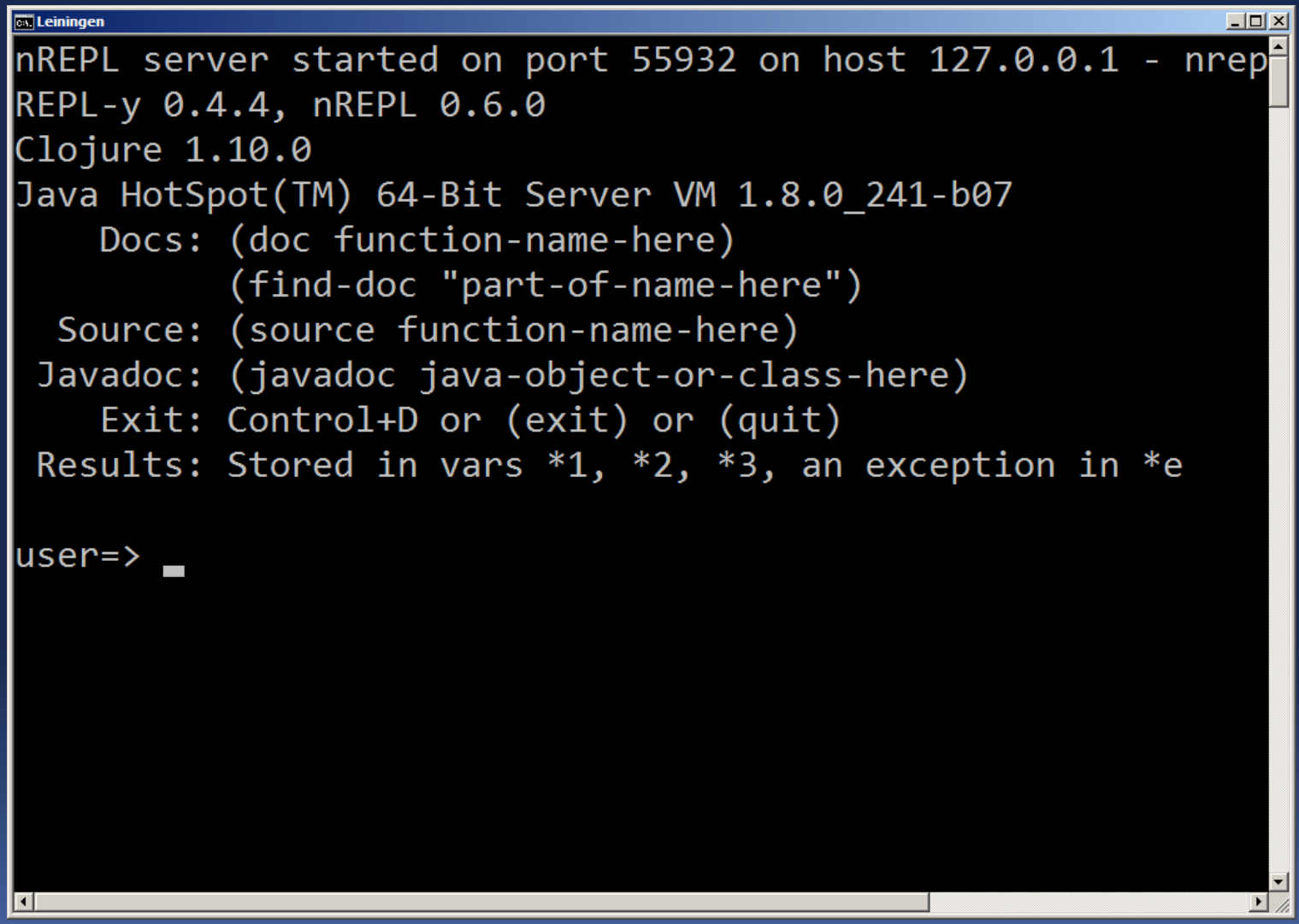
```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

This command sets the execution policy to bypass for only the current PowerShell session after the window is closed, the next PowerShell session will open running with the default execution policy. "Bypass" means nothing is blocked and no warnings, prompts, or messages will be displayed.

Subindo nREPL

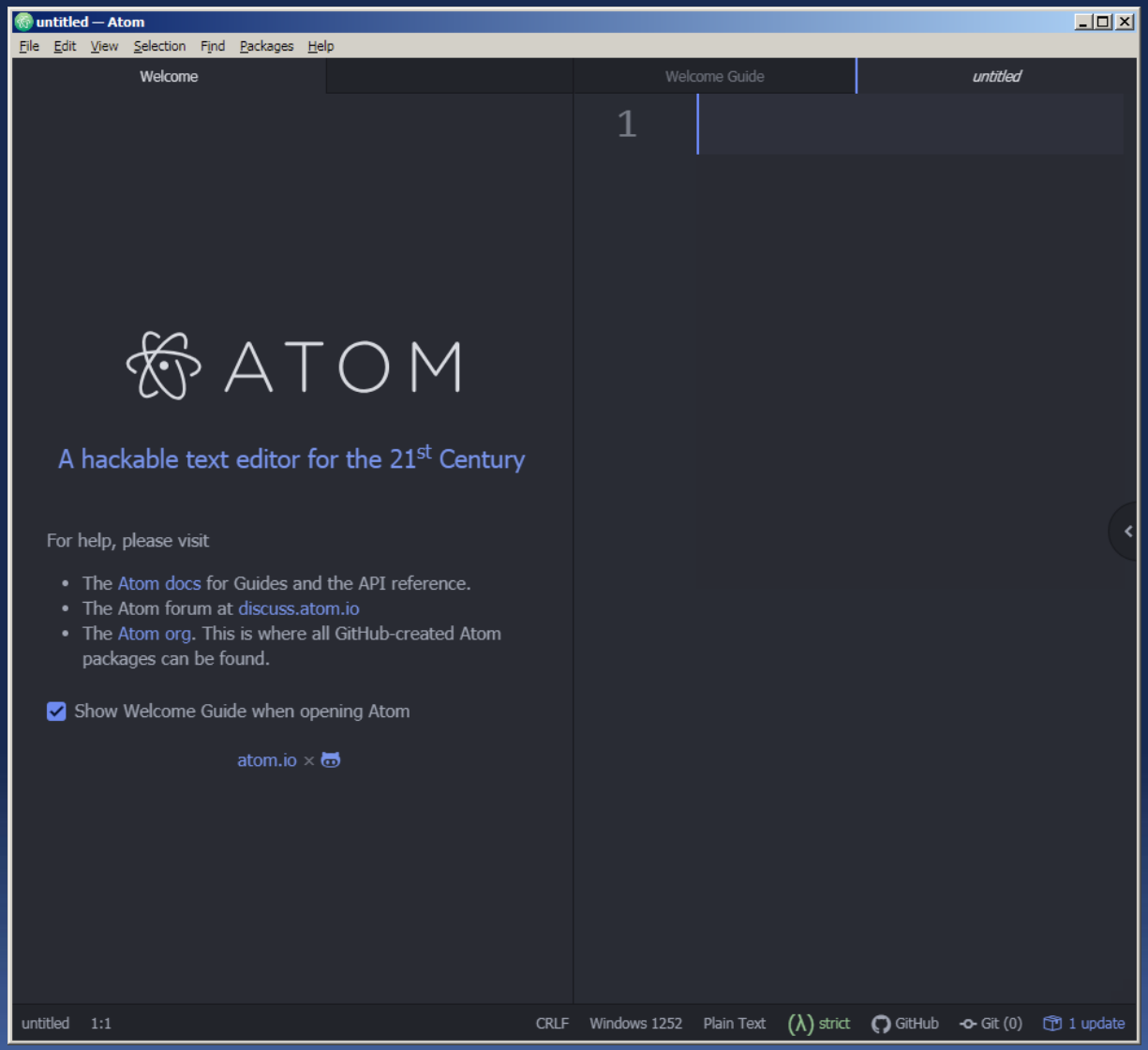


Leiningen - nREPL



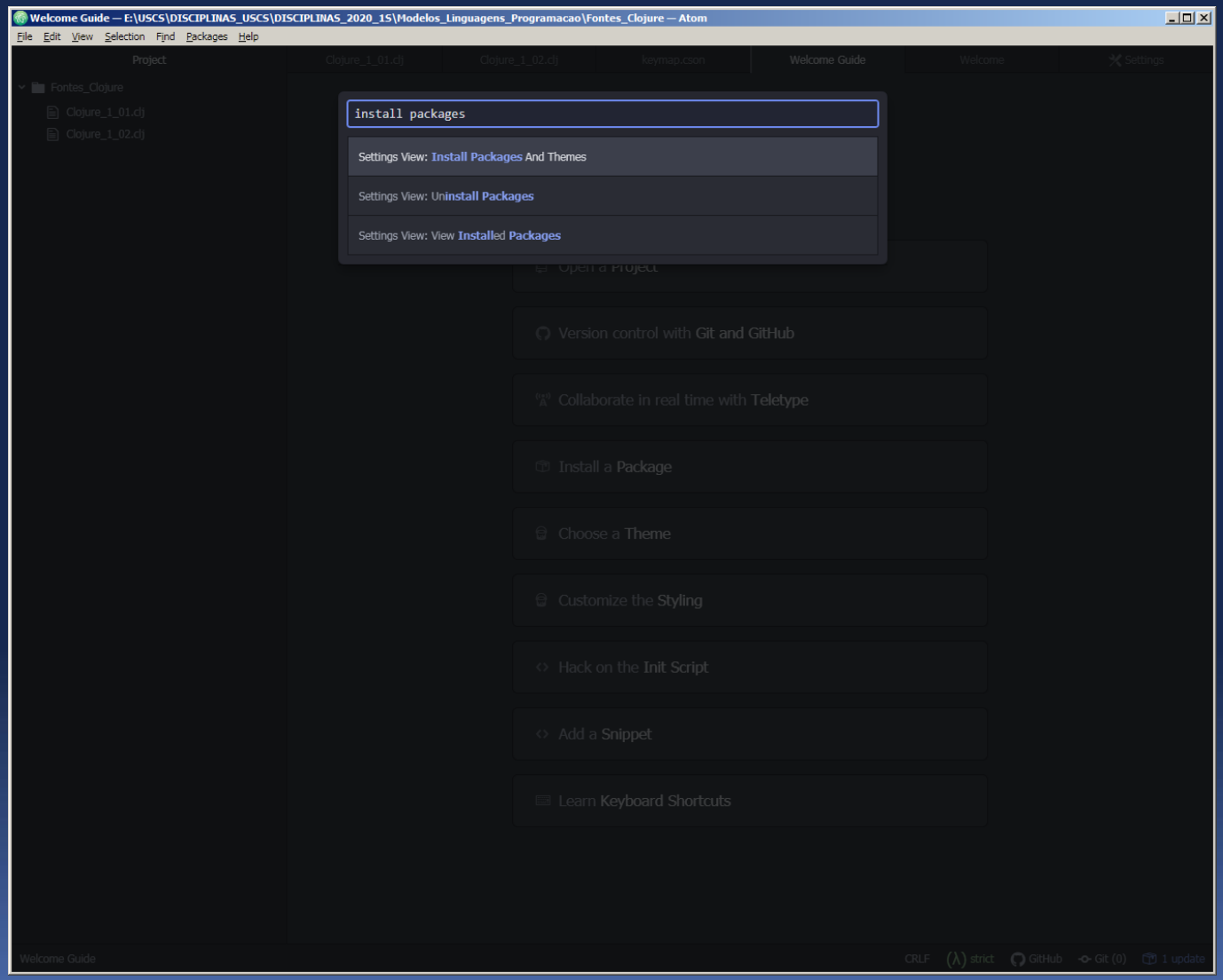
Desenvolvimento Clojure com a IDE Atom

Código Clojure no Atom



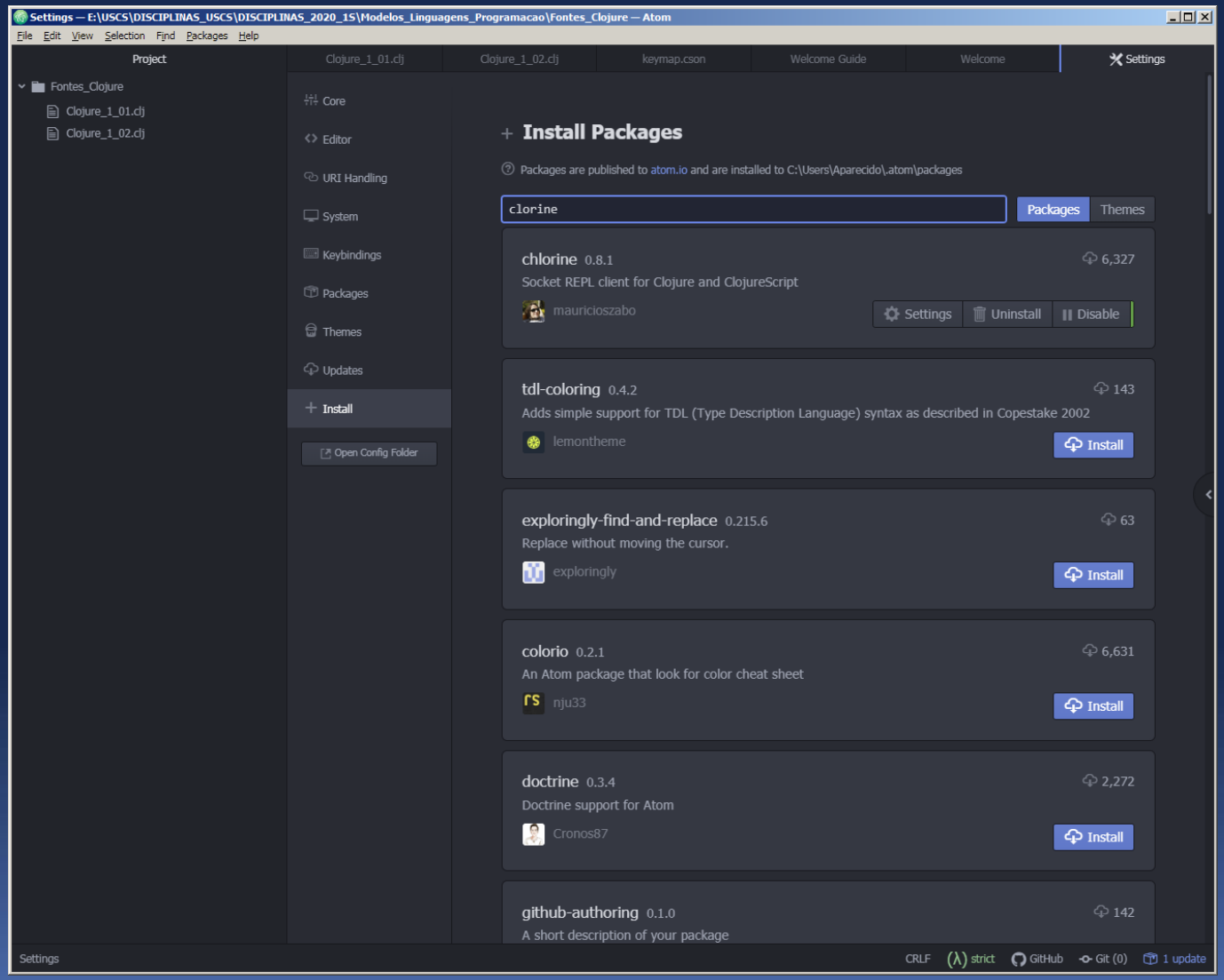
Atom – Package Chlorine

✓ Ctrl + Shift + p > Install Packages and Themes



Atom – Package Chlorine

✓Instalar => package Chlorine



Chlorine – Configuração de Hot Keys

- ✓ Instalar => package **Chlorine**

  - ✓ Ao se teclar **CTRL+SHIFT+p** e procurar por "Open your keymap", Atom irá abrir um arquivo pra customizar hotkeys;

  - ✓ Nesse arquivo podem ser incluídas as configurações:

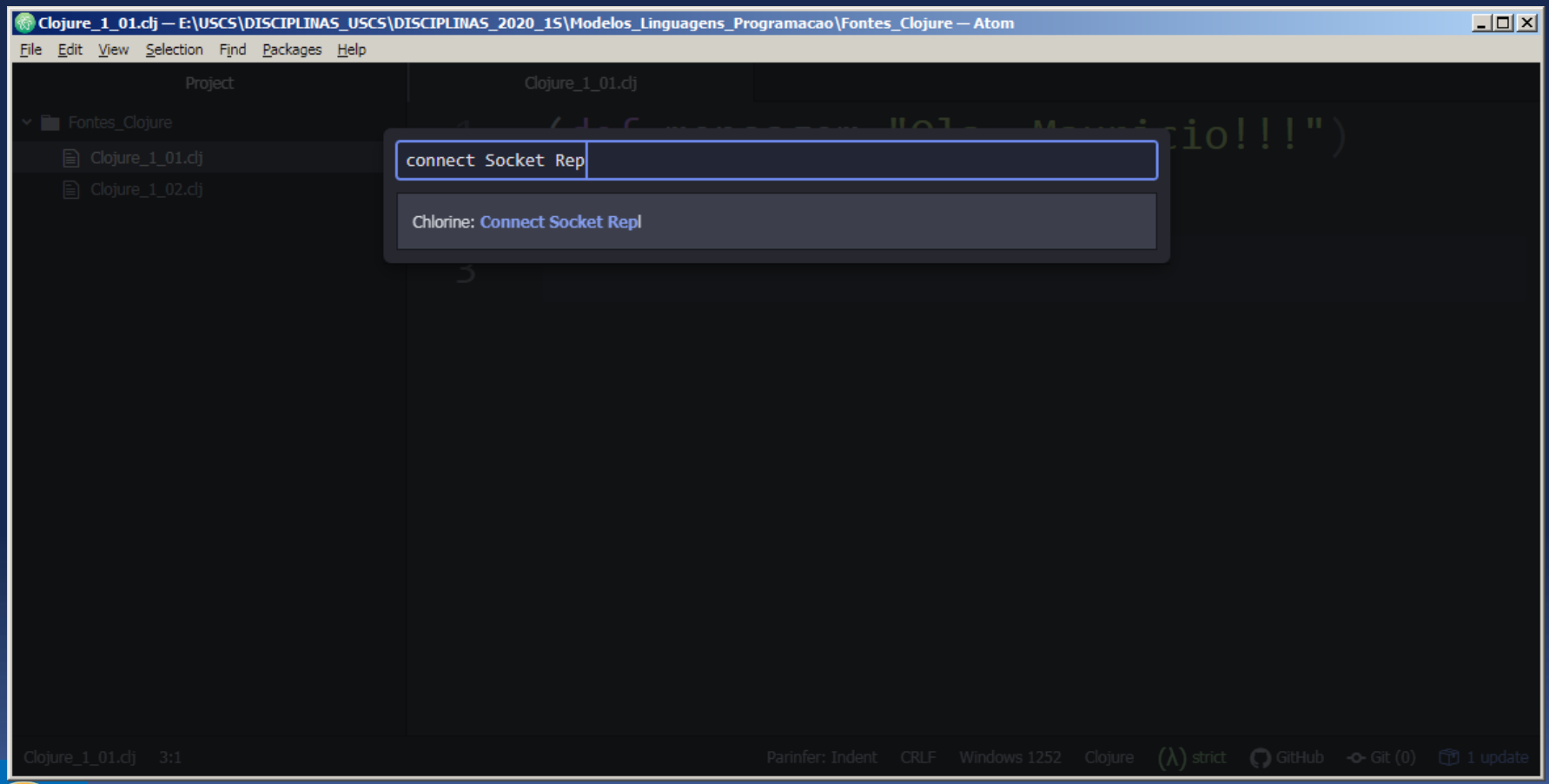
```
'atom-text-editor':
        'ctrl-k':               'chlorine:clear-console'
        'ctrl-shift-I':         'chlorine:load-file'
        'ctrl-shift-l':         'chlorine:clear-inline-results'
        'ctrl-shift-enter':     'chlorine:evaluate-block'
        'ctrl-enter':           'chlorine:evaluate-top-block'
        'ctrl-c':               'chlorine:break-evaluation'
        'ctrl-d':               'chlorine:doc-for-var'
```

Chlorine – Configuração de Hot Keys

```
# Atom Flight Manual:
# http://flight-manual.atom.io/using-atom/sections/basic-custom

'atom-text-editor':
  'ctrl-k':              'chlorine:clear-console'
  'ctrl-shift-I':        'chlorine:load-file'
  'ctrl-shift-L':        'chlorine:clear-inline-results'
  'ctrl-shift-enter':    'chlorine:evaluate-block'
  'ctrl-enter':          'chlorine:evaluate-top-block'
  'ctrl-c':              'chlorine:break-evaluation'
  'ctrl-d':              'chlorine:doc-for-var'
```

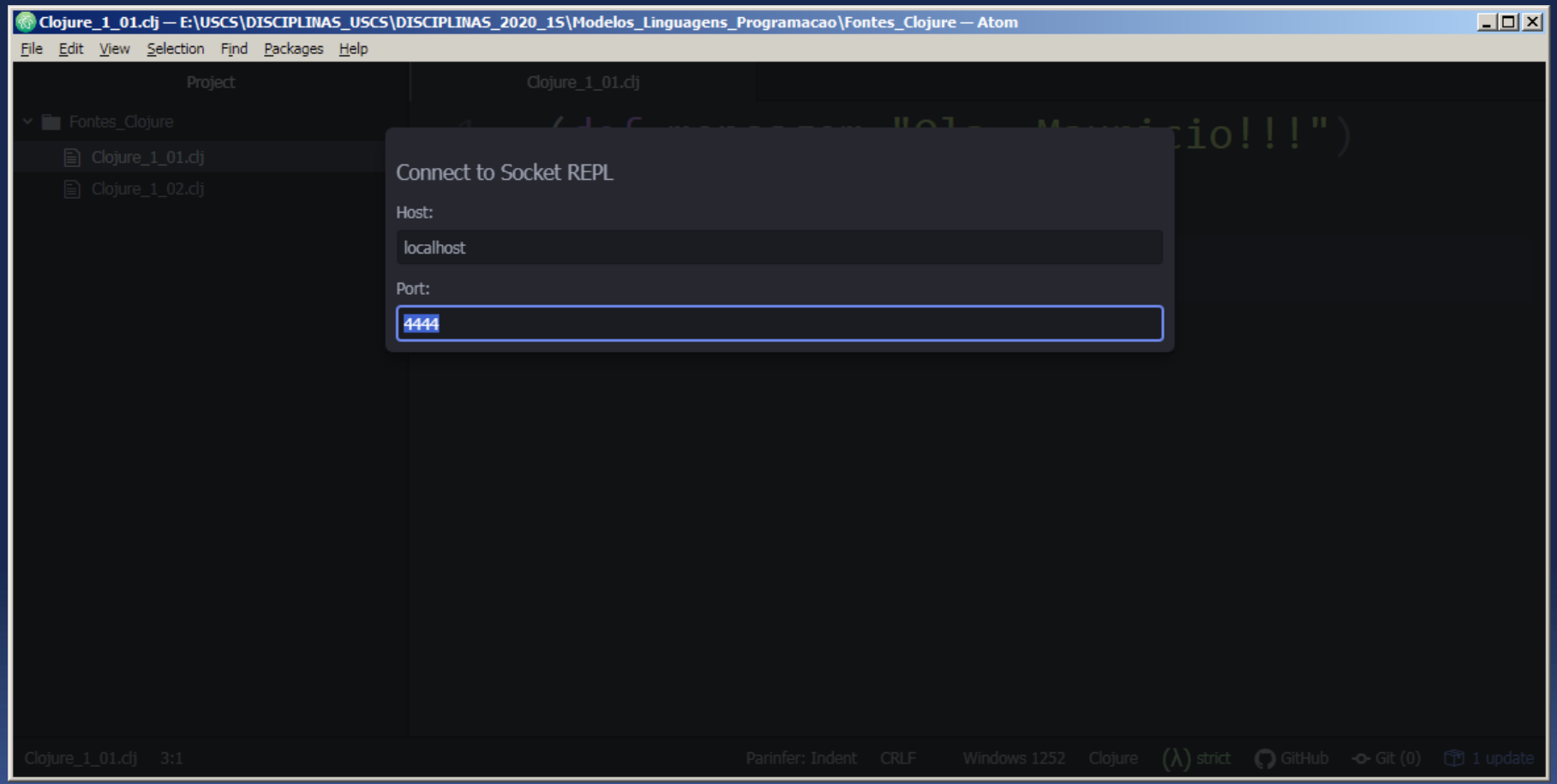
Atom – Package Chlorine

- ✓ Após instalação do Chlorine e subida do socket com REPL remoto
- ✓ **Ctrl + Shift + p** > Connect Socket Repl



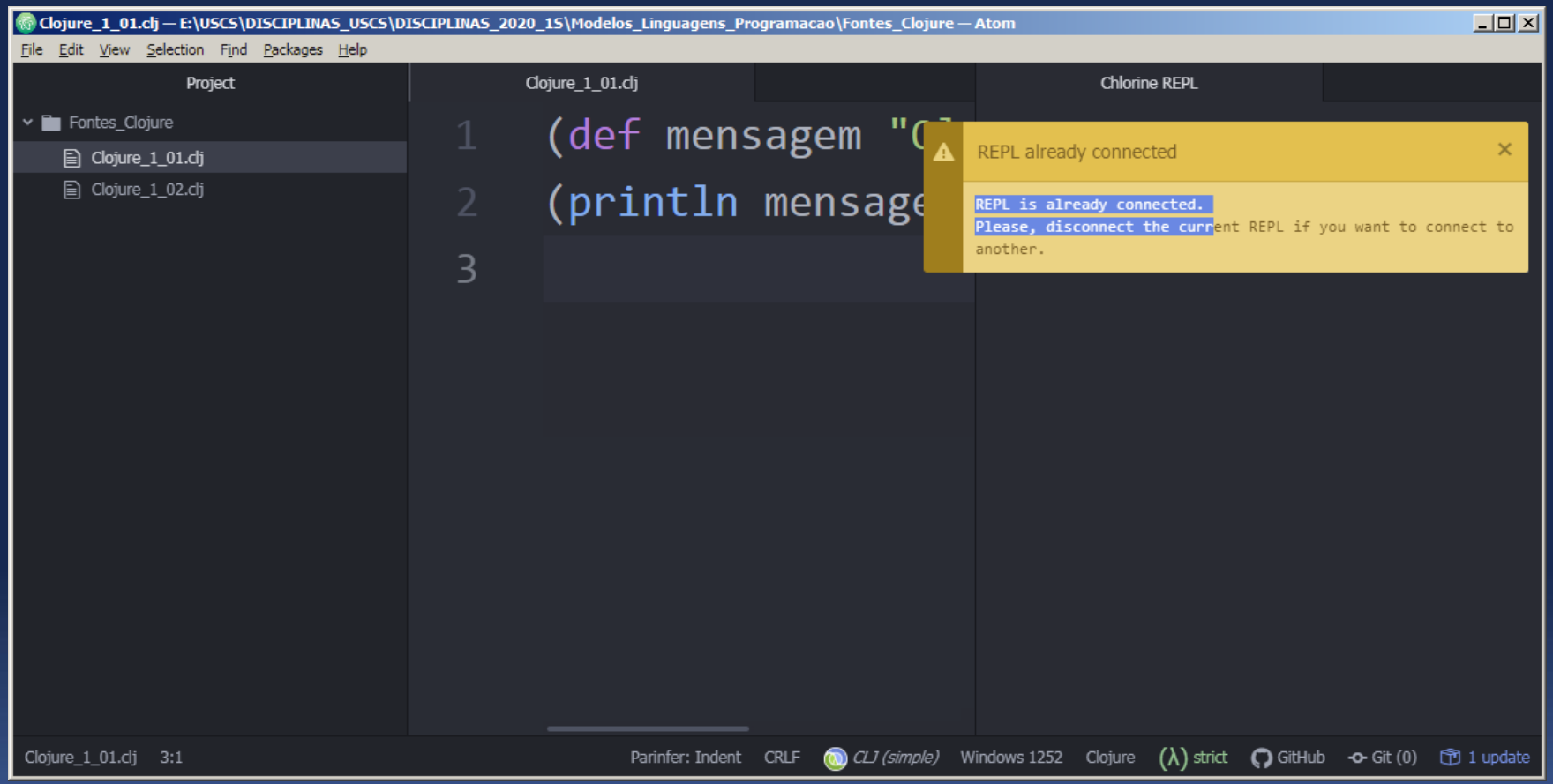
Atom – Package Chlorine

- ✓ Informar Host e porta onde o REPL remoto está sendo executado



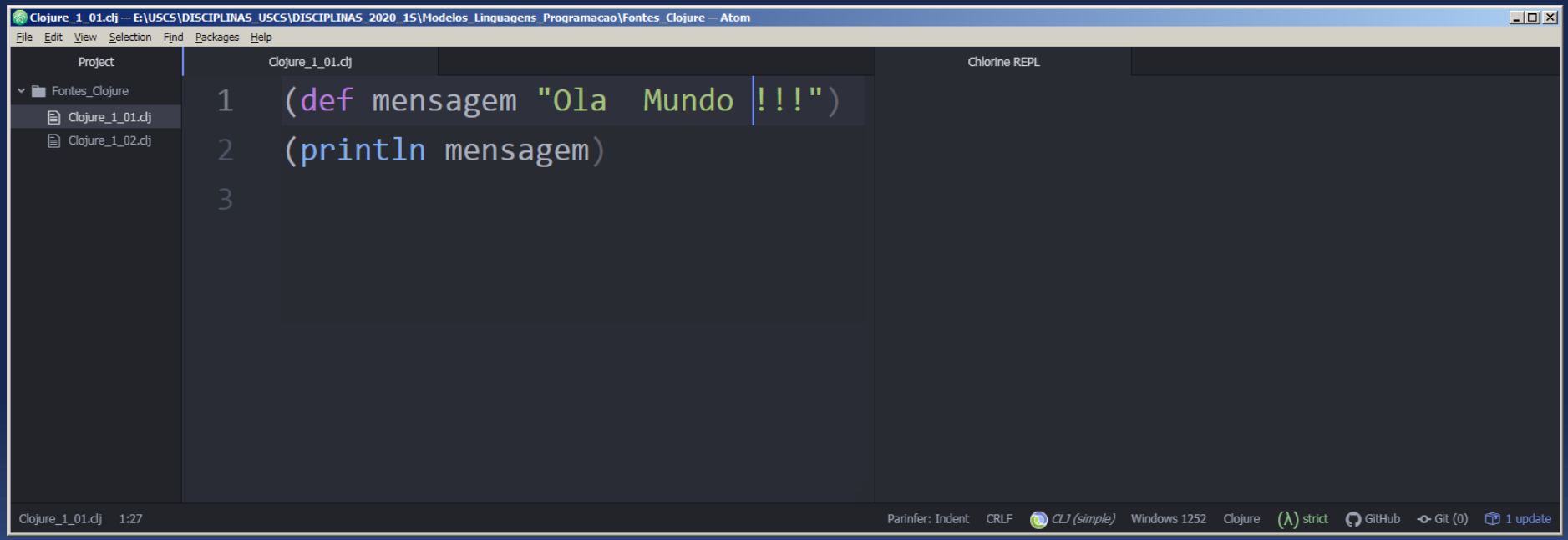
Atom – Package Chlorine

✓REPL connected !!!



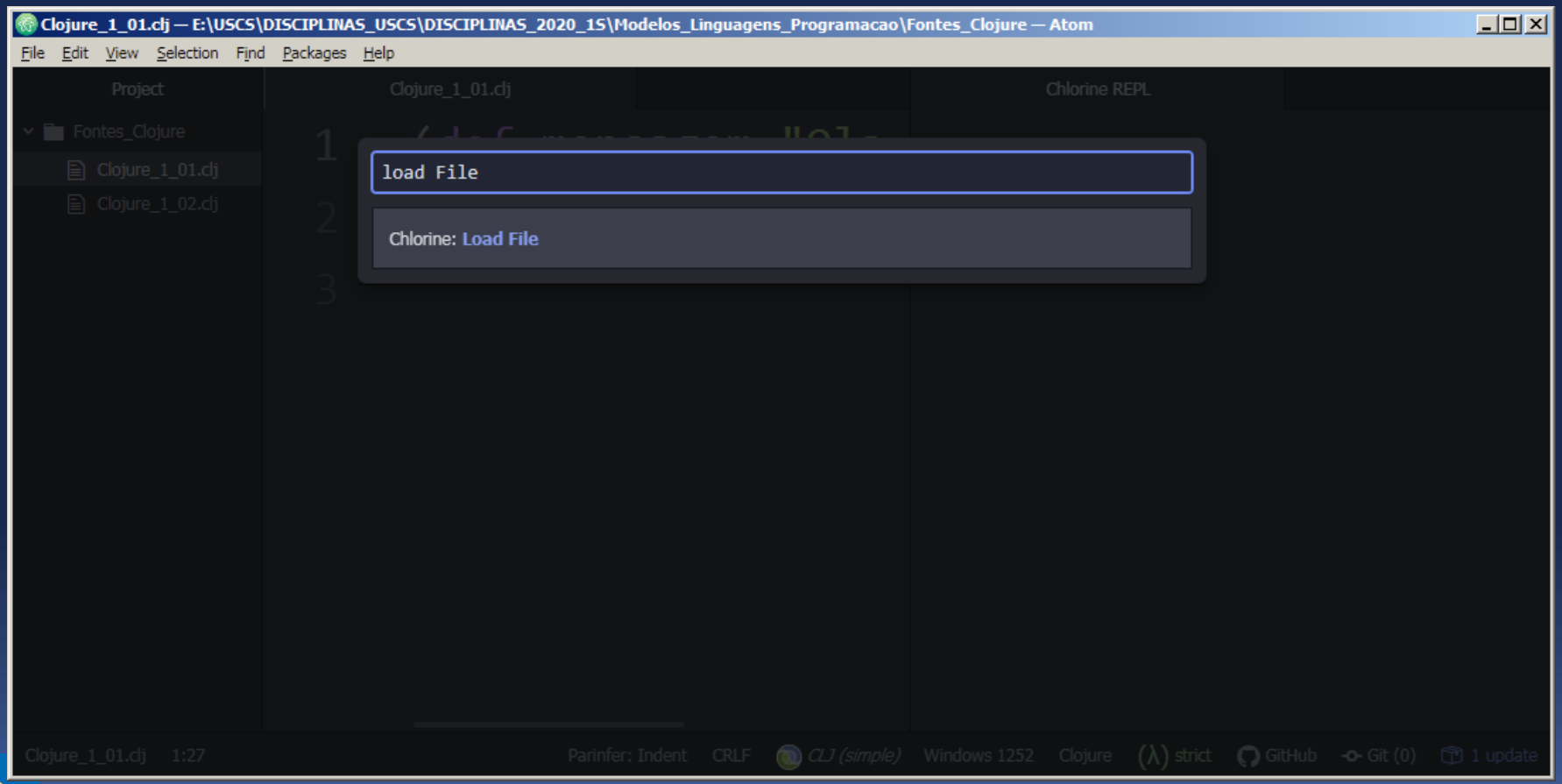
Atom – Package Chlorine

- ✓ Escrever código Clojure (extensão .clj)



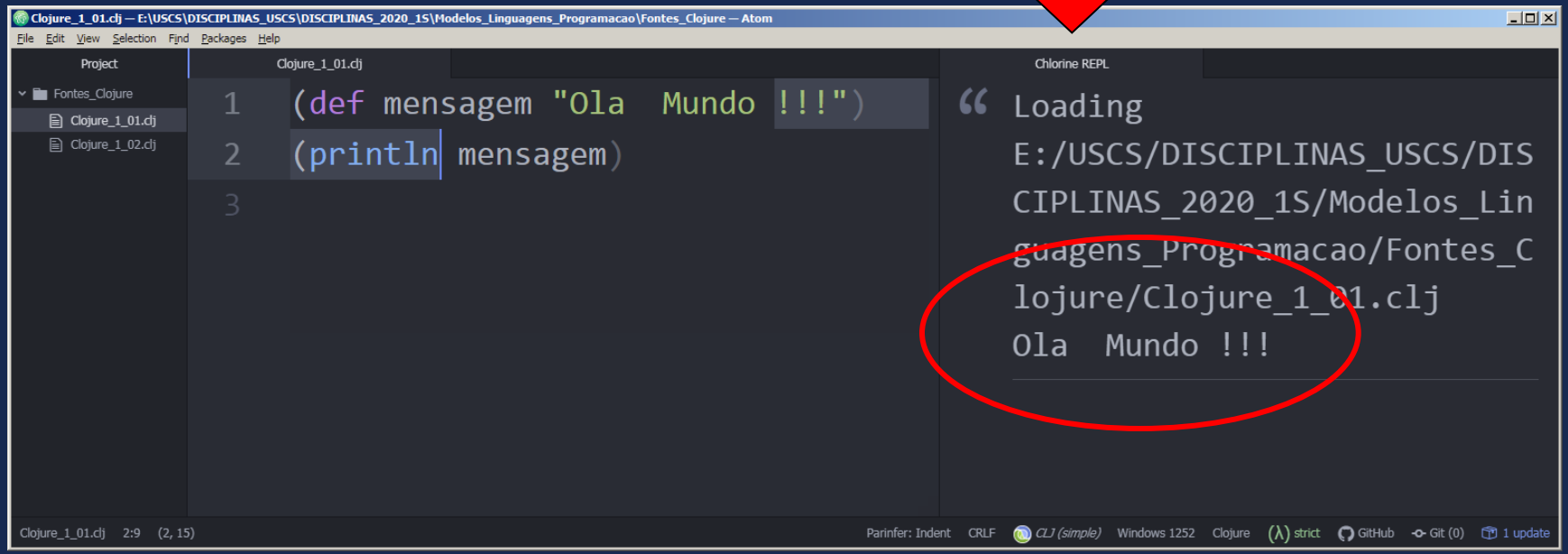
Atom – Package Chlorine

- ✓ Carregar código para Repl remoto
- ✓ Ctrl + Shift + p > load File



Atom – Package Chlorine

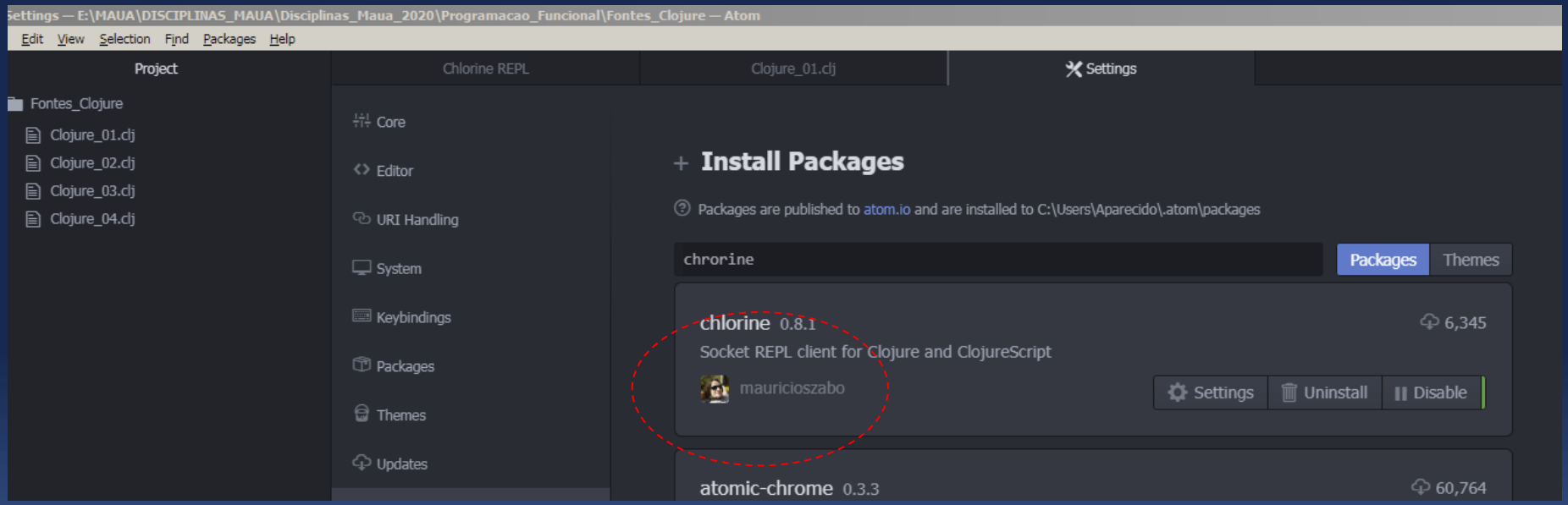
✓ Carga do código Clojure para o REPL



Atom – Package Chlorine + nREPL (Leiningen)

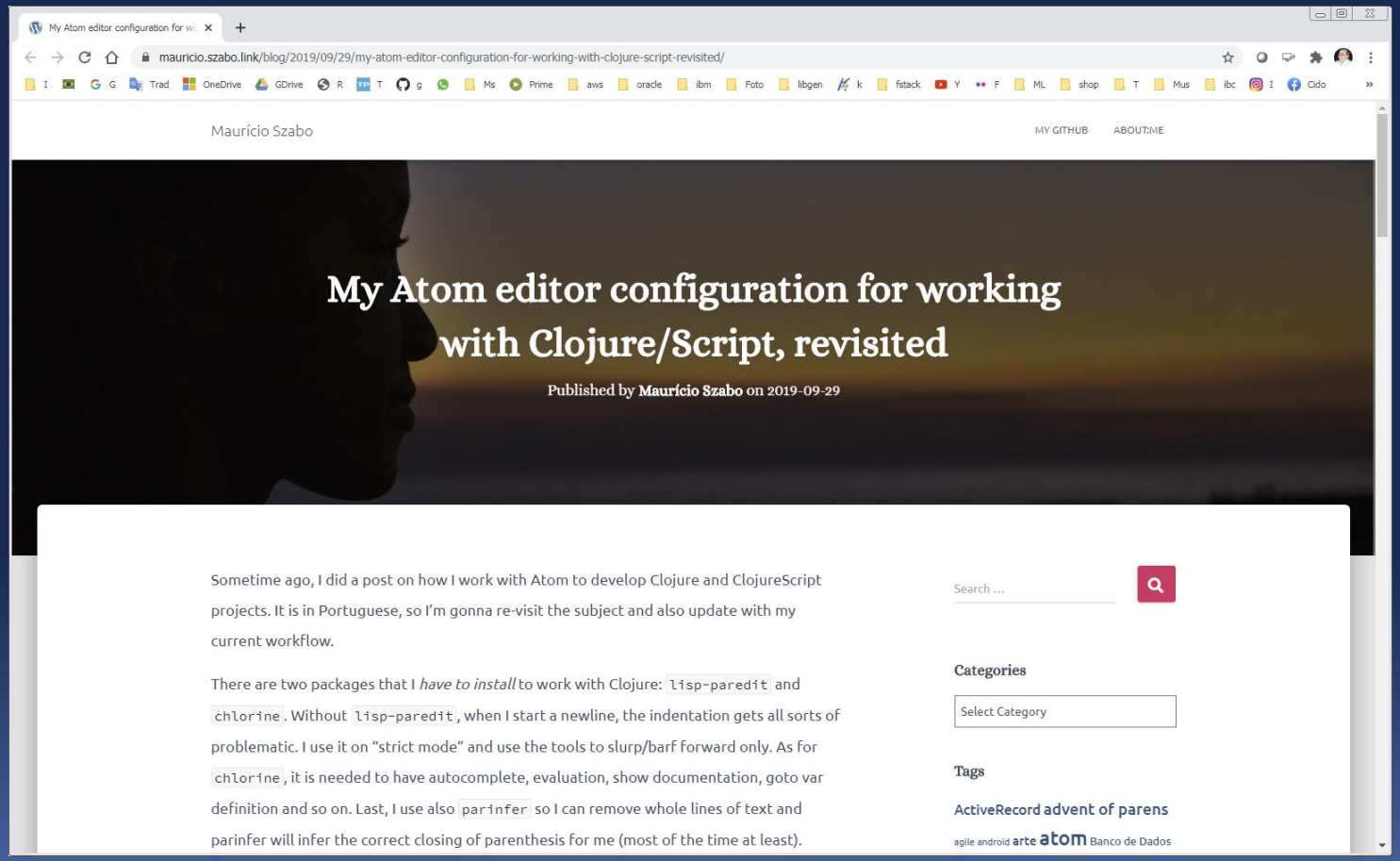
Configuração do Package Chlorine no Atom

- ✓ Instalação do Package Chroline no Atom
- ✓ No menu do Atom: Packages > Settings View > Install Packages / Themes



Configuração do Package Chlorine no Atom

https://mauricio.szabo.link/blog/2019/09/29/my-atom-editor-configuration-for-working-with-clojure-script-revisited/



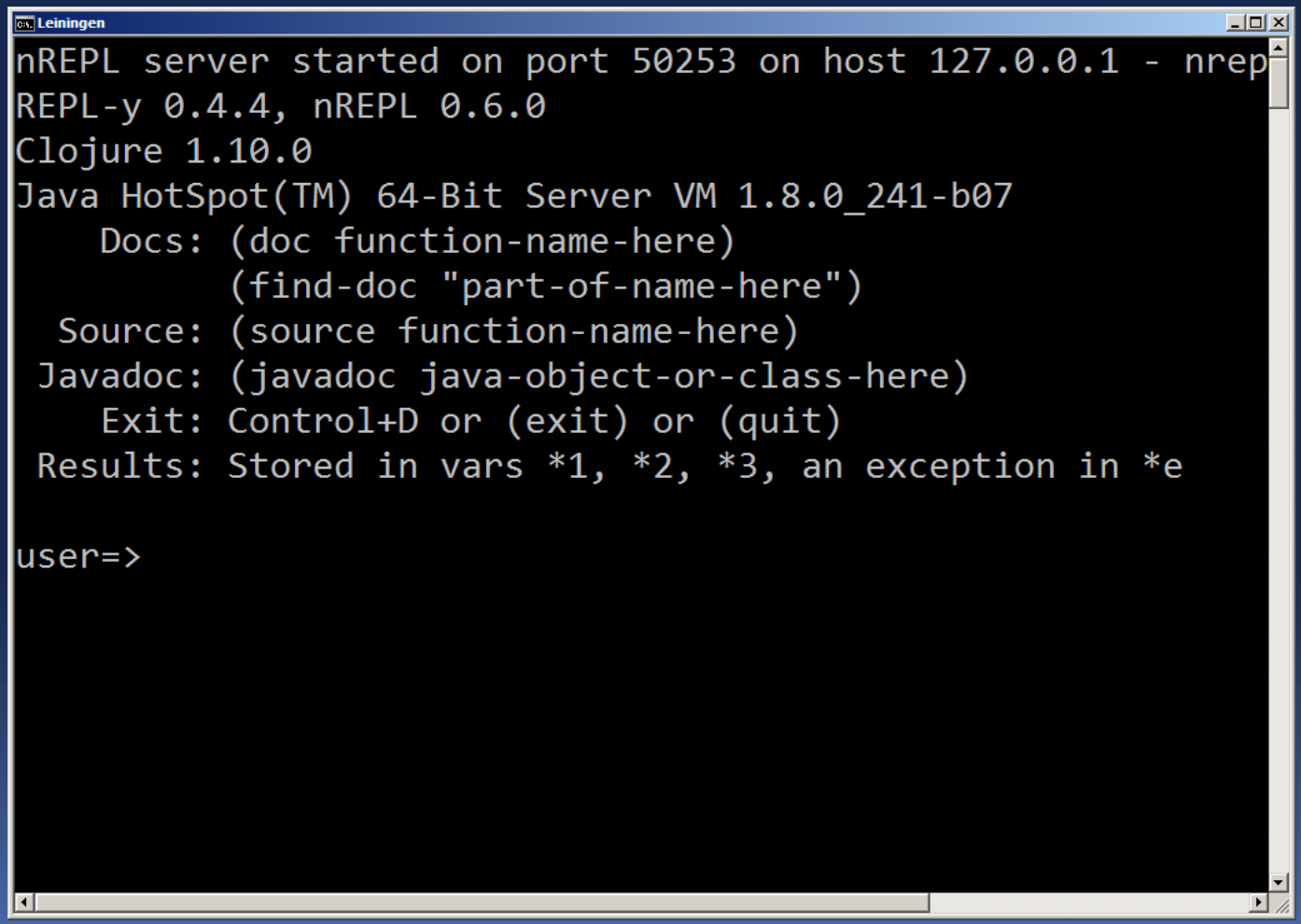
Configuração do Package Chlorine no Atom

- ✓ Sob **Atom**, teclar Crtl + Shift + p e procurar "Open your keymap";
- ✓ Após abertura do arquivo para customizar **keys**, incluir:

```
'atom-text-editor':
  'ctrl-k':                'chlorine:clear-console'
  'ctrl-shitf-l':          'chlorine:load-file'
  'ctrl-shift-enter':      'chlorine:evaluate-block'
  'ctrl-enter':            'chlorine:evaluate-top-block'
  'ctrl-c':                'chlorine:break-evaluation'
  'ctrl-d':                'chlorine:doc-for-var'
```

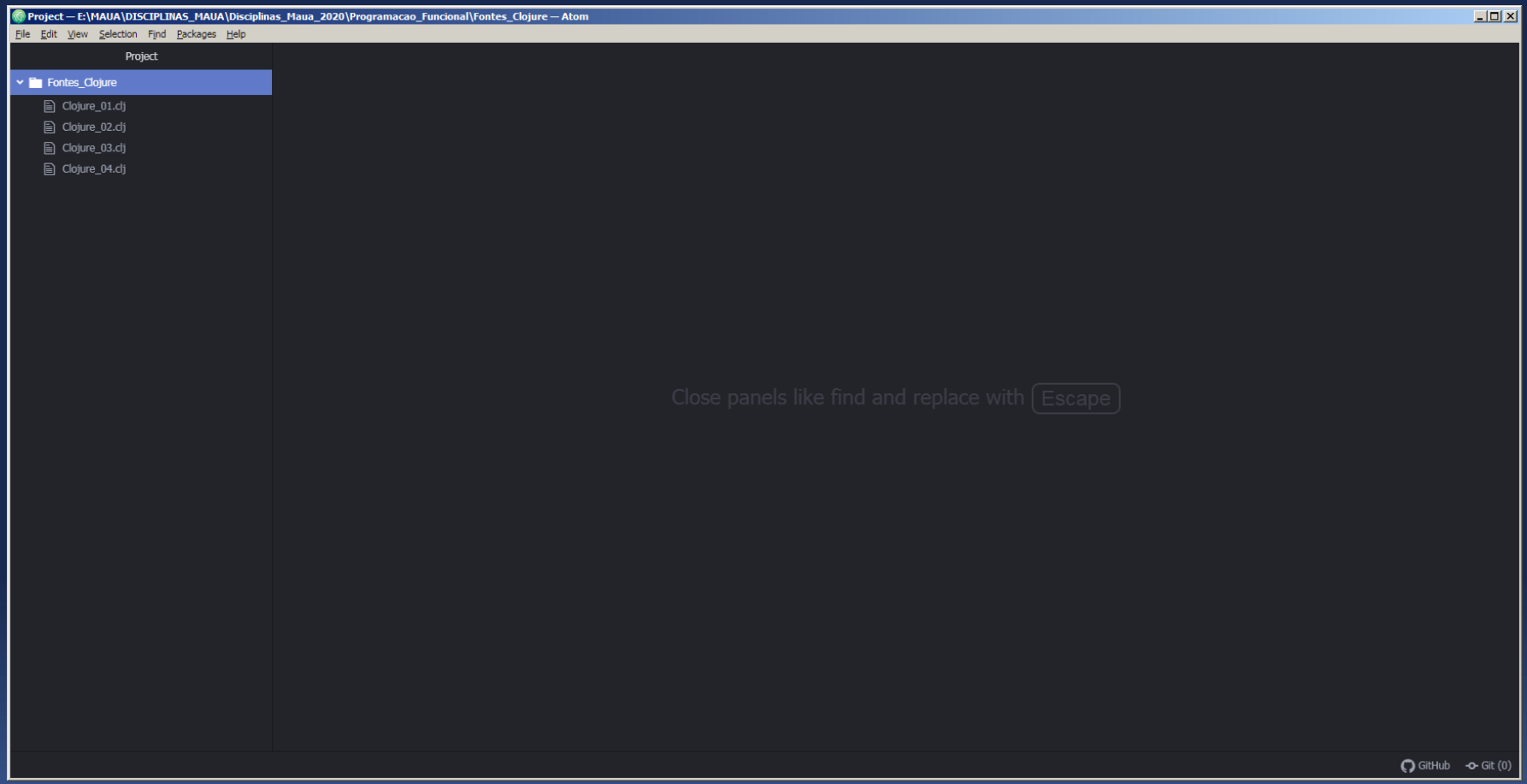
Ativar nREPL - Leiningen

- ✓ Anotar a porta de comunicação, por exemplo: **50253**



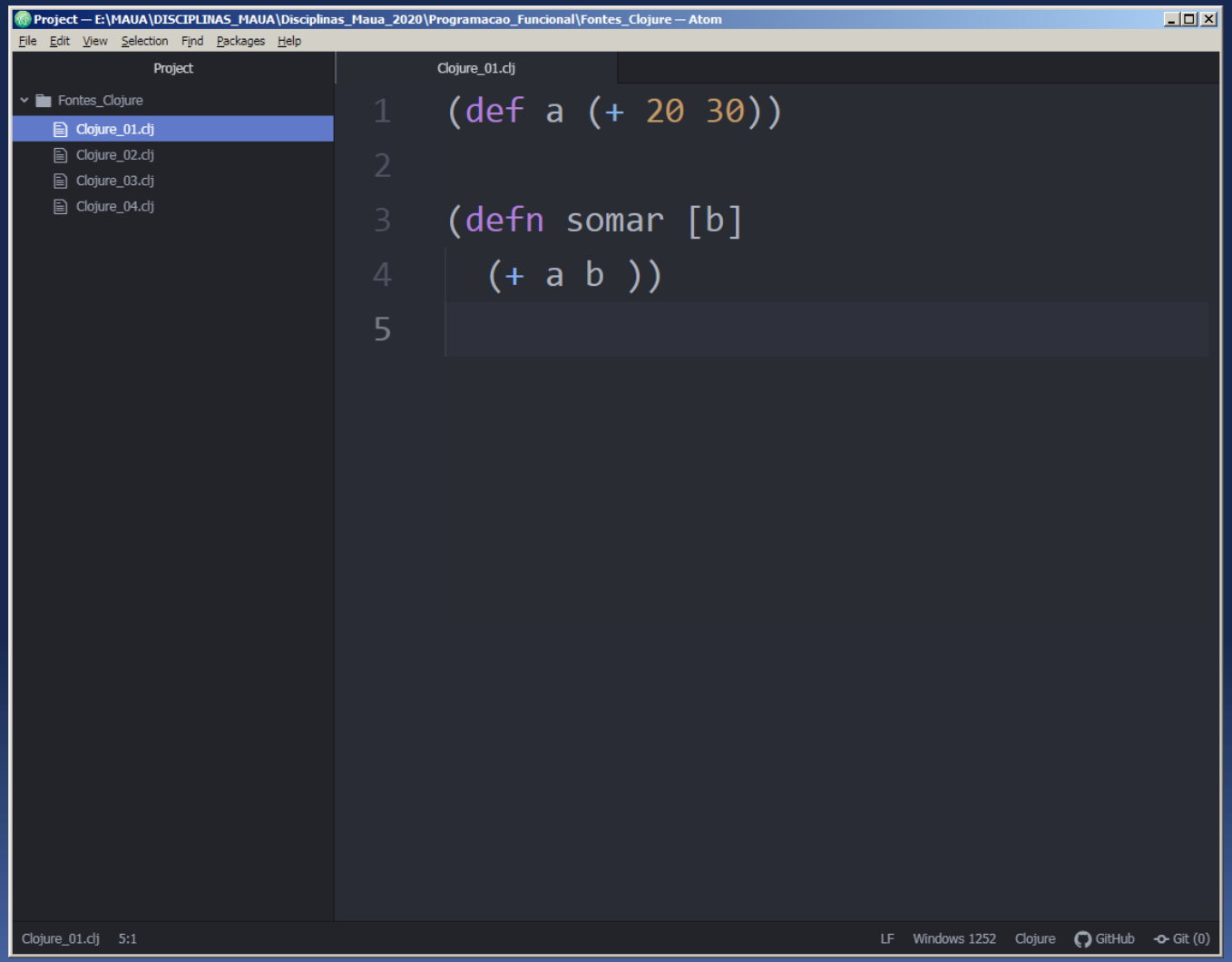
Ativar Atom

- ✓ File > Open Folder



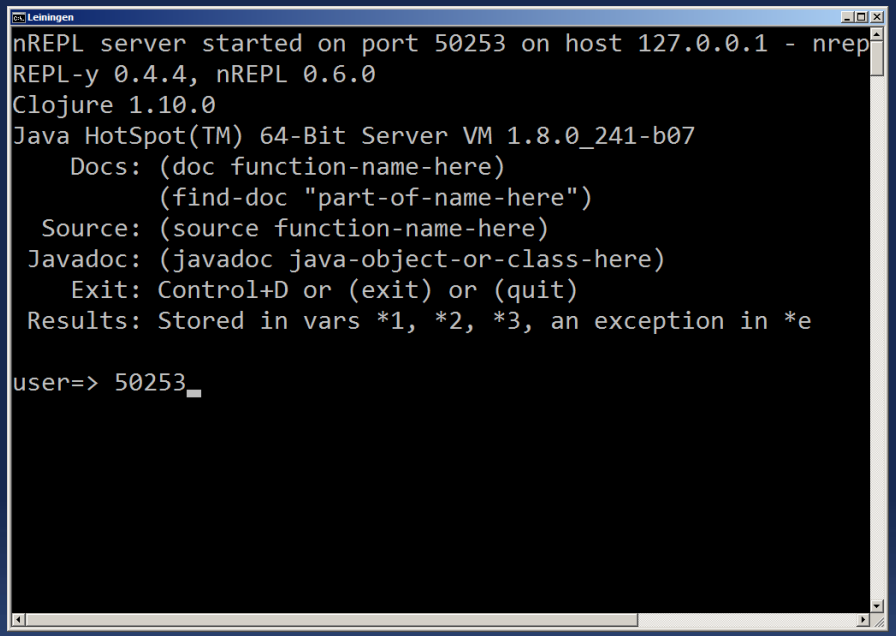
Ativar Atom

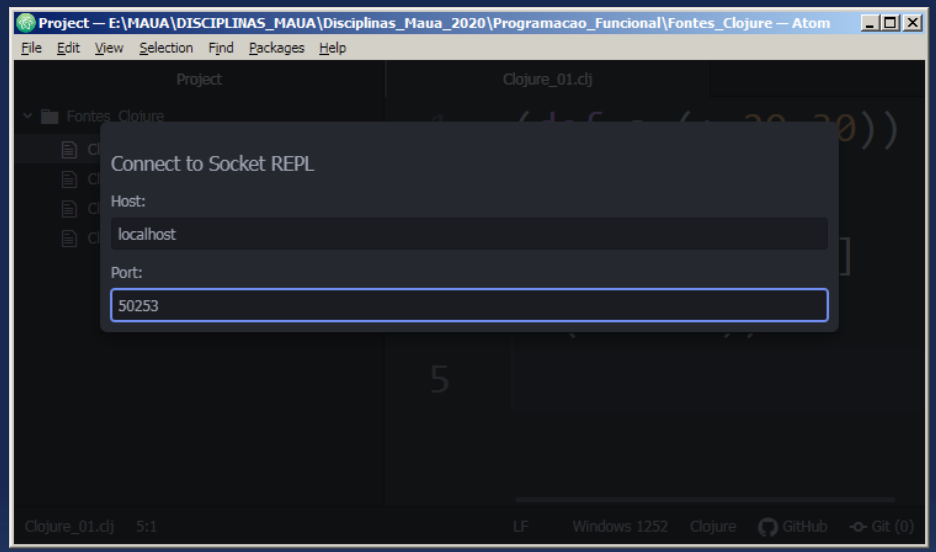
- ✓ Escrevendo código Clojure no Atom



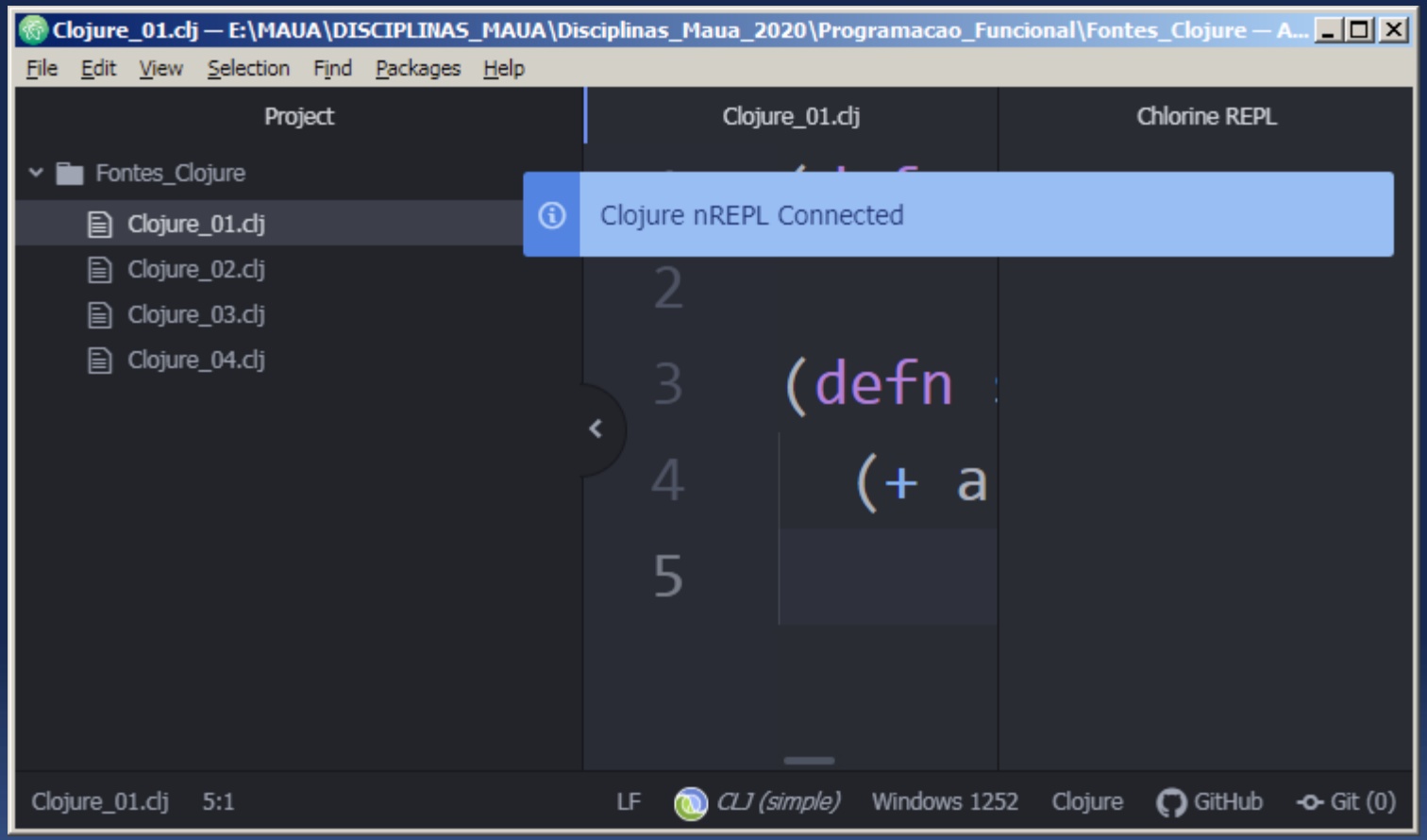
Ativar Atom

- ✓ Ativando **nREPL** (Leiningen)
- ✓ Ctrl + Shift + p > Connect to Socket REPL
  - > Host: **localhost**
  - > Port: aquela fornecida pela nREPL (Leiningen), por exemplo: **50253**



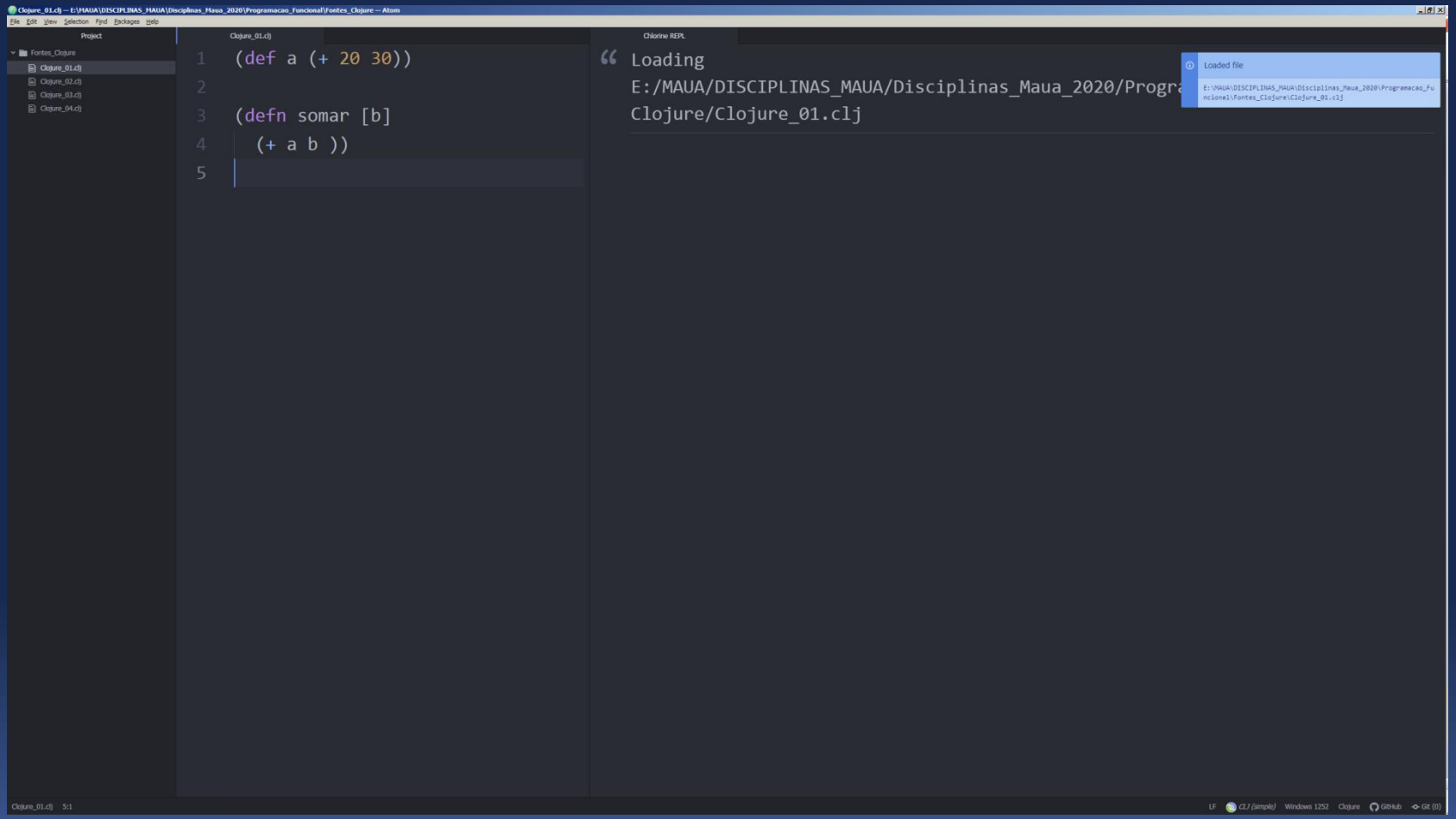


Clojure nREPL Connected



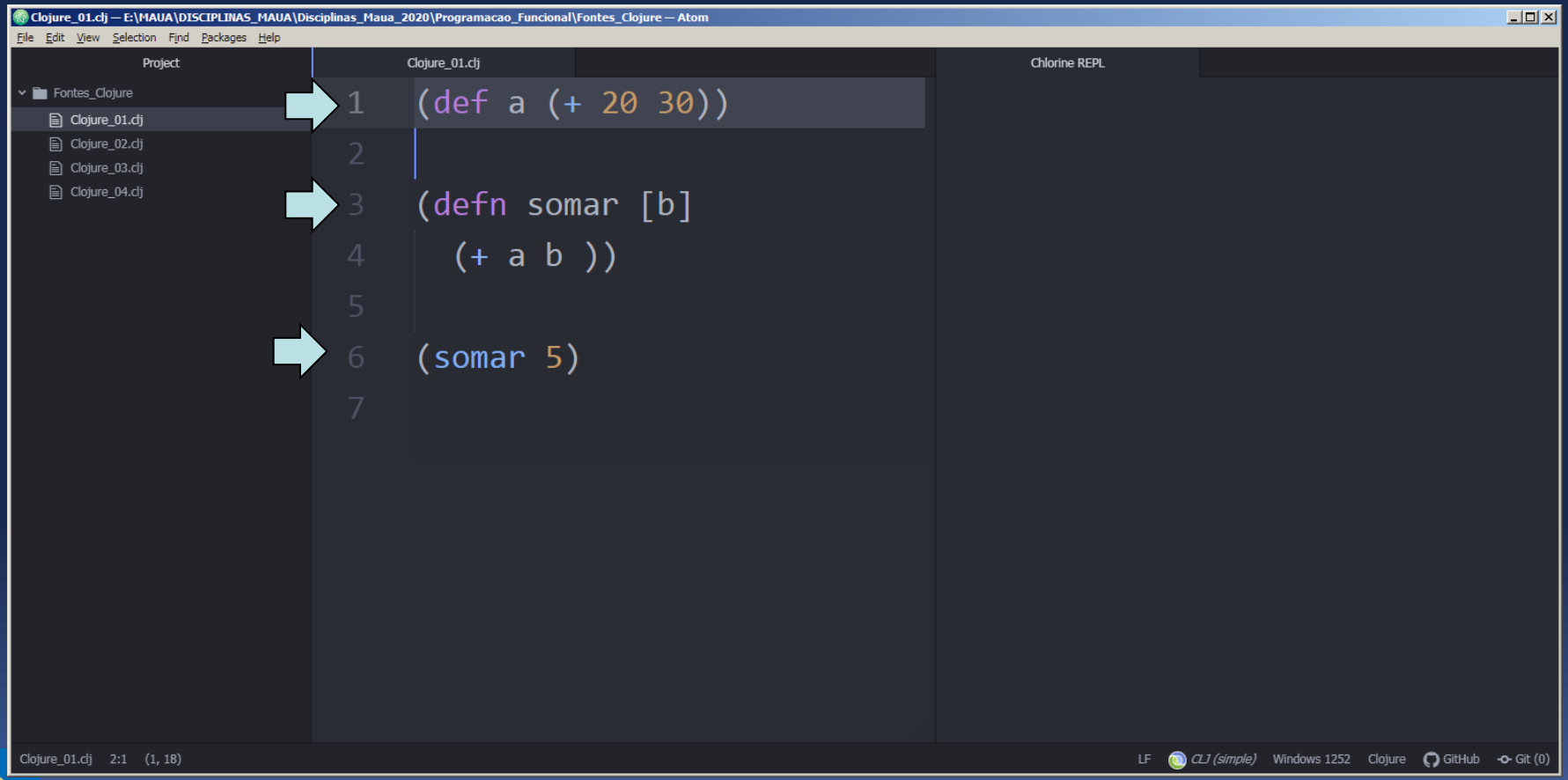
Carregando Código para nREPL

- ✓ Ctrl + Shift +p > Chroline: Load File



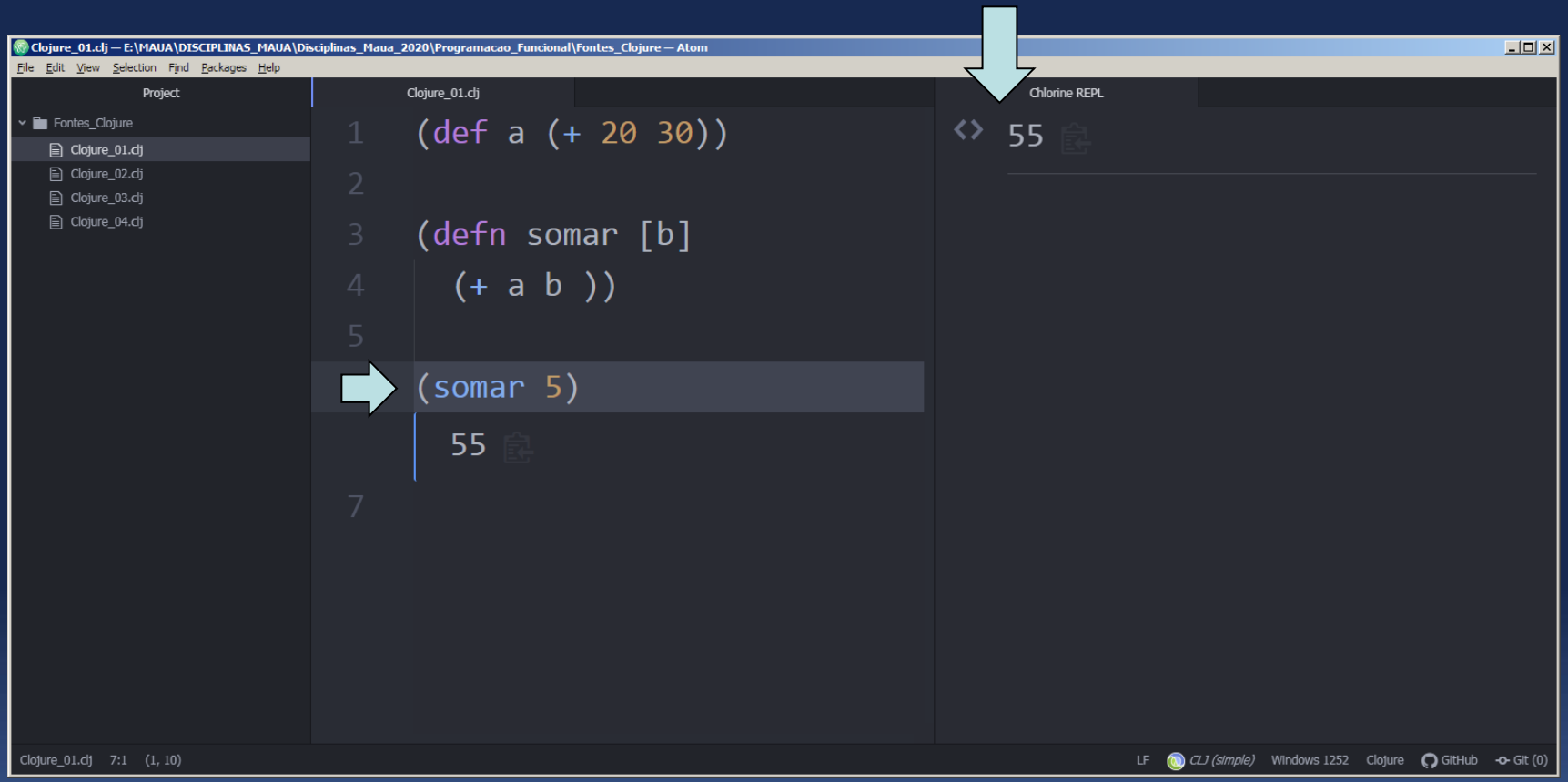
Evaluate "top-blocks"

- ✓ Hot Key: Ctrl + Enter
- ✓ Selecionar o top block desejado e teclar => Ctrl + Enter



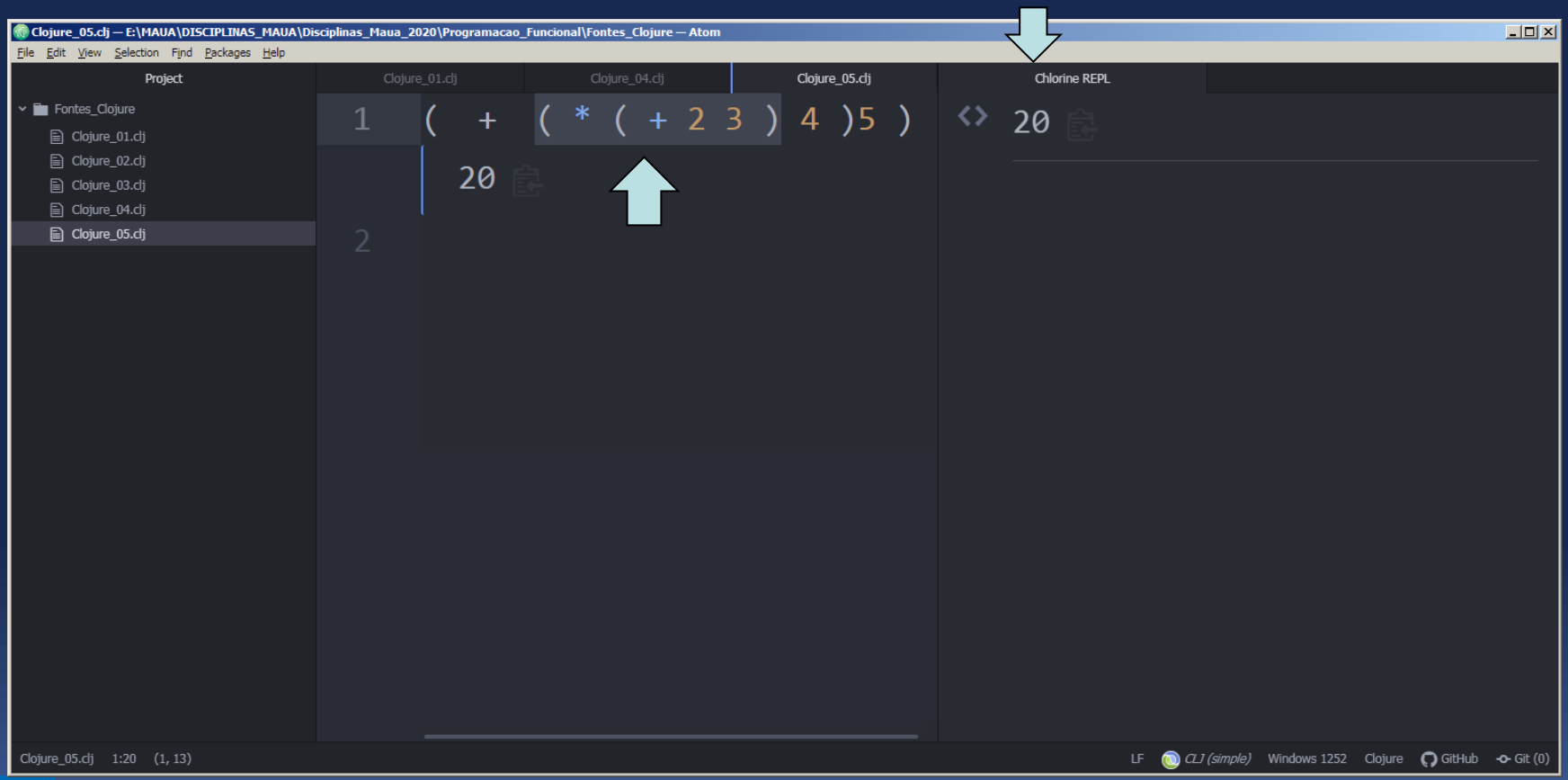
Evaluate "top-blocks"

- ✓ Por exemplo, marcar o top block (somar 5) e teclar => Ctrl + Enter



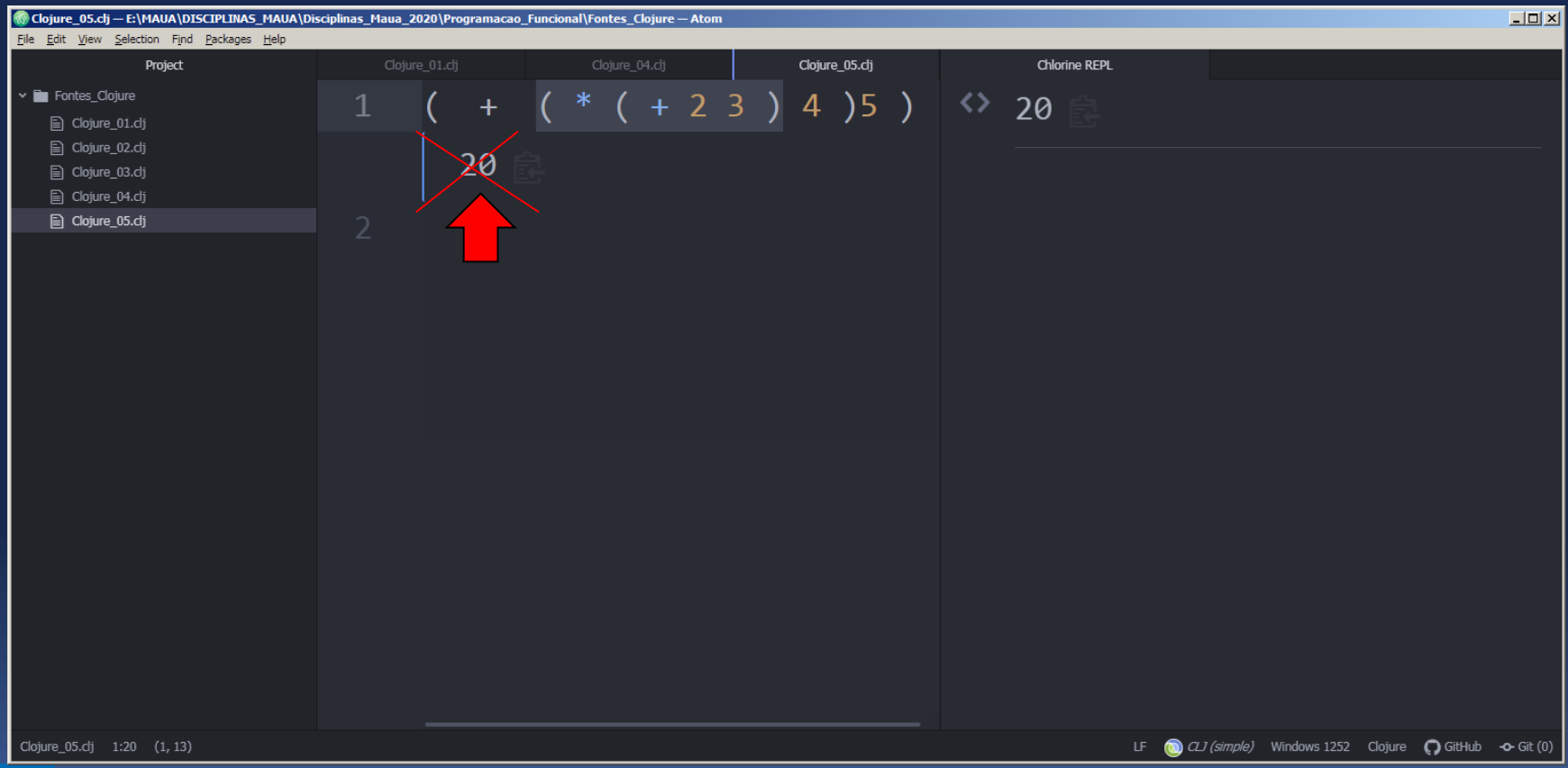
Evaluate "blocks"

- ✓ Hot Key: Ctrl + Shift + Enter
- ✓ Selecionar o block desejado e teclar => Ctrl + Shift +Enter



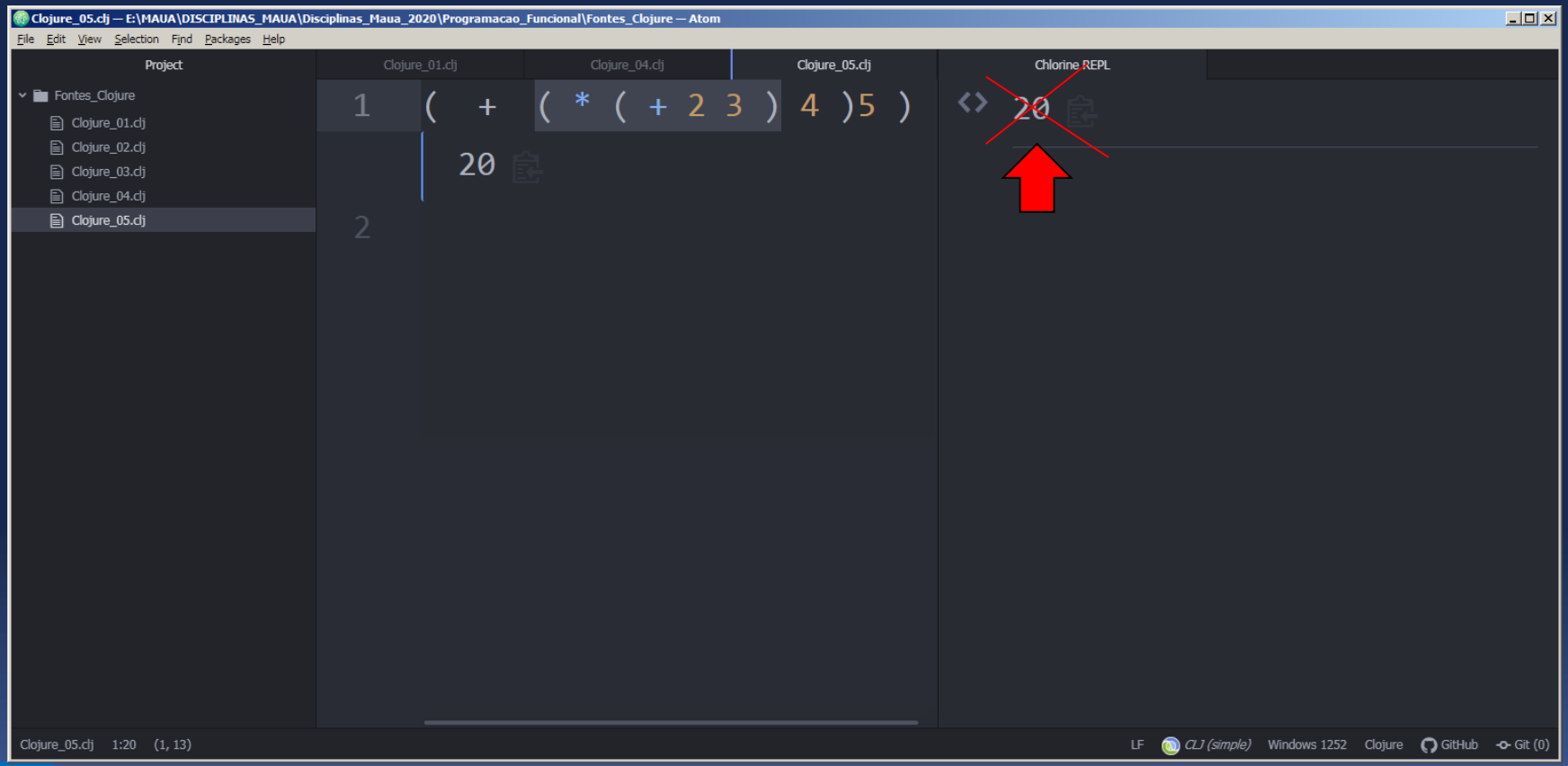
Clear Inline Results

- ✓ Pode também ser feito pela hot key: **Ctrl + Shift + L**



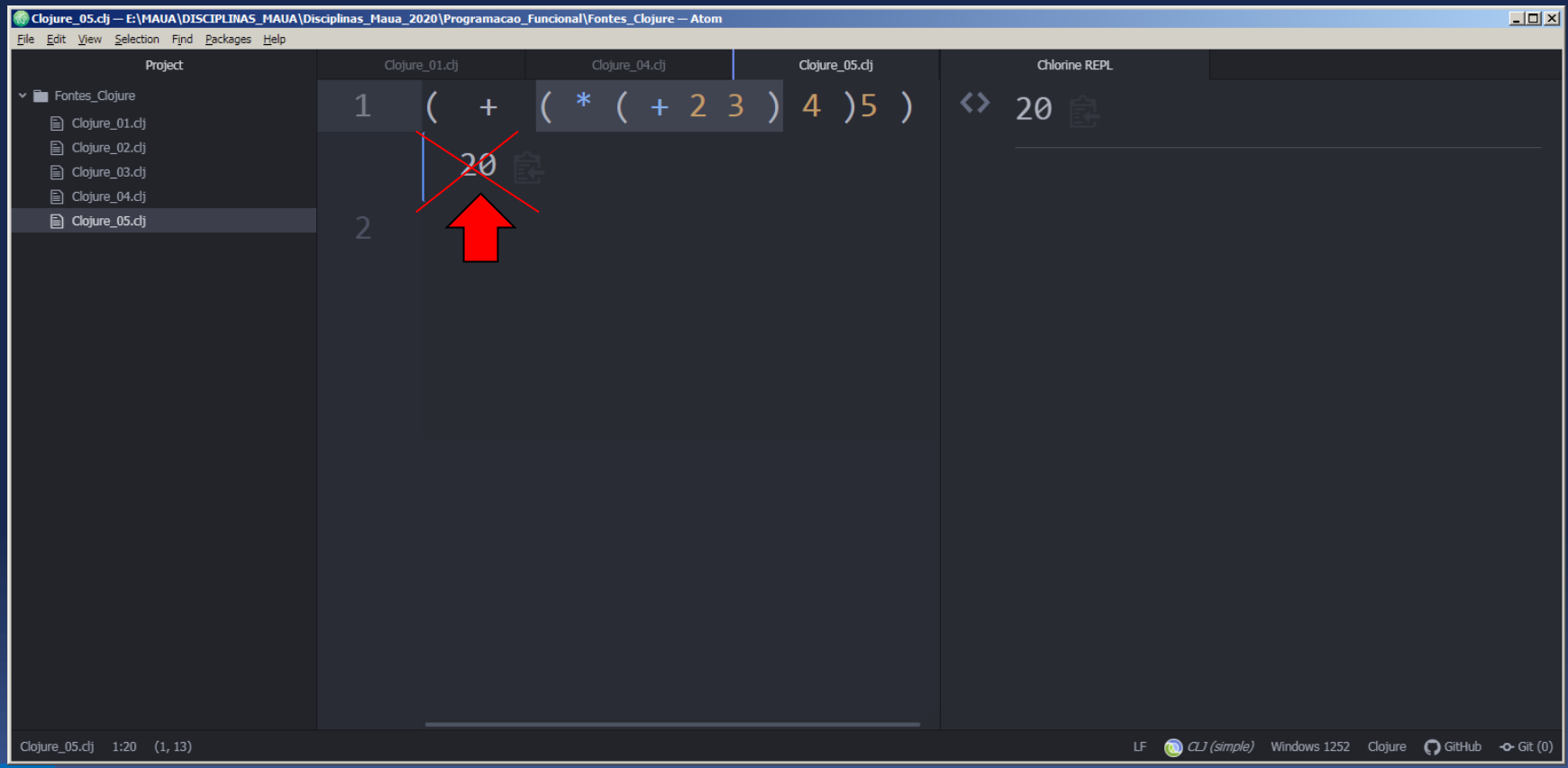
Clear Console

- ✓ Pode ser feito pela hot key: **Ctrl +k**



Clear console

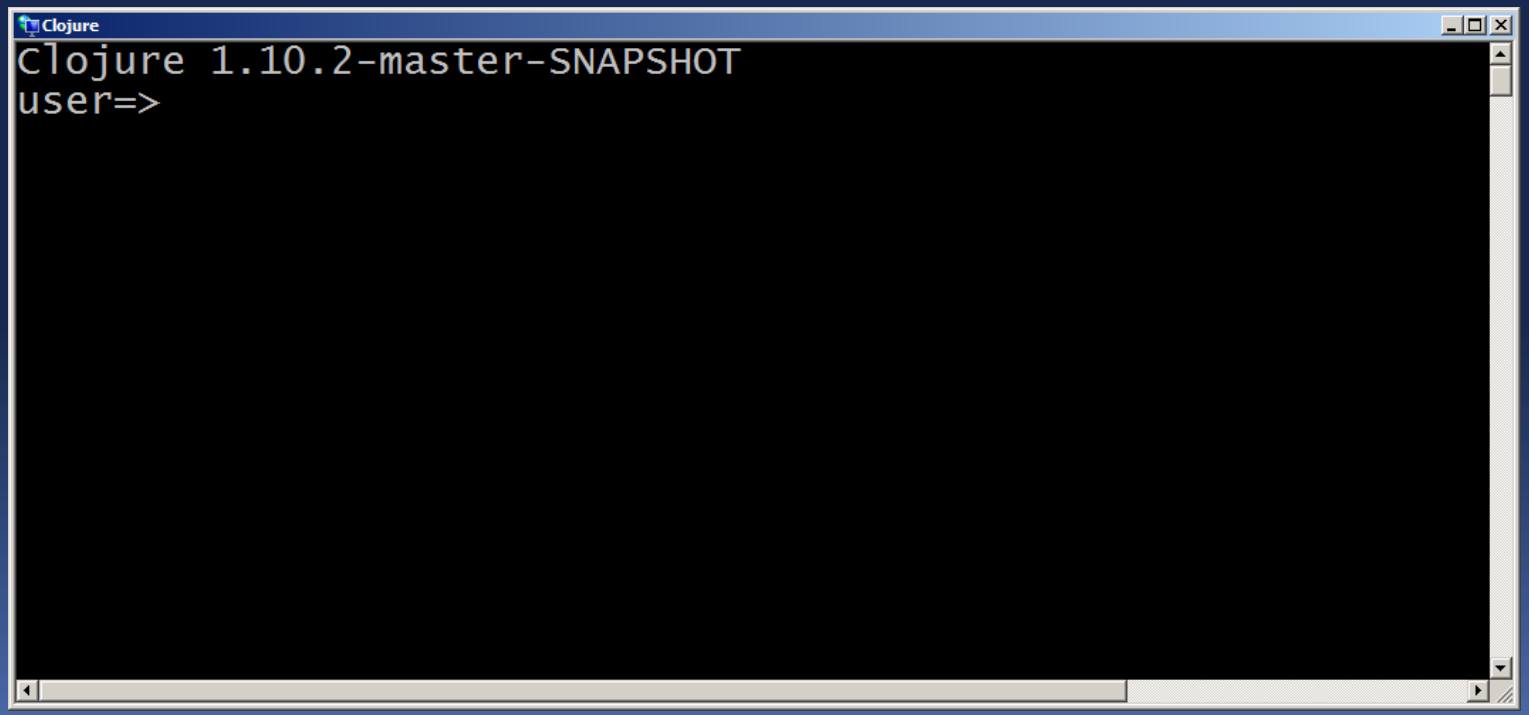
- ✓ Ctrl + Shift + p > Chroline: Clear Inline Results



Desenvolvimento com REPL

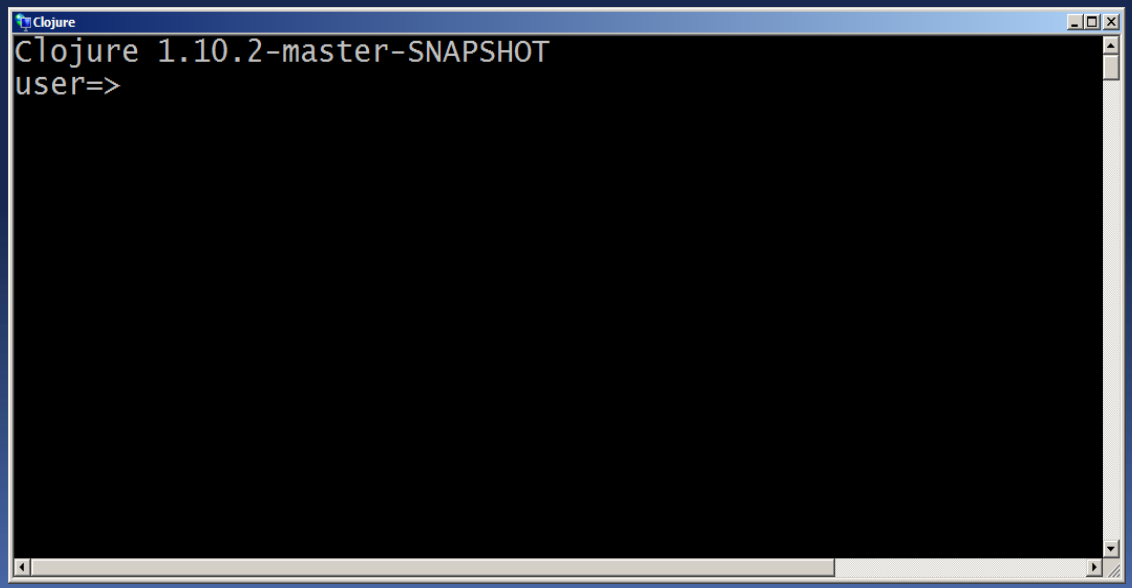
Usando REPL

- ✓ REPL significa **READ EVAL PRINT LOOP** e representa um bom recurso para iniciar o estudo de **Clojure**;
- ✓ É uma **interface** de **comandos** que permite a avaliação direta de código Clojure;
- ✓ No prompt do REPL, a primeira linha representa a **versão** do Clojure, o qual em nosso caso é a **1.10.2**;



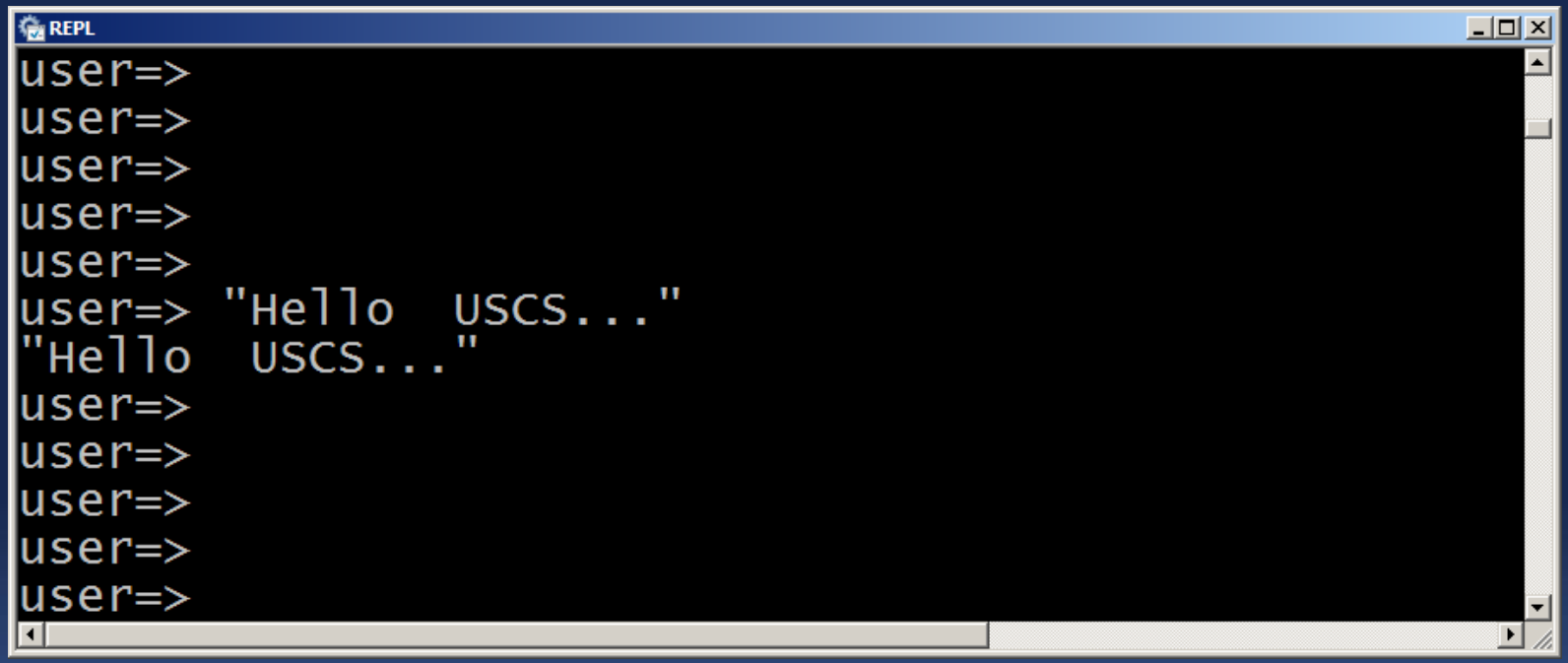
Usando REPL

- ✓ A segunda linha exibe o **namespace** corrente (**user**) e solicita input do usuário;
- ✓ Um **namespace** é um grupo de coisas (tais como funções) que estão agrupadas em um espaço;
- ✓ Aqui, nesse caso, tudo que for criado estará no **namespace user** por default;
- ✓ **REPL** está agora pronto (**ready**).

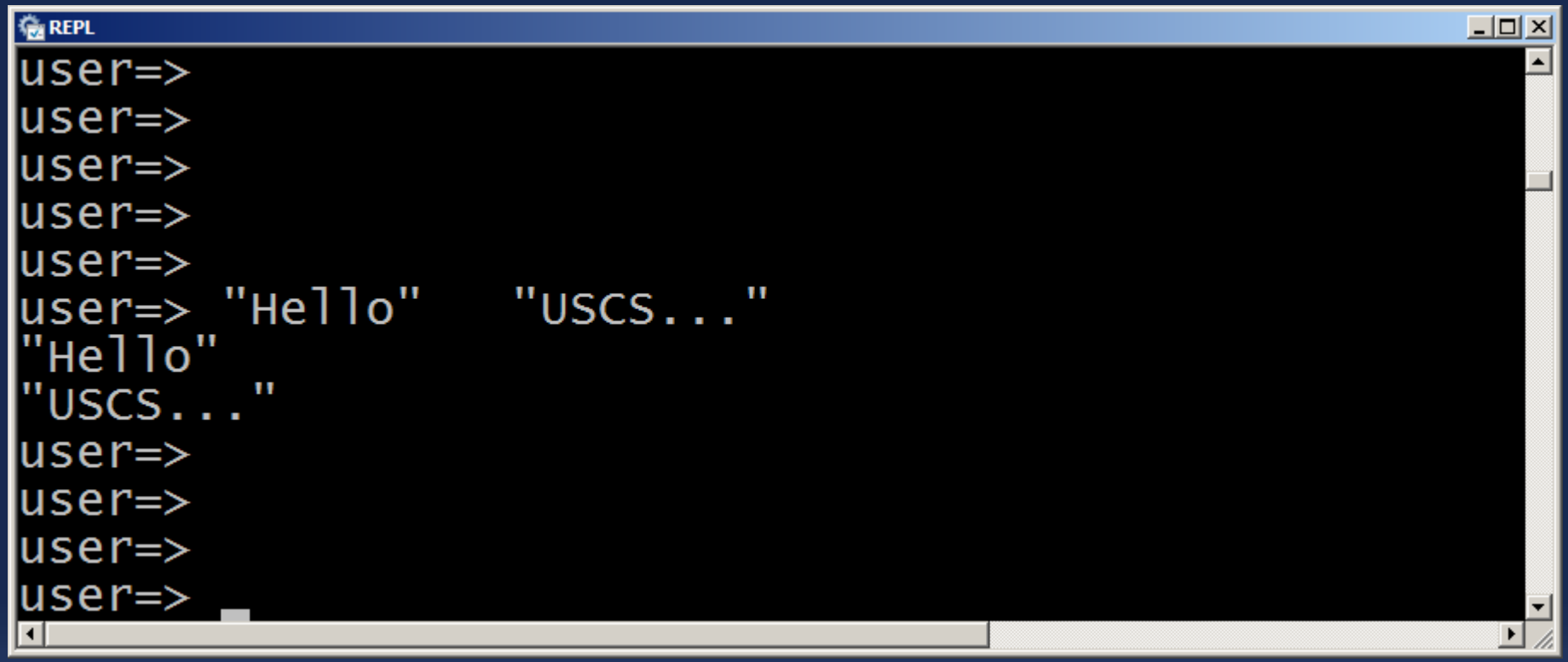


Avaliando expressões

- ✓ Em **Clojure**, **literais string** são criados com aspas duplas, " ";
- ✓ Um **literal** é uma notação para representar valores **fixos** no código fonte.

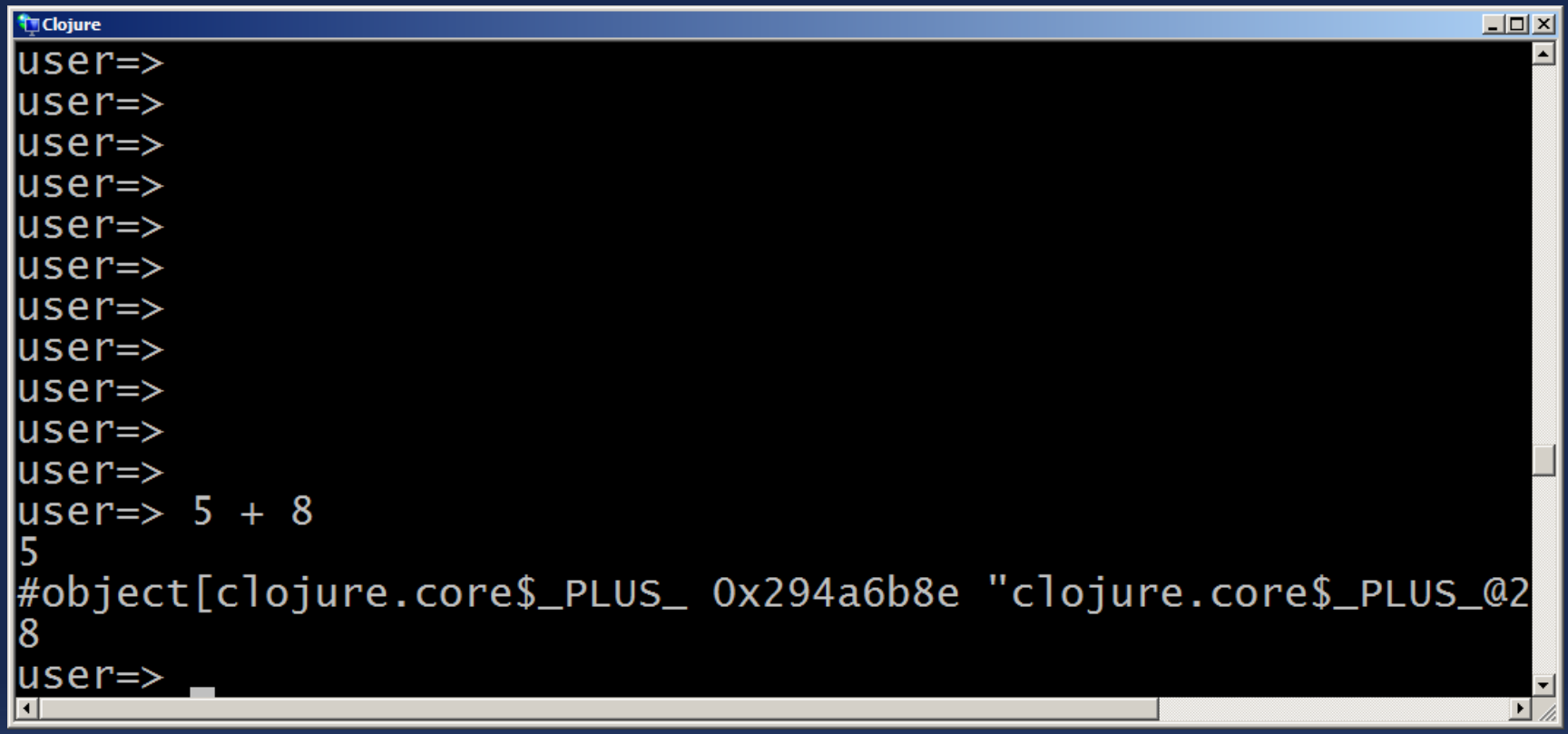


Avaliando múltiplos strings



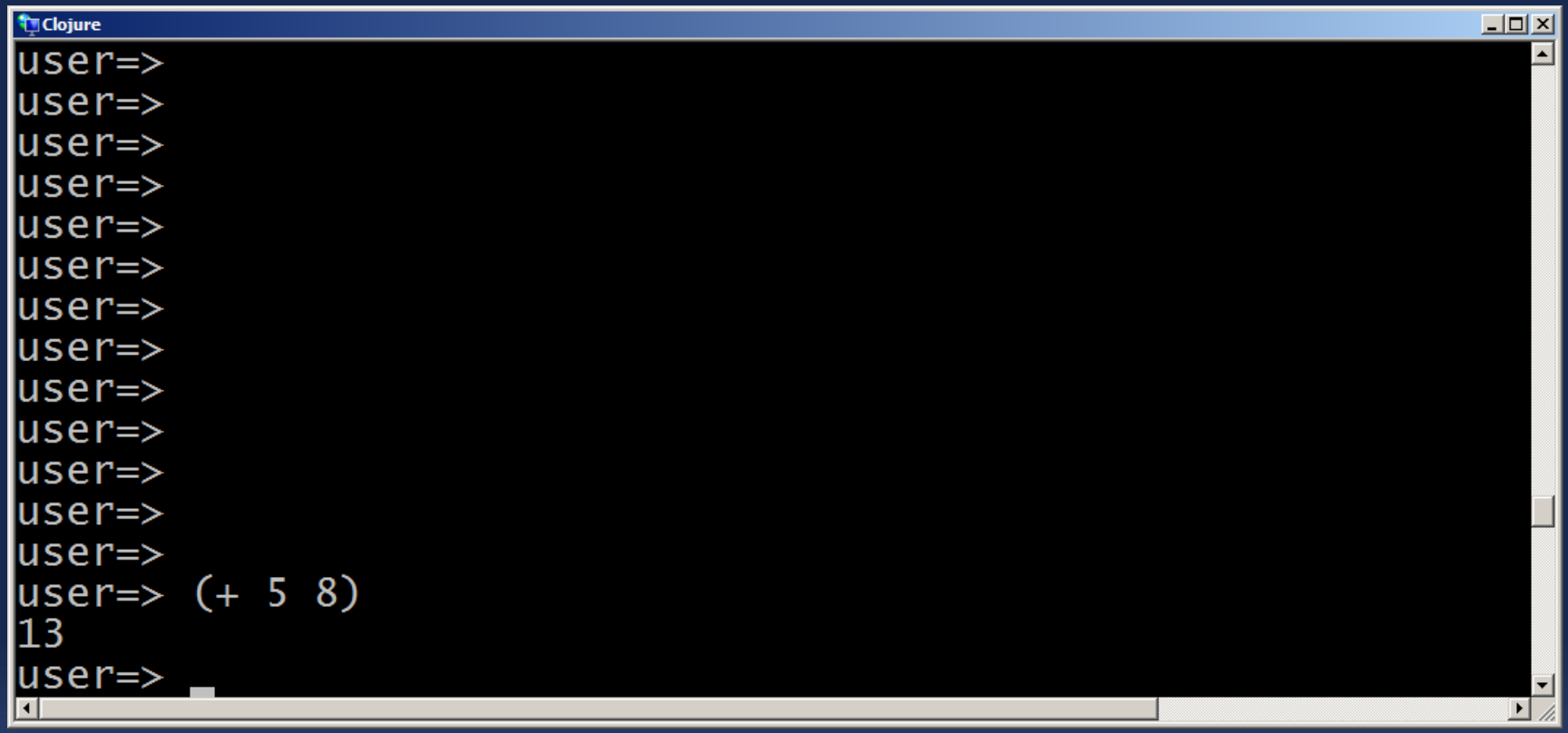
- ✓ Nesse exemplo, **duas** expressões **string** foram avaliadas sequencialmente, e cada qual foi **retornada** em duas linhas separadas;

Avaliando expressões aritméticas

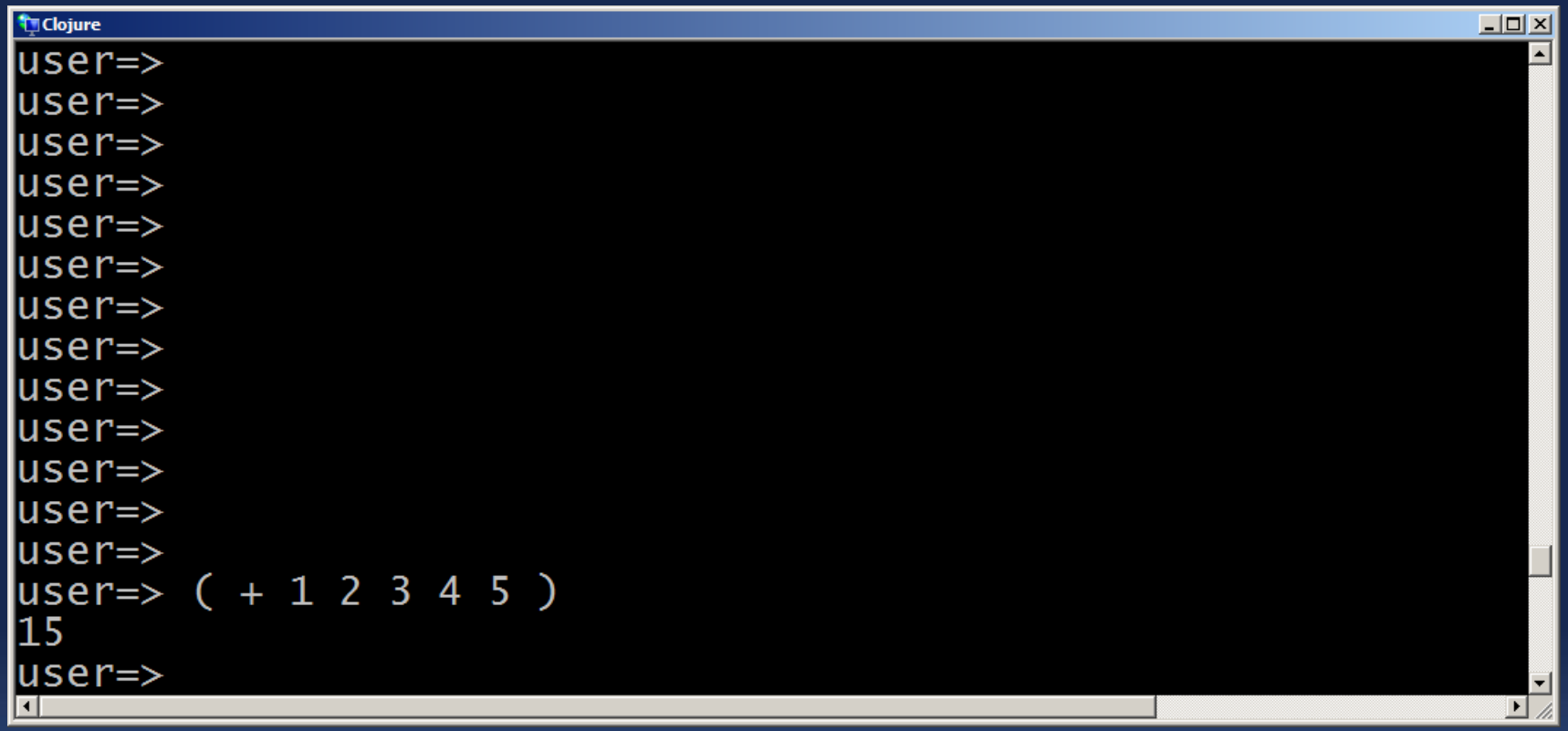


- ✓ Clojure retornou **erro**, pois o primeiro argumento deve ser uma **função**. No caso, o símbolo **+** está ligado (**bounded**) à uma função e deve ser o primeiro elemento da lista a ser avaliada, seguido pelos operandos (**argumentos**).

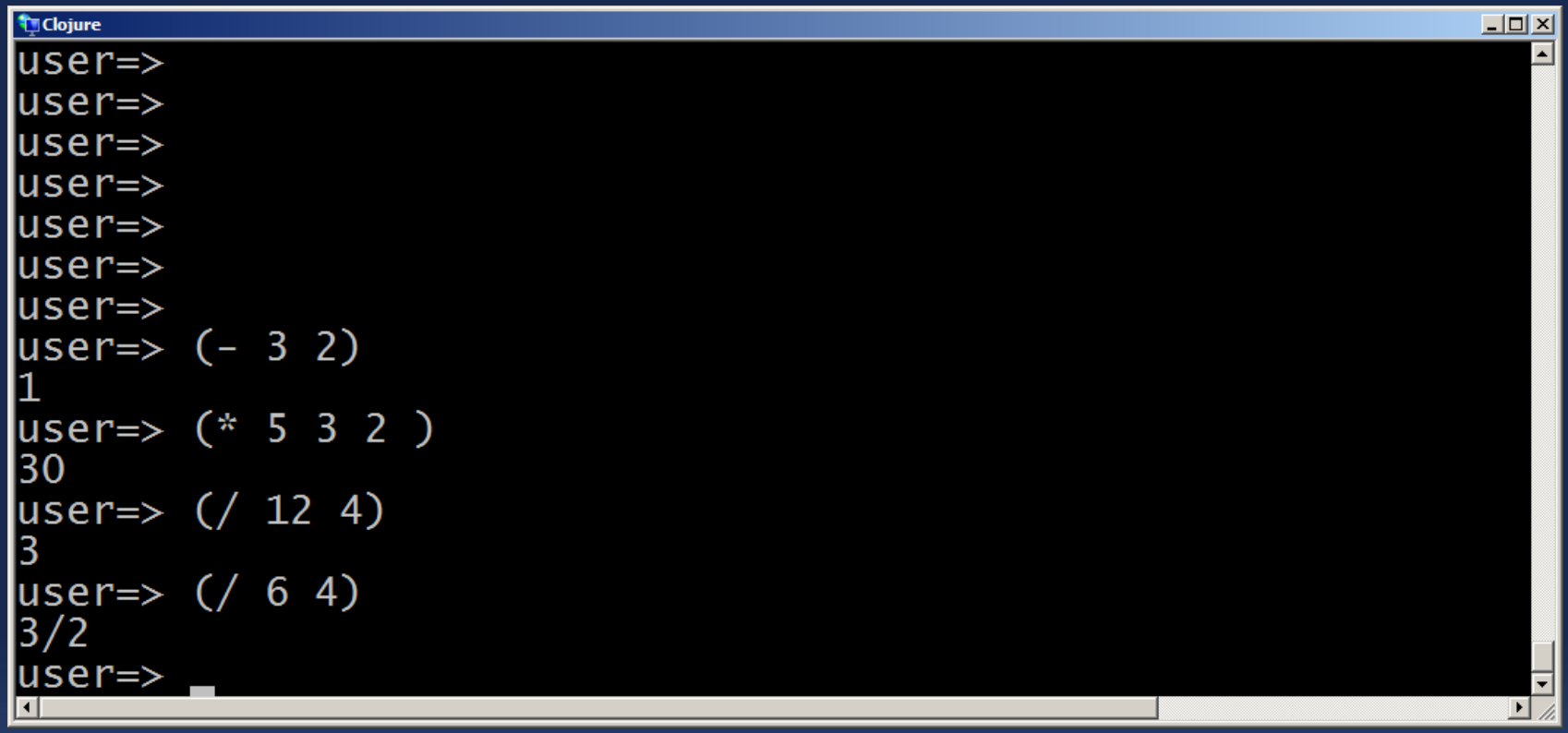
Avaliando expressões aritméticas



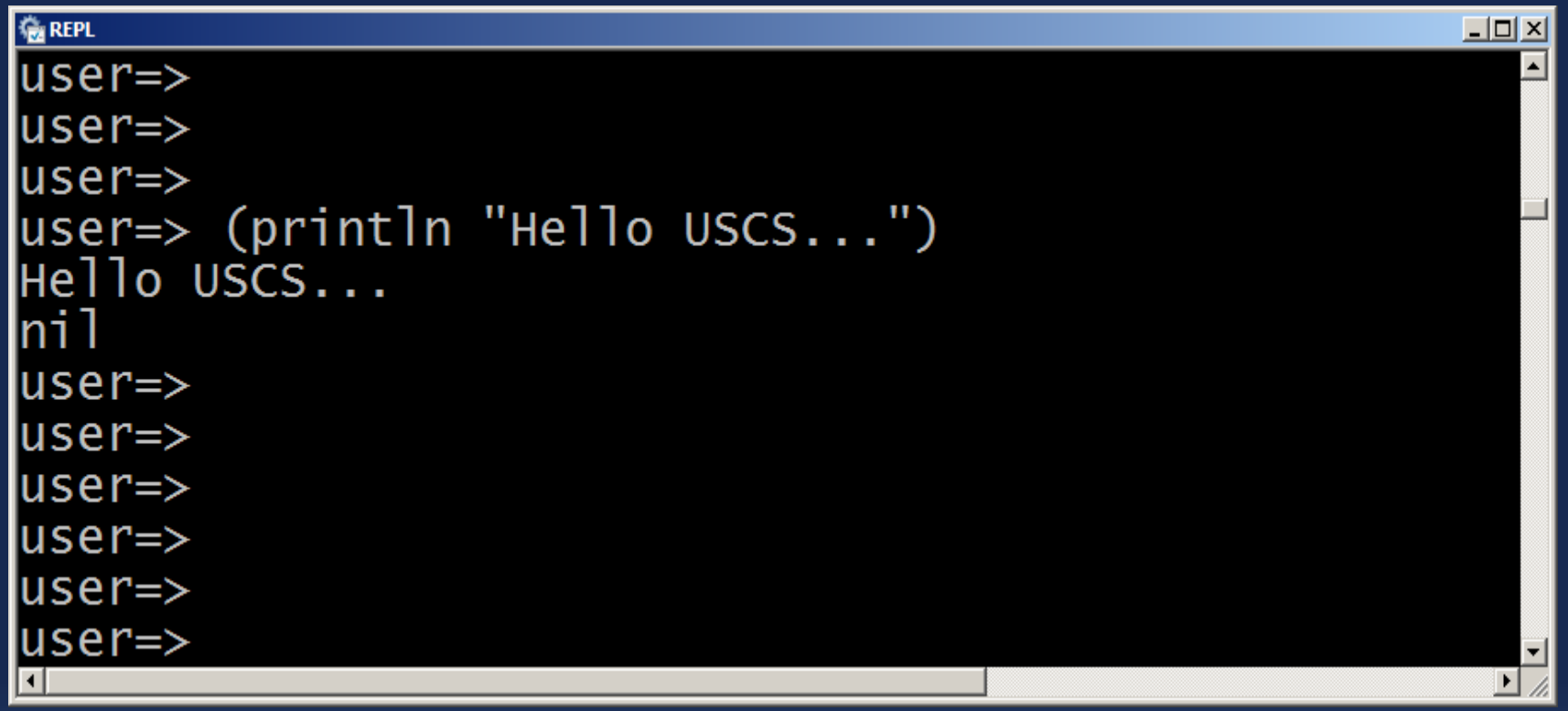
Avaliando expressões aritméticas



Avaliando expressões aritméticas



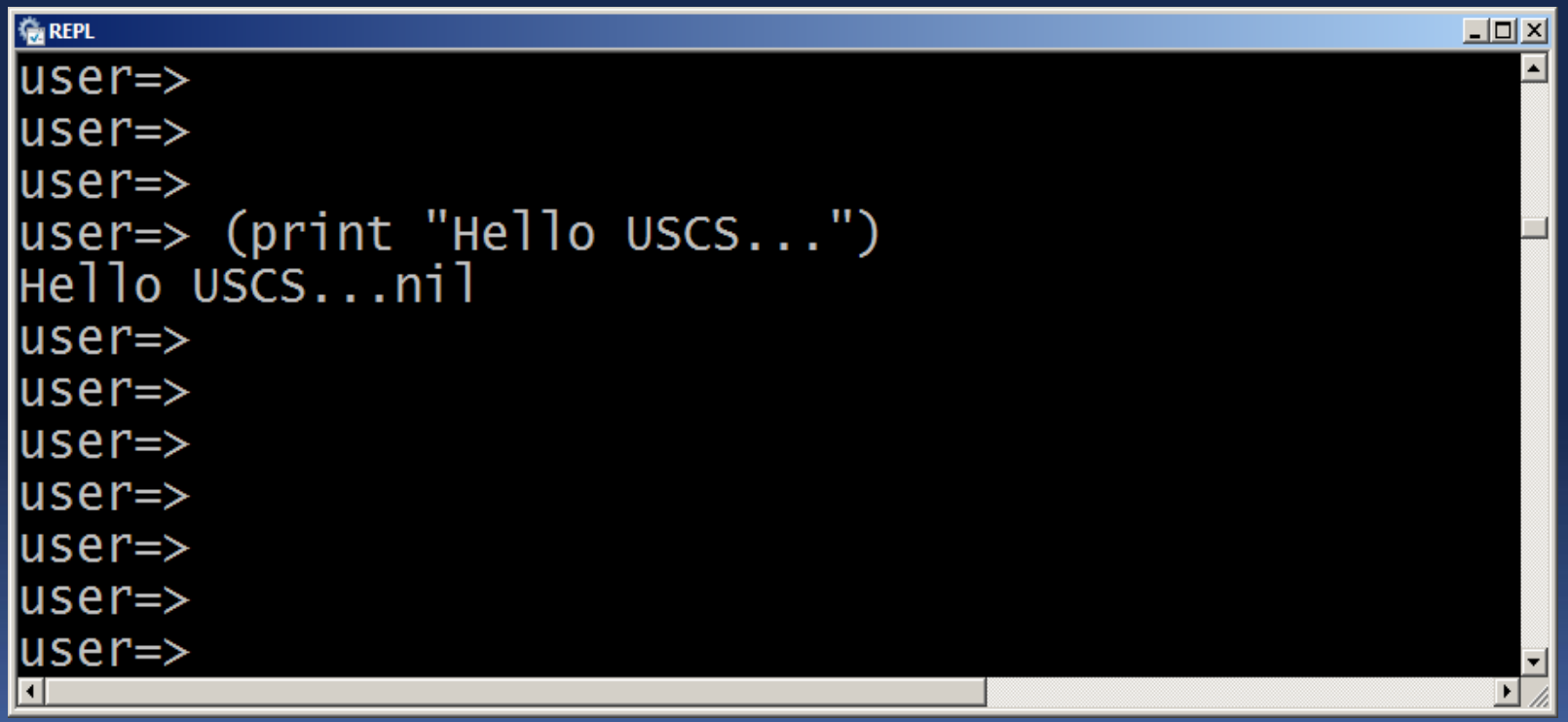
Função println



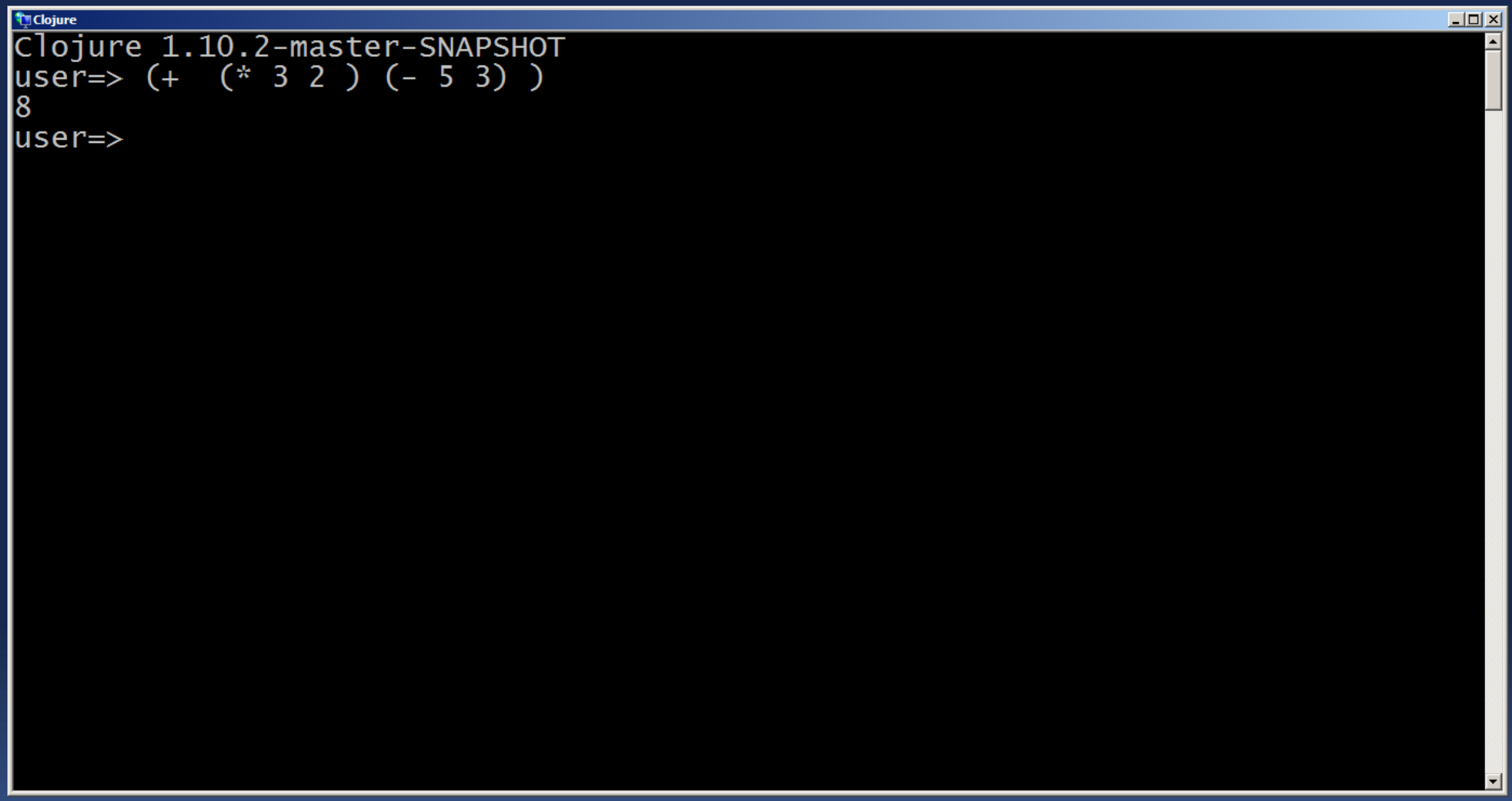
- ✓ O texto impresso pela função **println** foi um **SIDE EFFECT**;
- ✓ A função na verdade **retornou** **nil**.

nil

- ✓ Equivale em **Clojure** a um valor "**null**" ou "**nada**";
- ✓ Ou seja, trata-se da **ausência** de **significado**;
- ✓ As funções **print** e **println** são usadas para imprimir objetos para a saída padrão e retornam **nil** uma vez que a impressão foi efetivada;



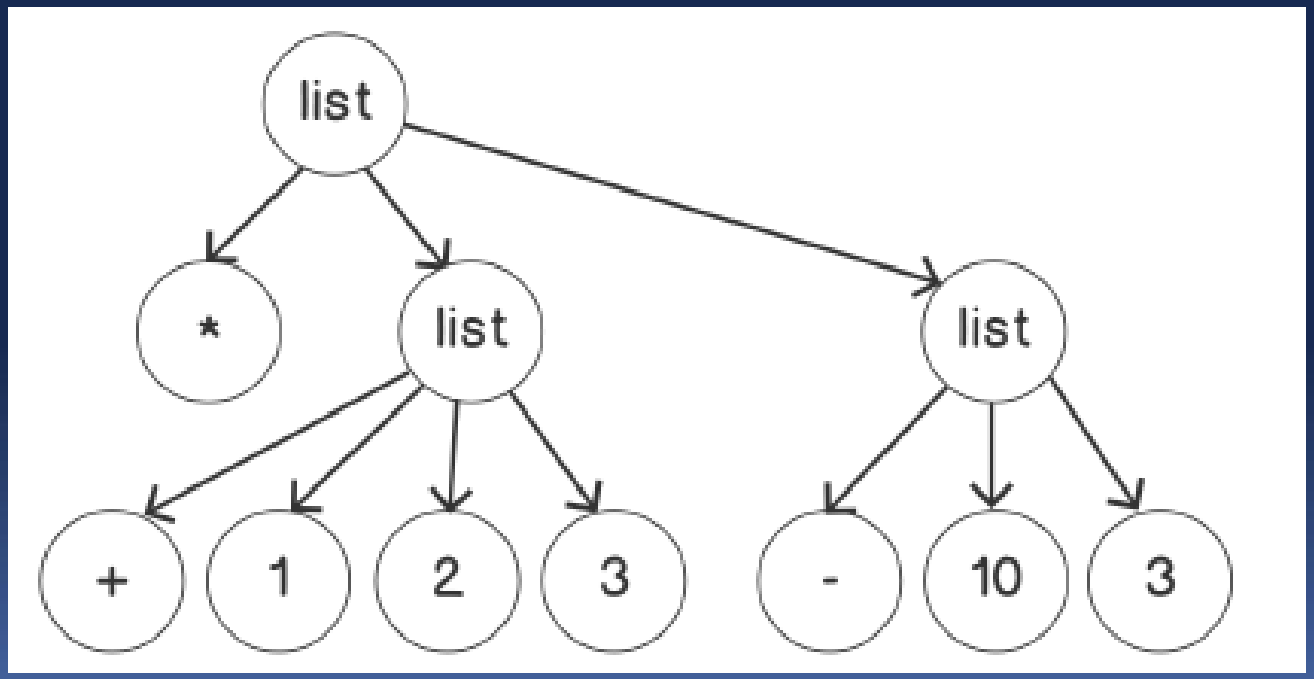
Funções nested



Expressões Simbólicas (s-expression)

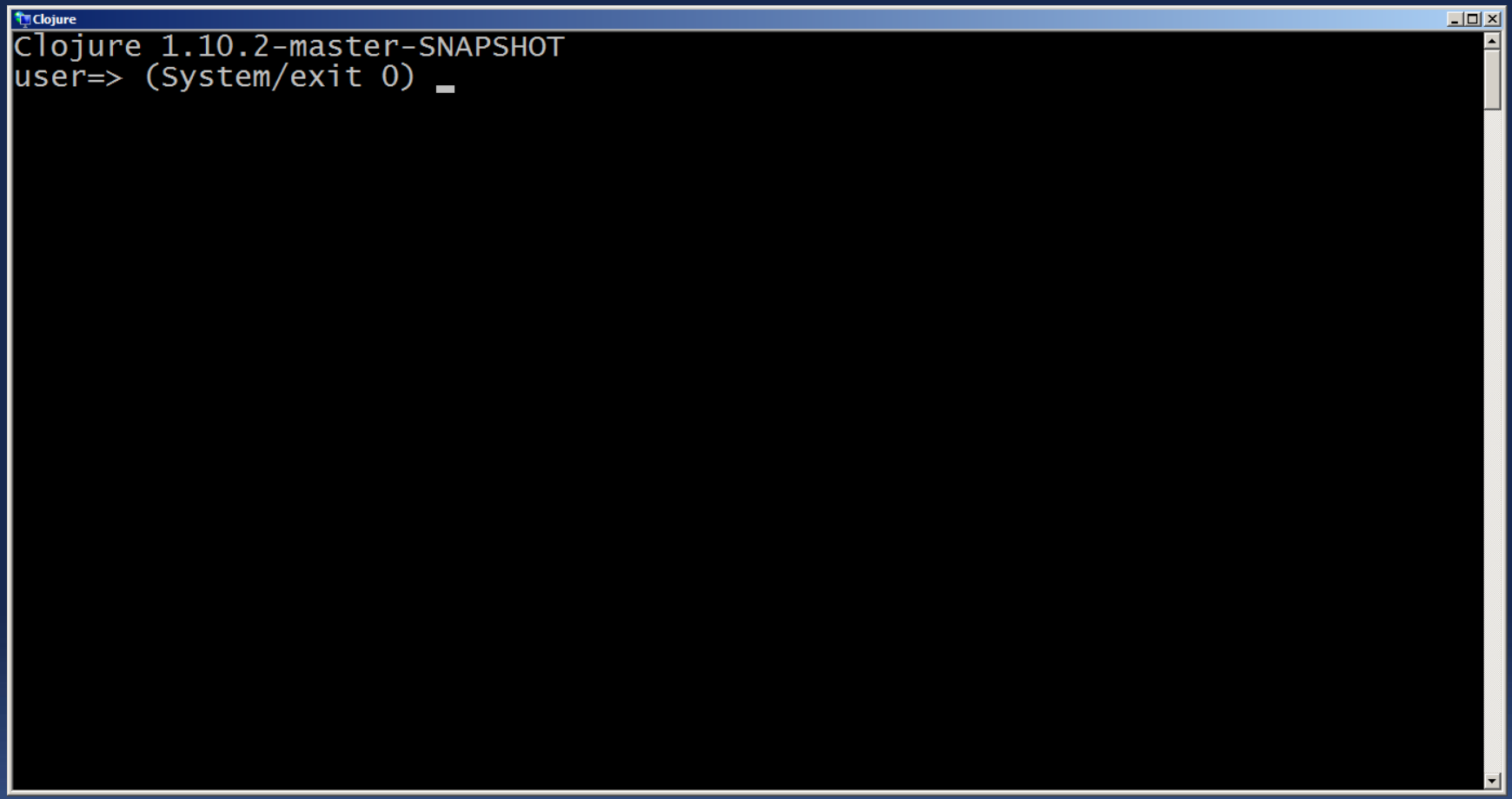
- ✓ Podem ser visualizadas em uma árvore

```
(* (+ 1 2 3) (- 10 3))
```



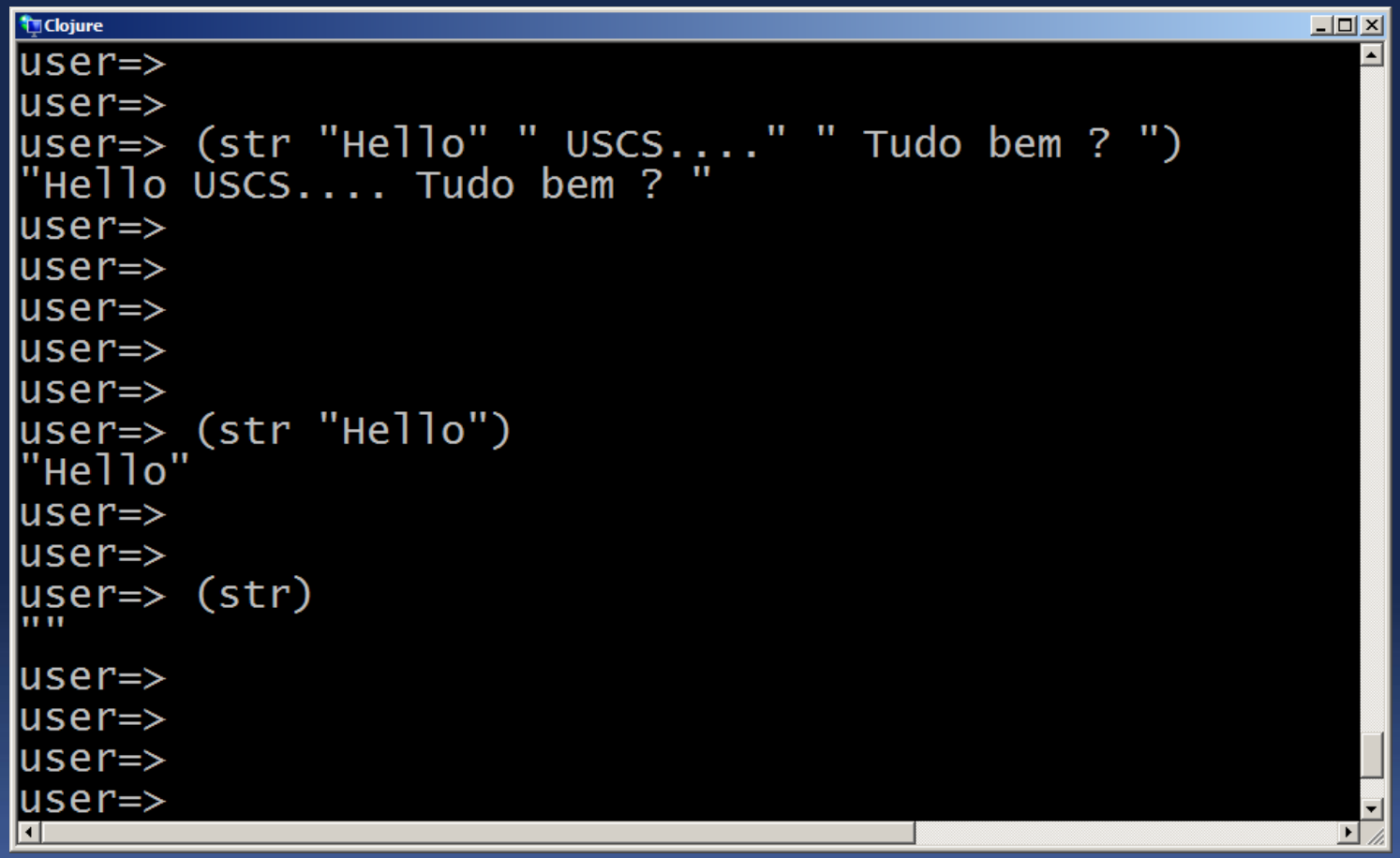
Exit REPL

(System/exit 0)



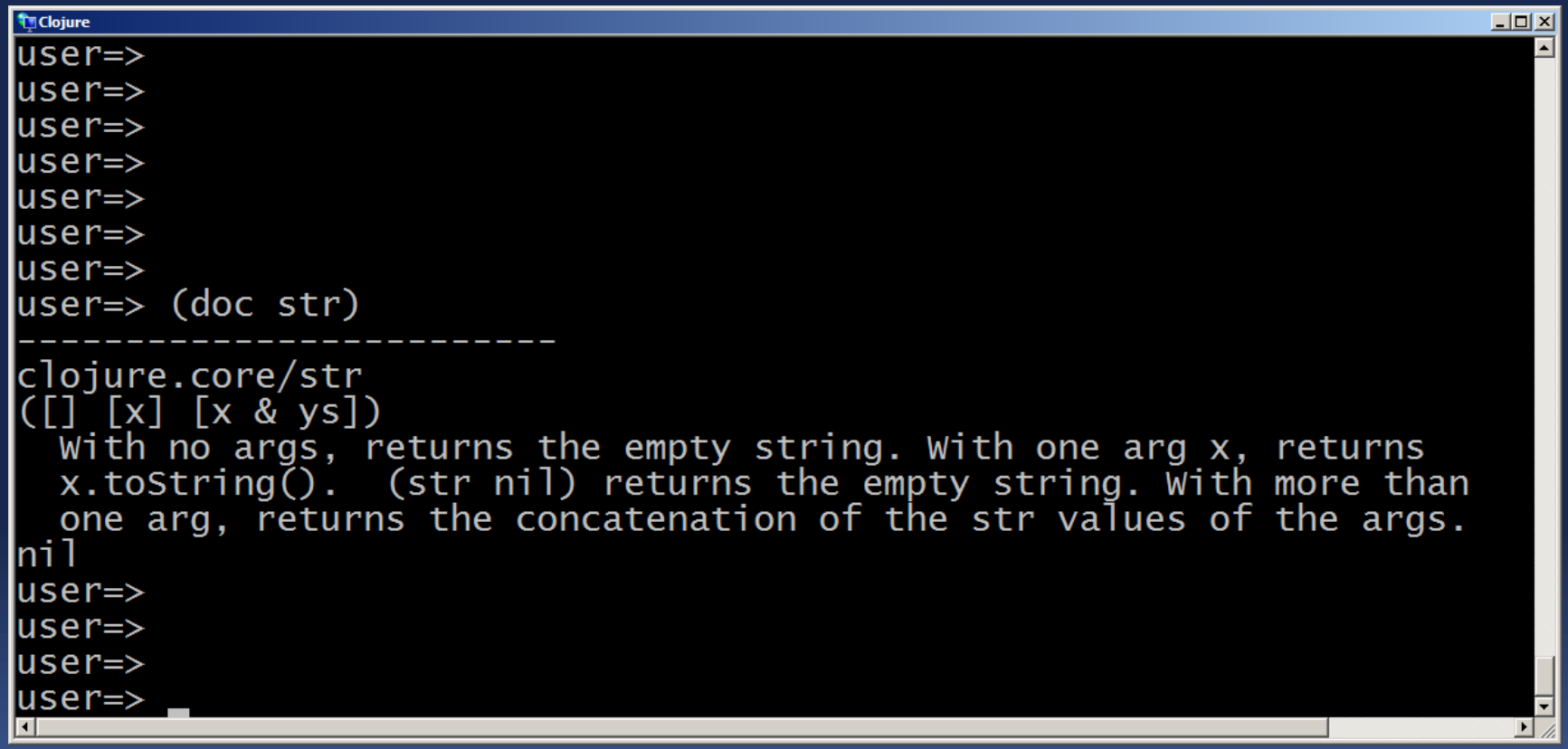
Concatenando Strings - str

- ✓ A função **str** sem parâmetros retorna o String **nulo** (“”)
- ✓ Com um argumento **x** retorna **x.toString()**;
- ✓ Com **mais de um argumento** x . . . y retorna a **concatenação** desses argumentos !



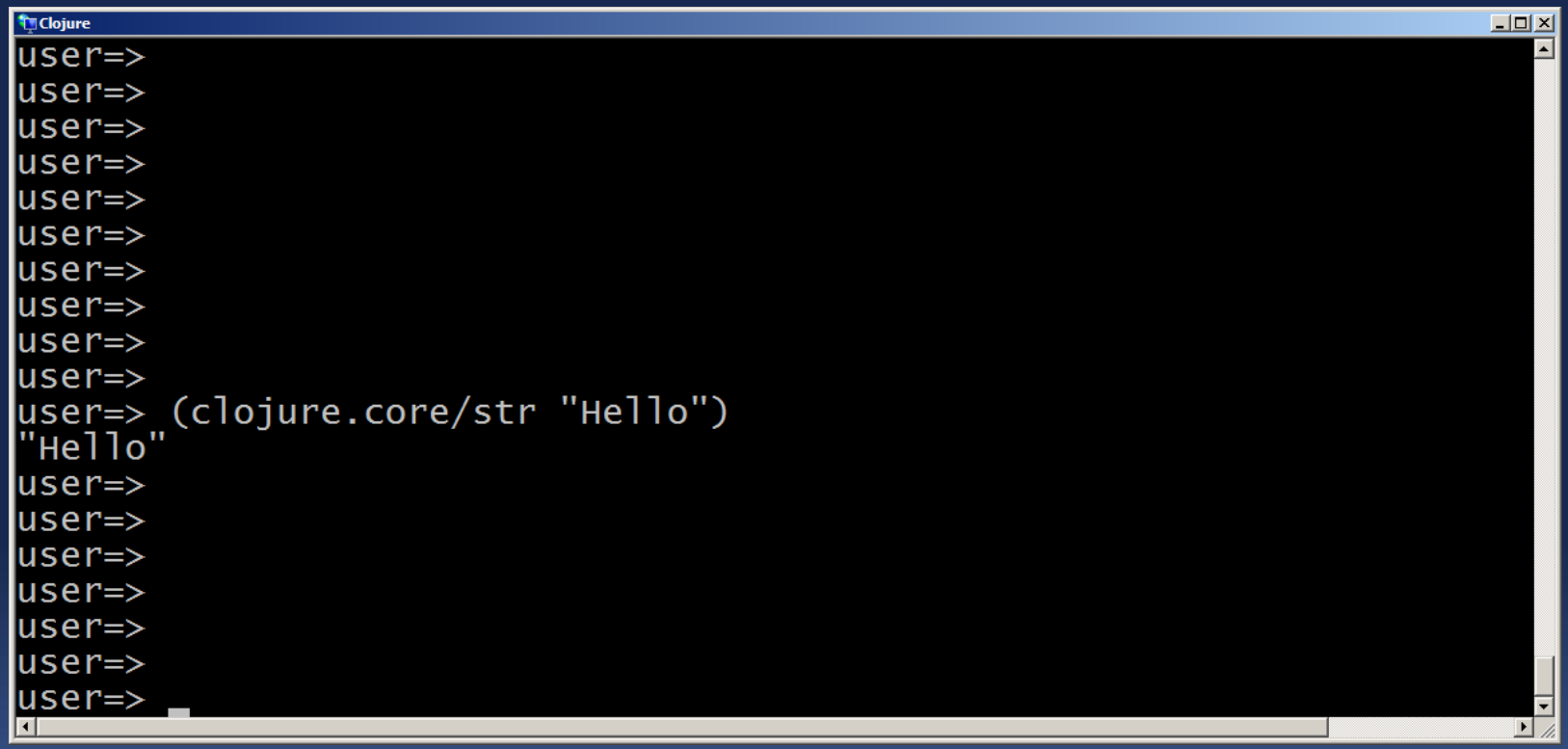
Função doc

- ✓ A função **doc** permite que se pesquise a **documentação** através de **REPL**;
- ✓ Dado que se conhece o nome da função que se quer usar, pode-se ler a documentação desta função por meio da função **doc**.
- ✓ Exemplo; Obtenção da documentação da função **str**:



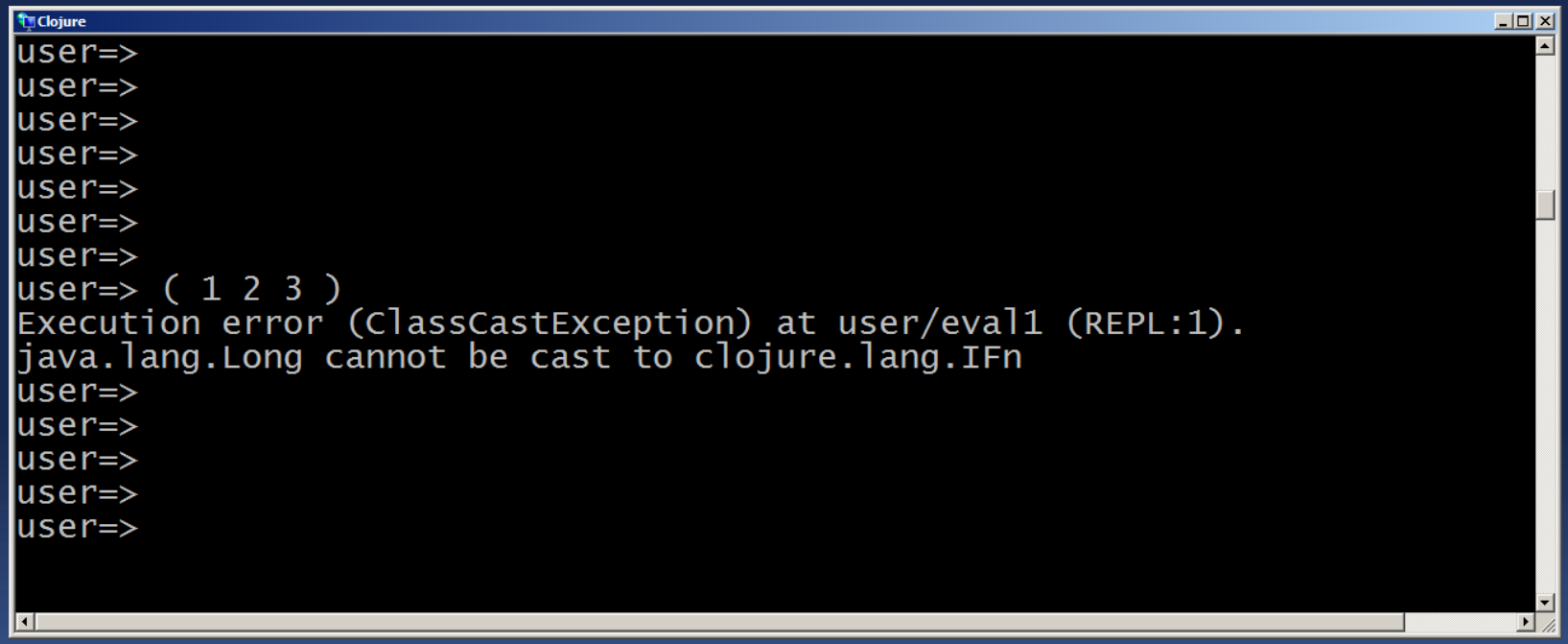
clojure.core

- ✓ Tudo o que está definido em **clojure.core** estará disponível ao seu **namespace** corrente por **default**, dispensando-se assim de se explicitamente requerê-la;
- ✓ Por exemplo: **(str "hello")** ao invés de **(clojure.core/str "hello")**



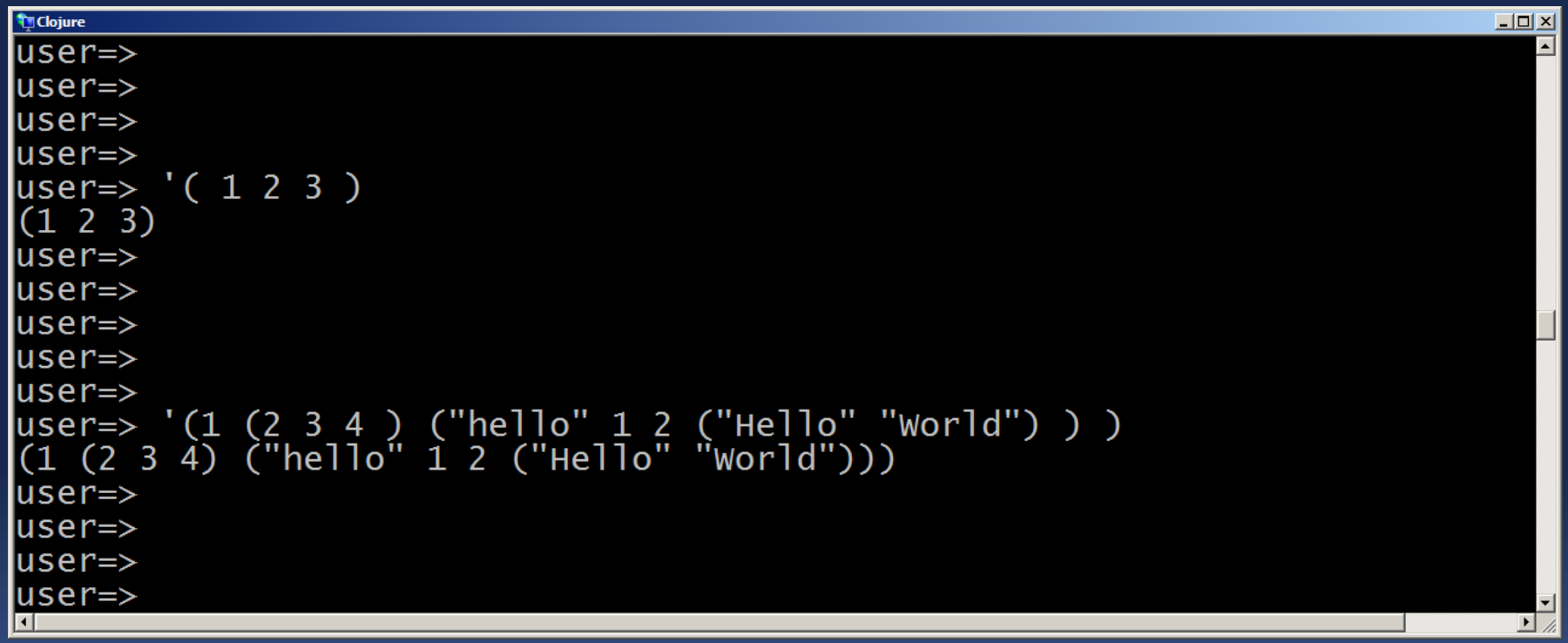
Listas

- ✓ São usadas para representar funções;
- ✓ Ao usarmos listas para representar **dados**, Clojure retorna erro:



Listas como dados

- ✓ Para que listas não sejam interpretadas como funções, mas simplemente como dados, devemos circundá-las por apóstrofe (');



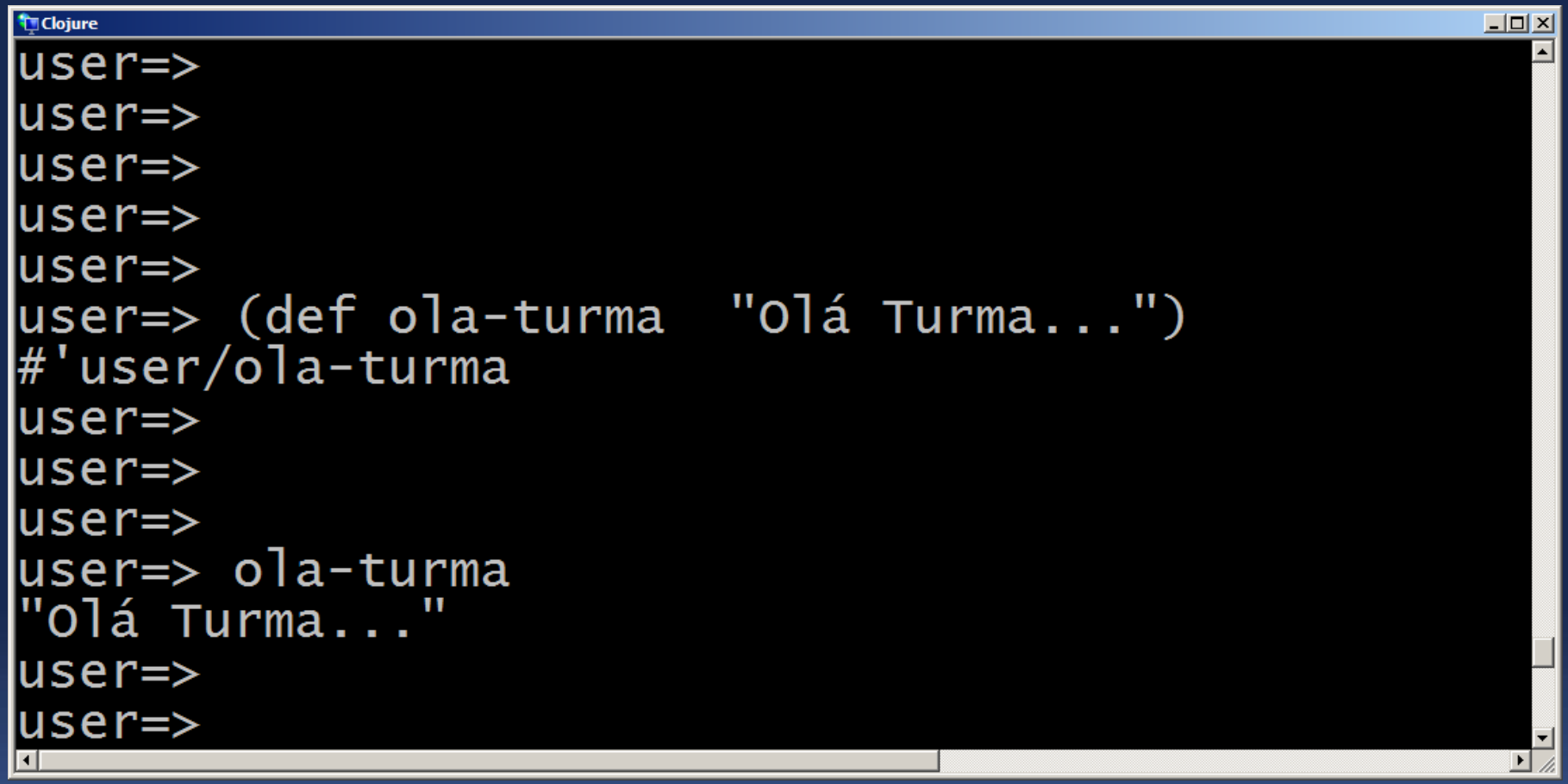
Special Forms

- ✓ Vimos até agora códigos nos quais **Clojure** emprega regras simples de avaliação incorporadas em listas;
- ✓ **Mas**, existem alguns comportamentos que **não** são tratados como nas avaliações usuais;
- ✓ Por exemplo, argumentos passados para uma função sempre são avaliados, mas e se **não** quizermos que todos os argumentos sejam avaliados ?
- ✓ Essas regras especiais de avaliação ocorrem nos "**Special Forms**".
- ✓ Por exemplo, a "**special form**" **if** pode **não** avaliar um de seus argumentos, dependendendo do resultado da avaliação do primeiro argumento;

Bindings

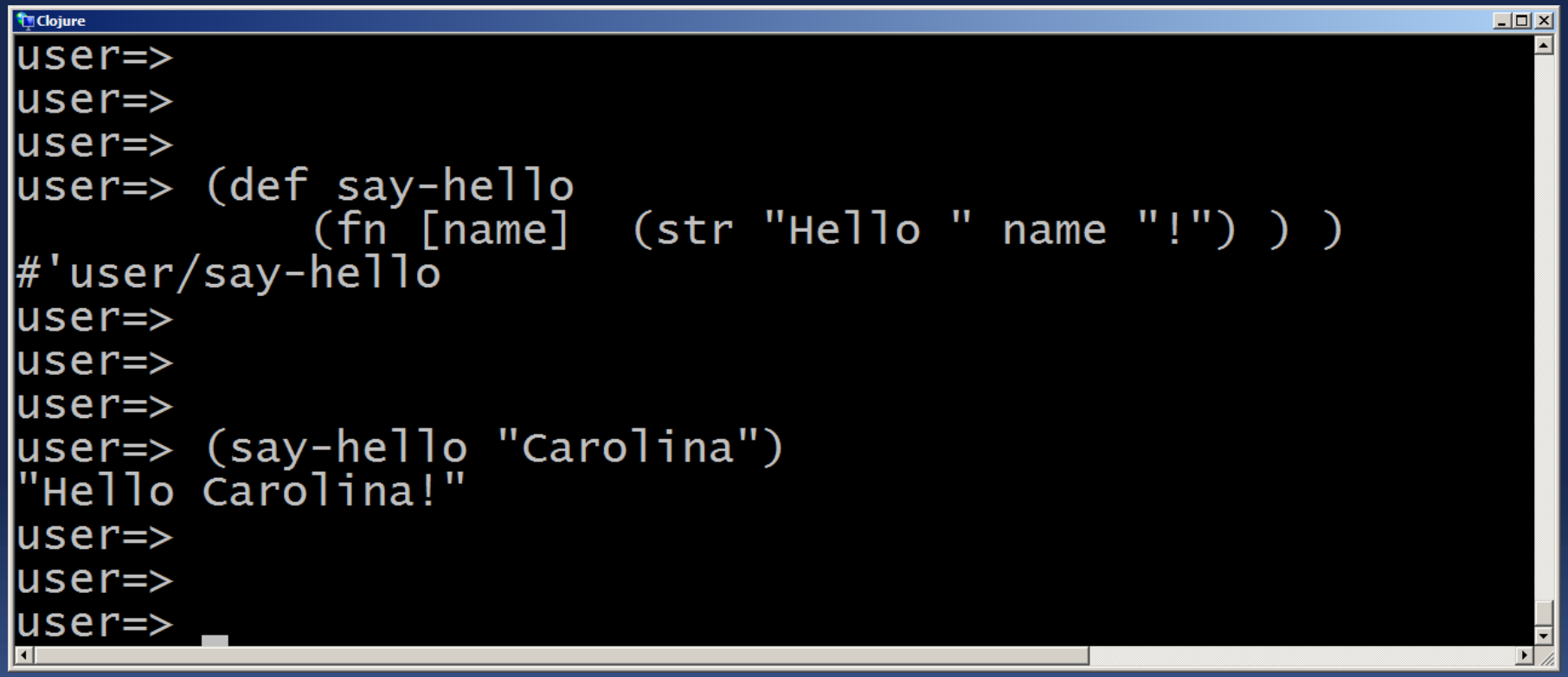
- ✓ Em **Clojure**, **Binding** significa **ligação** de símbolos à valores;
- ✓ O termo **Binding** é preferível ao invés de atribuição, pois essas ligações quase sempre são feitas uma única vez;
- ✓ Diferentemente de atribuição em variáveis que, na grande maioria das linguagens, ocorrem com frequência justificando assim o uso no nome de variáveis;
- ✓ Em **Clojure**, o nome **símbolo** é preferível ao nome **variável**;
- ✓ Bindings podem ser feitos em **Clojure** de forma local (função let) ou de forma global (função def).

Macro - def

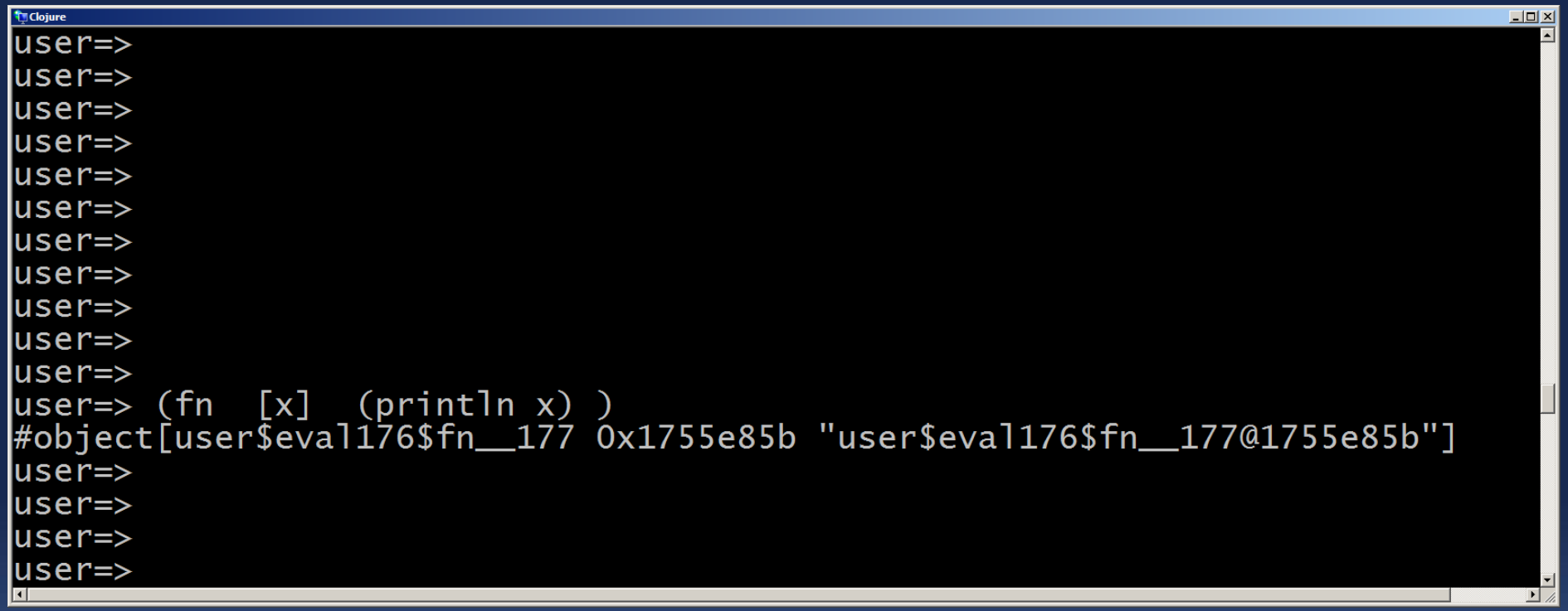

Clojure
user=>
user=>
user=>
user=>
user=>
user=> (def ola-turma "Olá Turma...")
#'user/ola-turma
user=>
user=>
user=>
user=> ola-turma
"Olá Turma..."
user=>
user=>

Macro - def

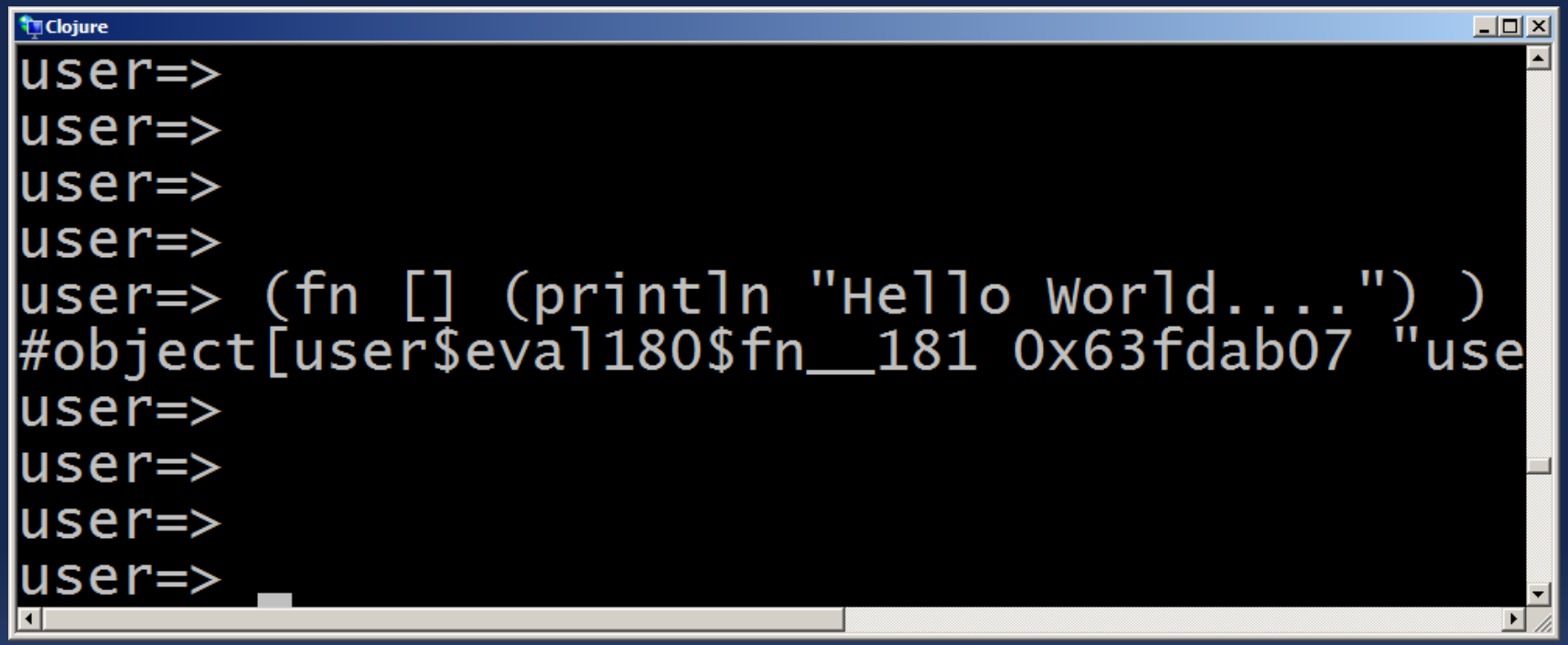
- ✓ Pode ser usada para se associar um **nome** à uma função;



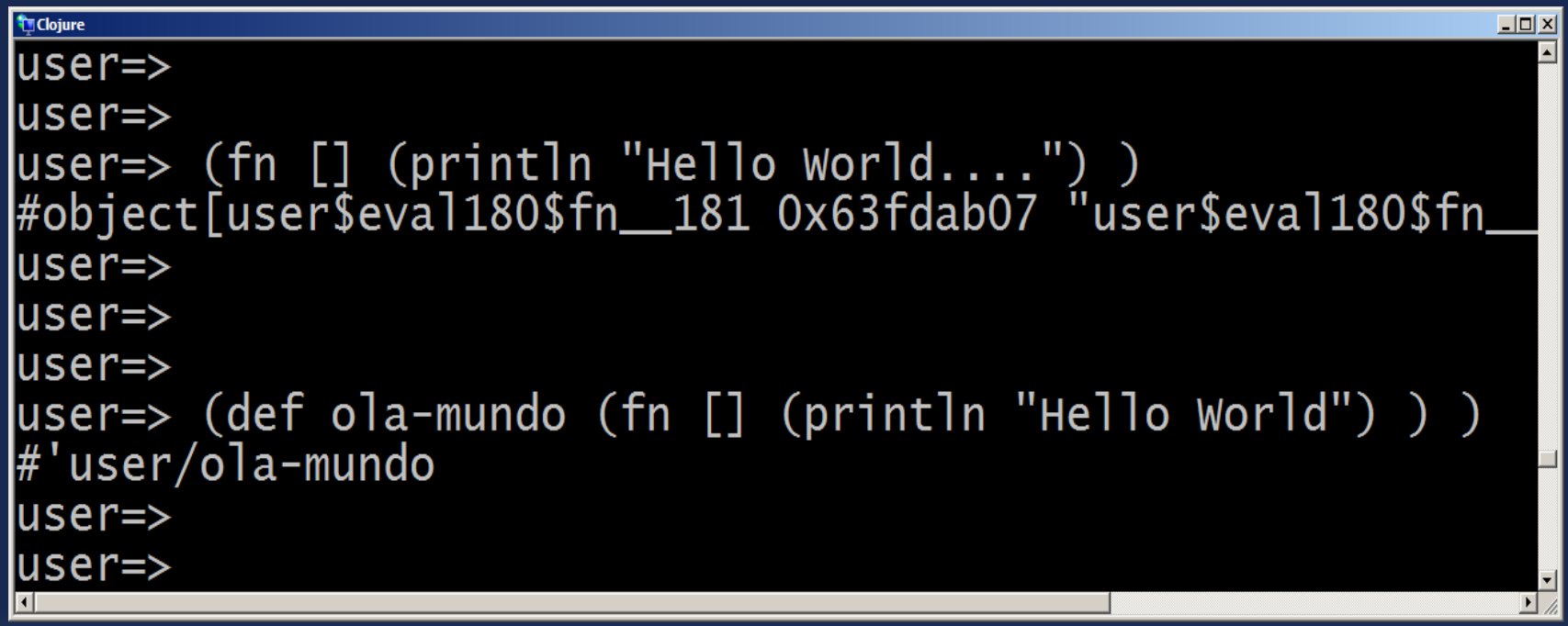
Macro - fn


Clojure
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> (fn [x] (println x))
#object[user$eval176$fn__177 0x1755e85b "user$eval176$fn__177@1755e85b"]
user=>
user=>
user=>
user=>

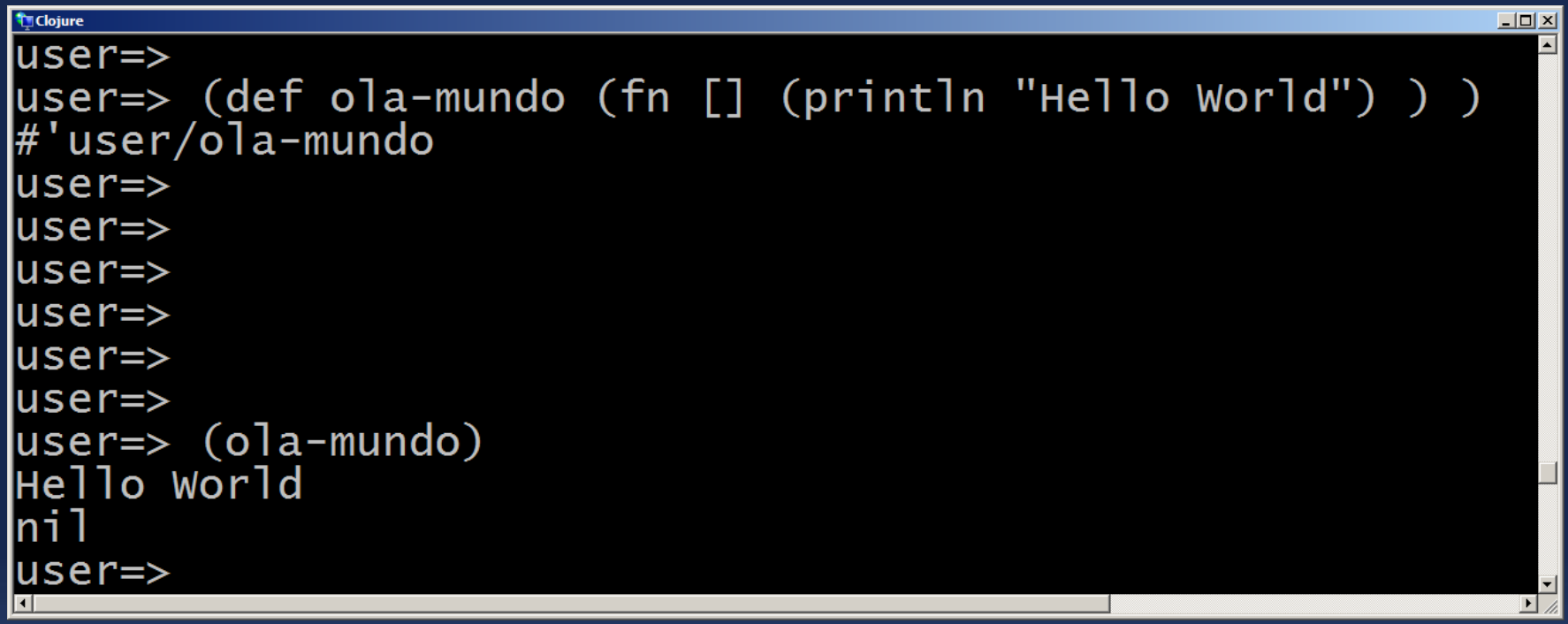
Macro - fn


Clojure
user=>
user=>
user=>
user=>
user=> (fn [] (println "Hello World...."))
#object[user$eval180$fn__181 0x63fdab07 "use
user=>
user=>
user=>
user=>

Macro - fn


Clojure
user=>
user=>
user=> (fn [] (println "Hello World...."))
#object[user$eval180$fn__181 0x63fdab07 "user$eval180$fn__
user=>
user=>
user=>
user=> (def ola-mundo (fn [] (println "Hello World")))
#'user/ola-mundo
user=>
user=>

Macro - fn

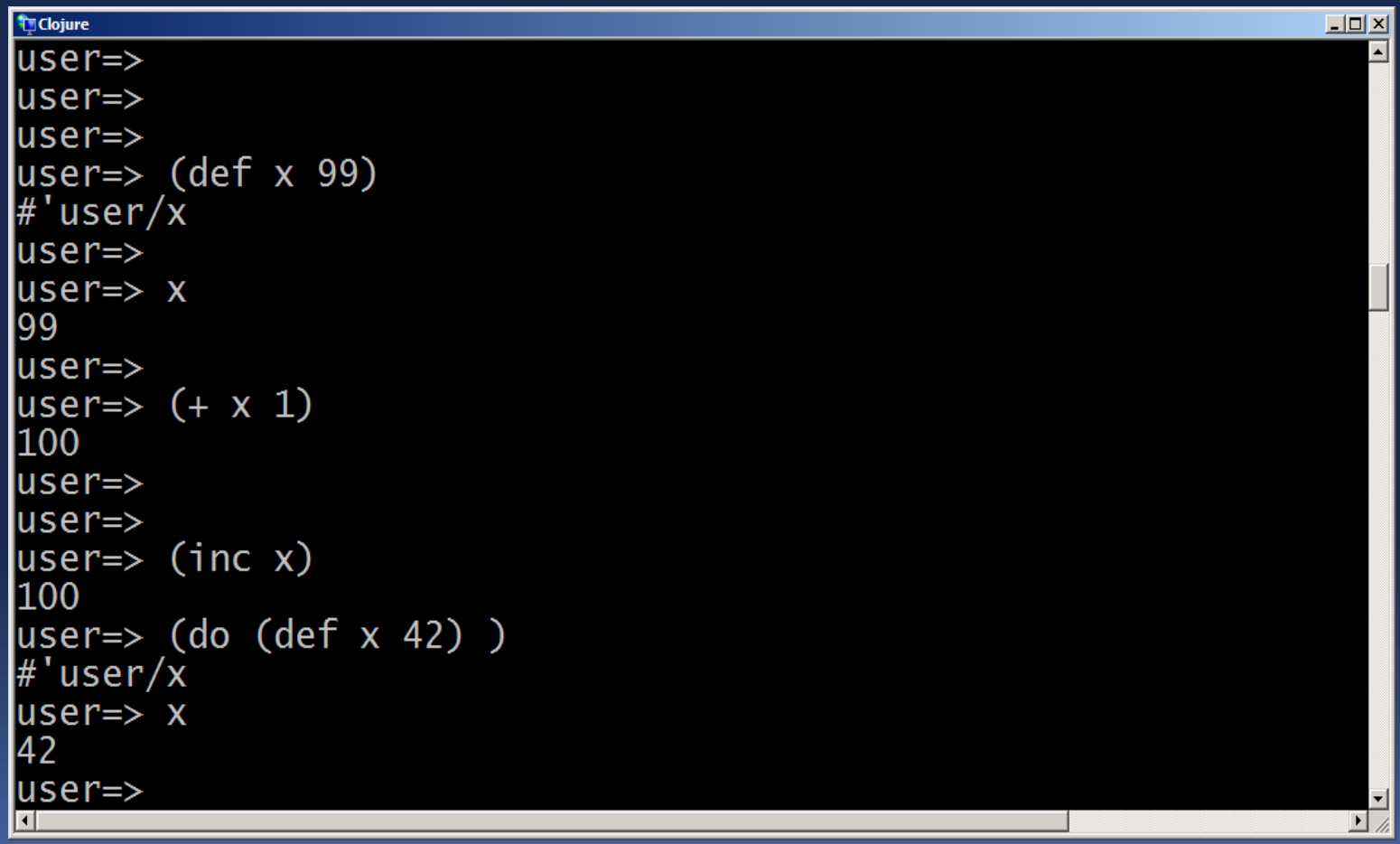

Clojure
user=>
user=> (def ola-mundo (fn [] (println "Hello World")))
#'user/ola-mundo
user=>
user=>
user=>
user=>
user=>
user=>
user=> (ola-mundo)
Hello World
nil
user=>

Binding

- ✓ É possível redefinir um símbolo criado previamente ?
- ✓ Sim, é possível modificar-se o binding por meio de: **(def x 20)**
- ✓ **Mas**, **não** se recomenda redefinir-se símbolos em nossos programas uma vez que tal procedimento dificulta a leitura e a manutenção do código;
- ✓ Assim, em **Clojure** uma boa prática é considerar a operação de binding como constante.

Binding

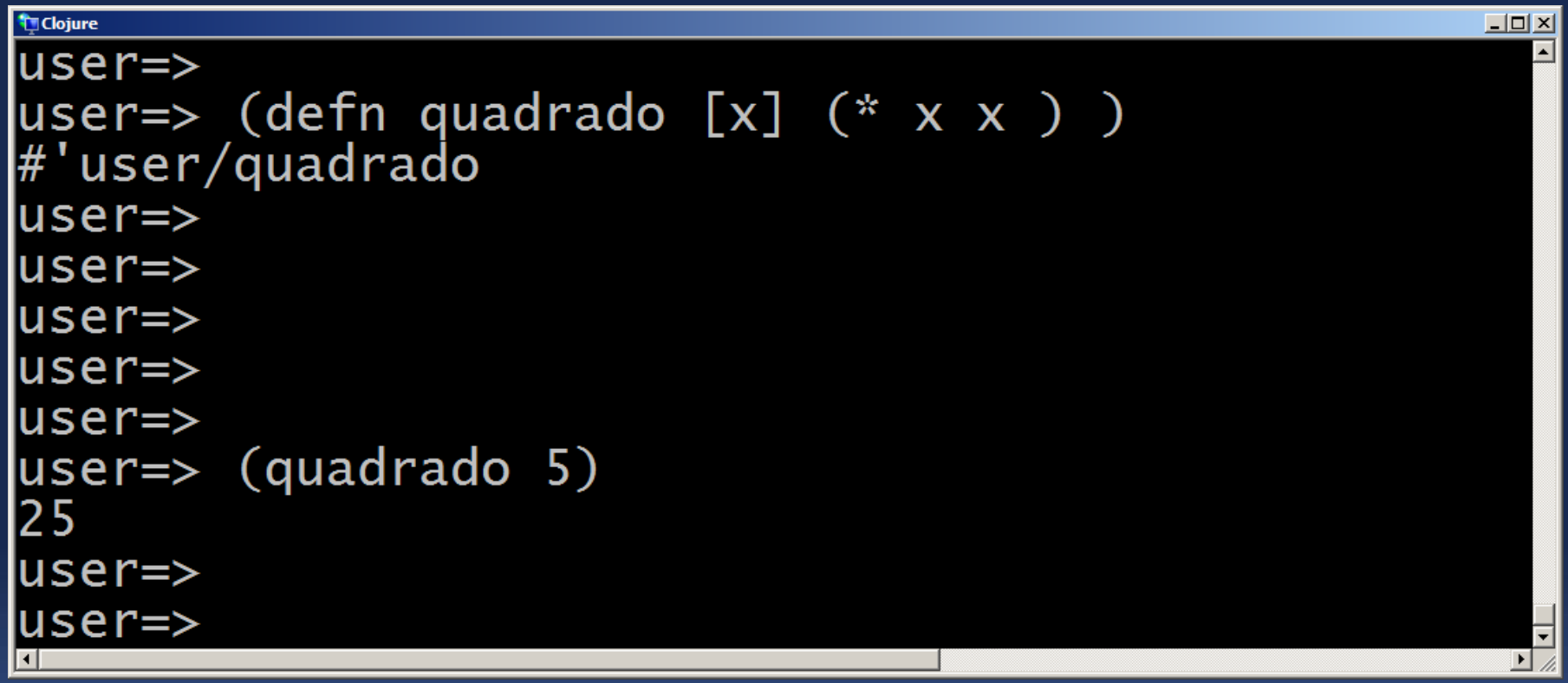
- ✓ Uma vez definido um símbolo por meio de **binding**, pode-se utilizá-lo em avaliação de expressões e redefiní-lo ainda por **binding**;



Macro defn

- ✓ Vimos nas atividades anteriores que com a special form **fn** podemos criar funções anônimas.
- ✓ Vimos também que podemos retirar da função essa característica anônima, ou seja deixá-la reusável por meio da special form **def**.
- ✓ Essa combinação de **def** com **fn** é portanto bem usual;
- ✓ Em função disso, **Clojure** disponibiliza uma macro built-in para essa finalidade.
- ✓ Trata-se da macro **defn**.

Exemplo - Macro defn



Estruturas de Dados

Estruturas de Dados

- ✓ Strings
- ✓ Números
- ✓ Valores Booleanos
- ✓ Coleções

Coleções

- ✓ Maps
- ✓ Sets
- ✓ Vectors
- ✓ Lists

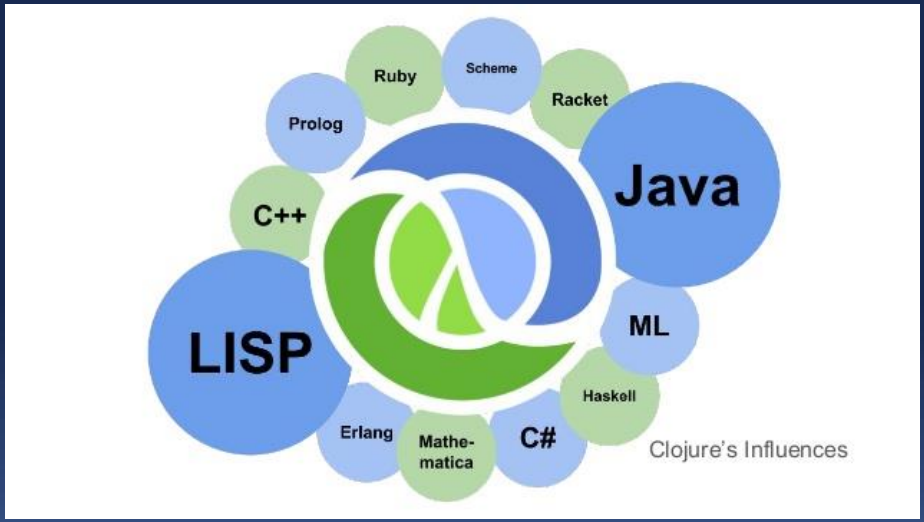
Interoperabilidade com Java

Interoperabilidade com Java

- ✓ Recomenda-se a criação do projeto com **Leiningen**;
- ✓ Um projeto nos ajuda a organizar o nosso código;
- ✓ O nosso projeto está estruturado em **namespaces**;
- ✓ Criamos novos **namespaces** e importamos bibliotecas externas a fim de usá-las em nosso código;

Interoperabilidade com Java

- ✓ A Linguagem **Clojure** é compilada para bytecodes e é processada na JVM;
- ✓ Assim, por causa disso **Clojure** é chamada linguagem hosted;



Usando Java em Clojure

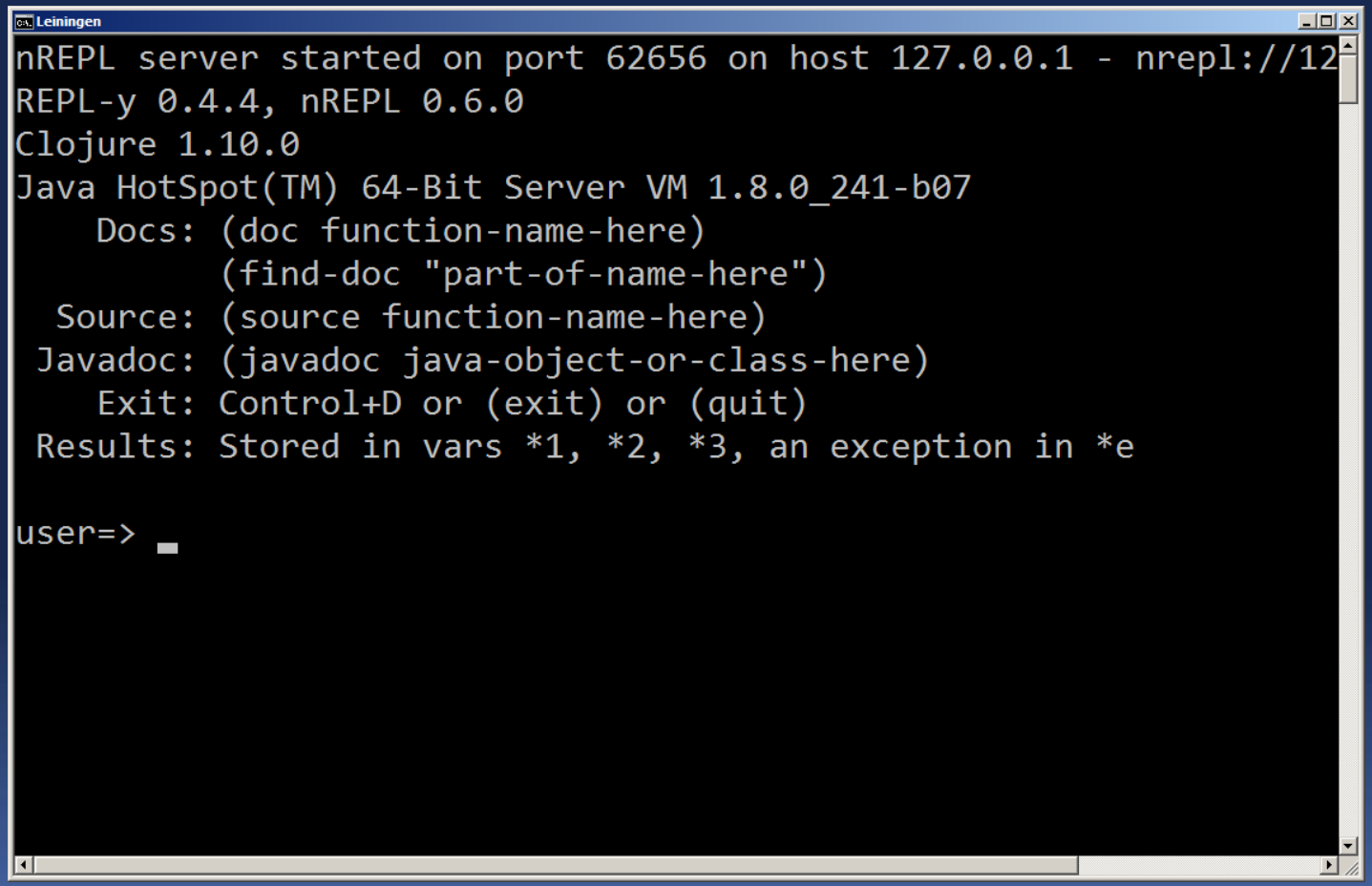
- ✓ **Clojure** é hosted language. Isso significa que **Clojure** usa JVM ao invés de criar um novo ambiente de runtime;
- ✓ Assim, Clojure reusa todas as facilidades providas pela JVM;
- ✓ Esta é uma poderosa característica de Clojure;
- ✓ Coisas tais como Garbagge Collection, threading, concorrência, operações de I/O que são largamente já testadas e consolidadas, são incorporadas ao **Clojure**.

Usando Java em Clojure

- ✓ Assim, todo desenvolvedor **Clojure** tem acesso ao ecosistema de bibliotecas JVM;
- ✓ Além disso, considerando que Java é uma das mais populares linguagens de programação, há uma comunidade muito grande e atuante, trazendo para os desenvolvedores **Clojure** os benefícios de usar bibliotecas bem testadas e otimizadas;
- ✓ Veremos agora como importar a classe Java **BigDecimal**.

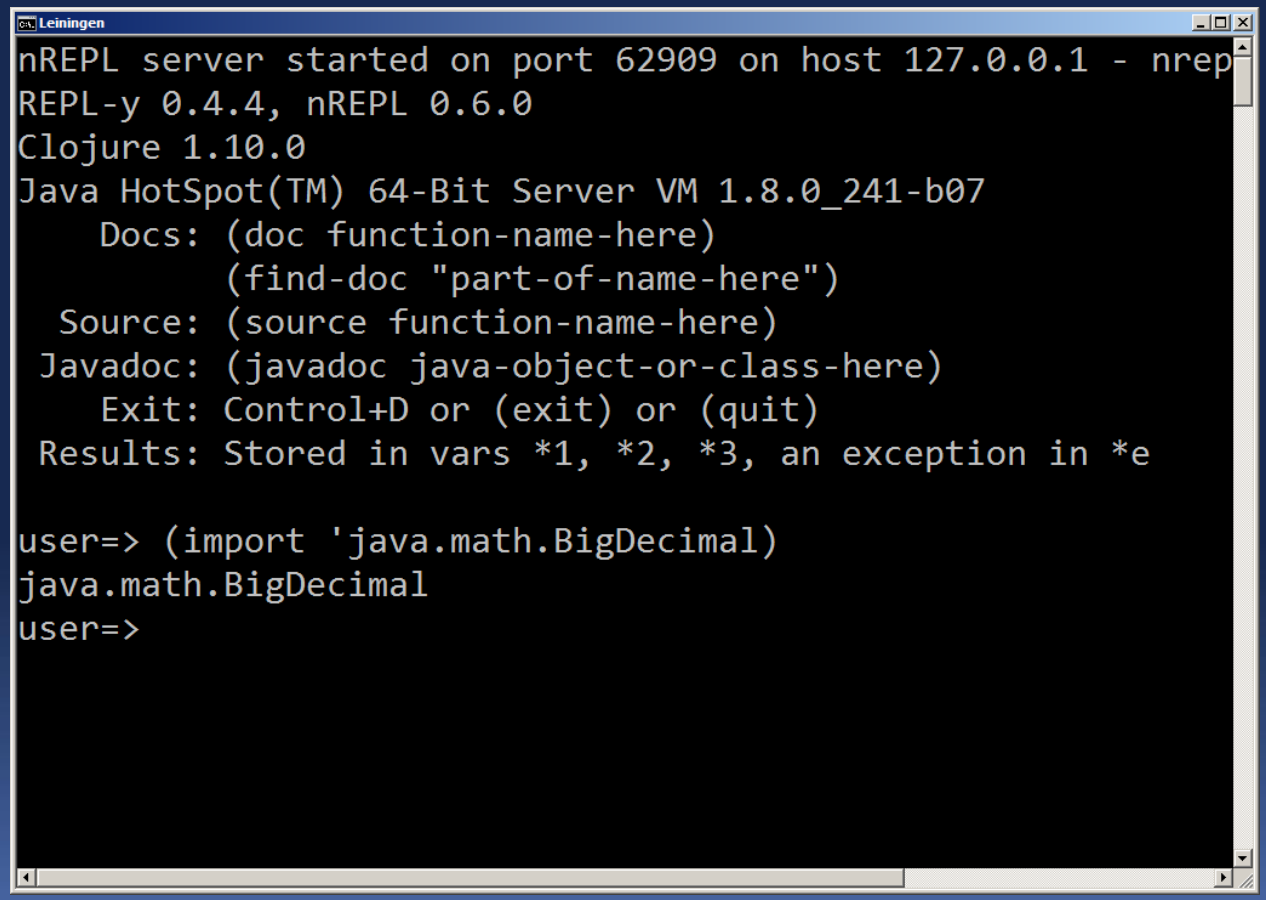
Importando a classe Java BigDecimal

lein repl



Importando a classe Java BigDecimal (import ...)

- ✓ A classe BigDecimal é chamada por meio da função import.

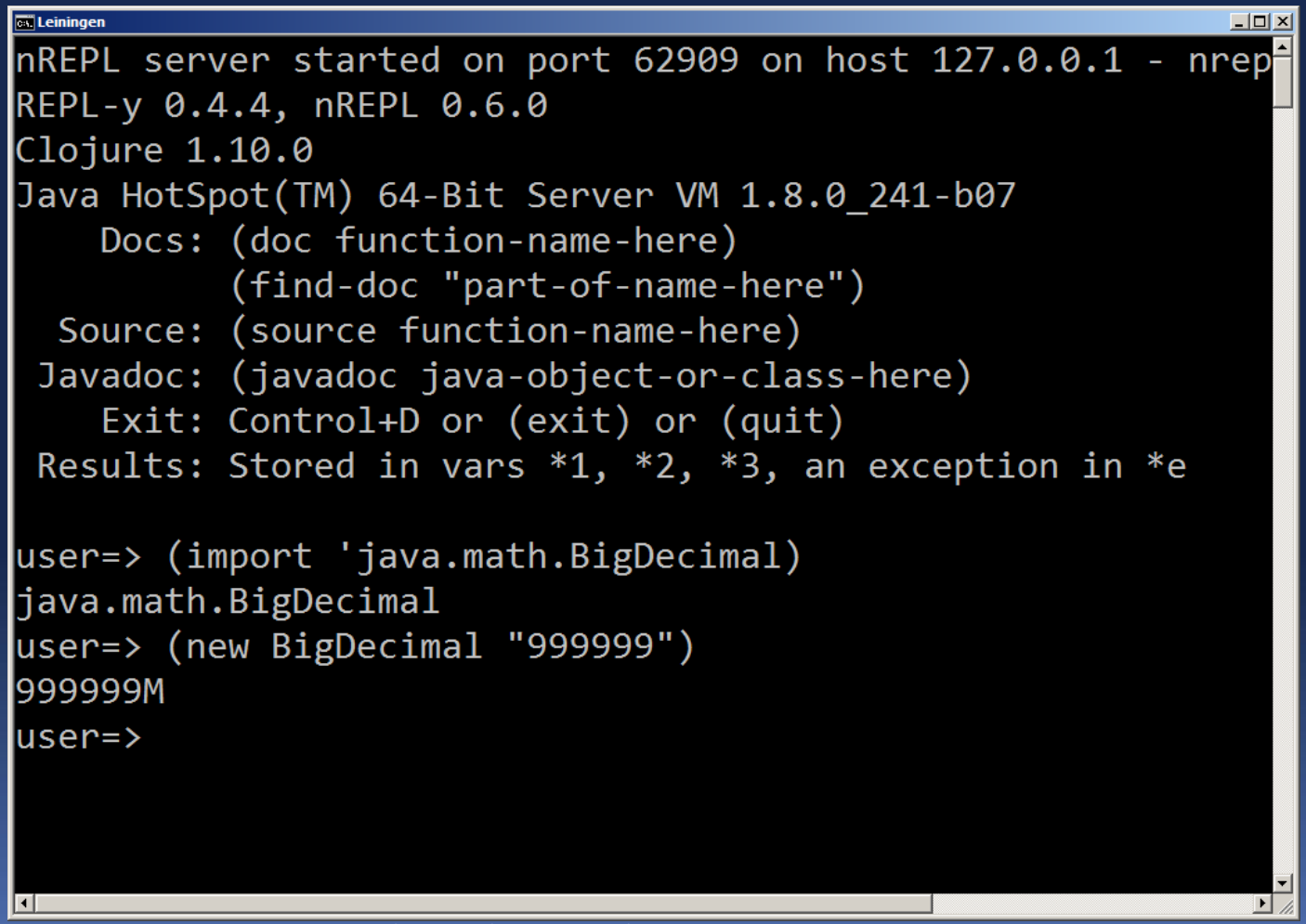


Importando a classe Java BigDecimal

- ✓ Em Java, criamos uma instância da classe BigDecimal por meio de:

```
BigDecimal grandeNumero = new BigDecimal("999999");
```

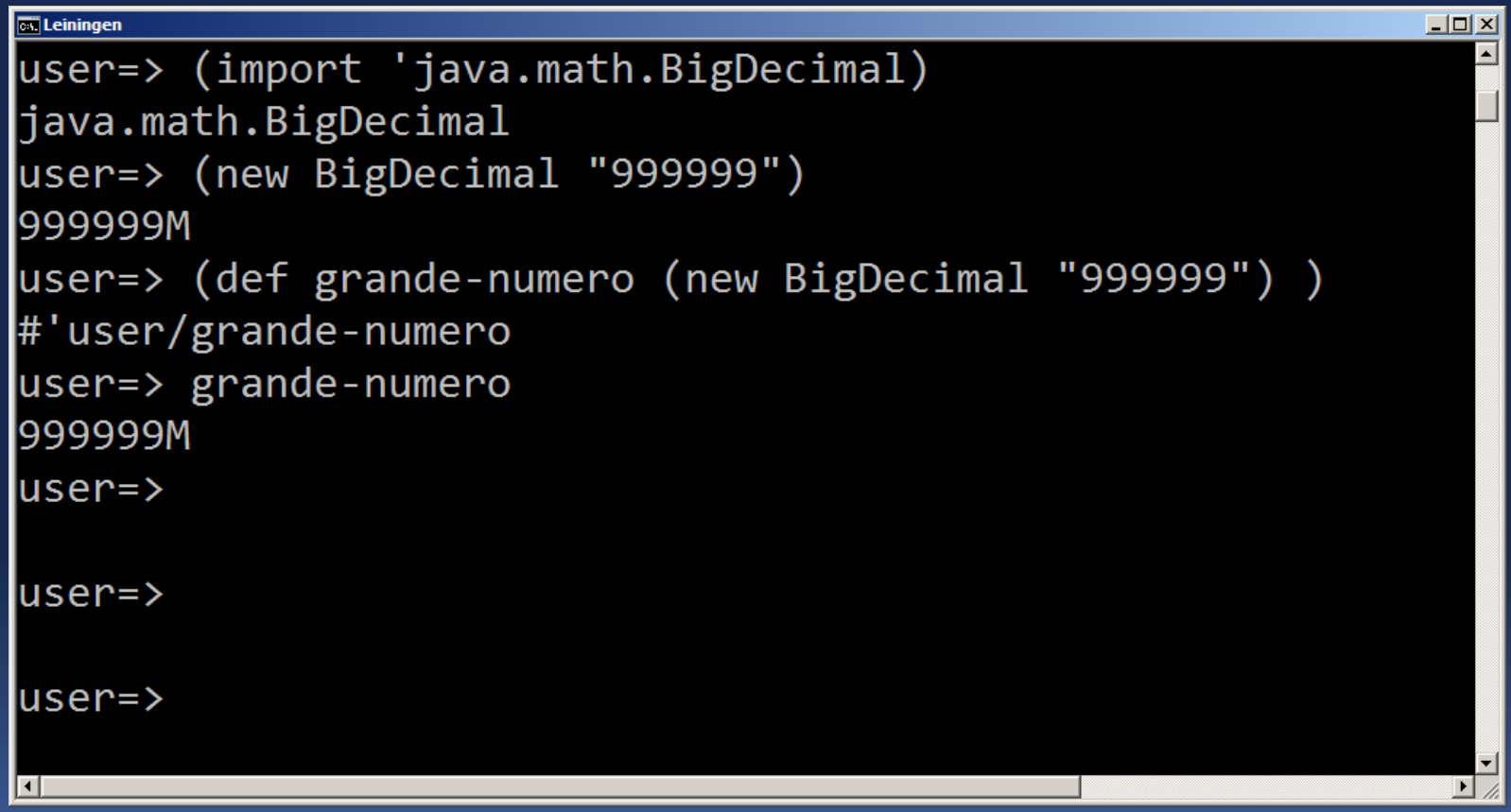
- ✓ Similarmente, em Clojure escrevemos: (new BigDecimal "999999")



Importando a classe Java BigDecimal

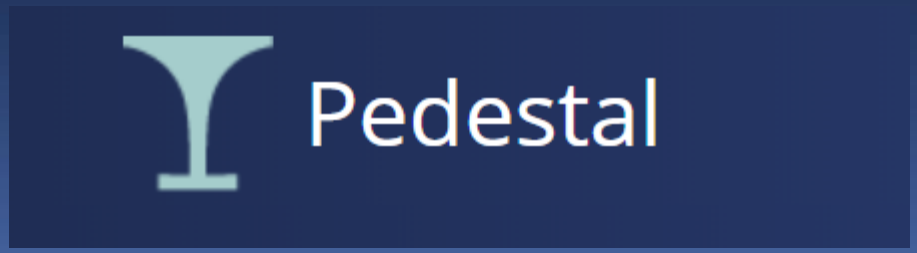
- ✓ Se quizermos usar várias vezes, podemos fazer o binding em um símbolo

(def grande-numero (new BigDecimal “999999”)



Desenvolvimento de API's em Clojure

Criação de API's Rest em Clojure com o Framework Pedestal



Setup do Ambiente

- Para a criação da API Rest, utilizaremos a biblioteca Pedestal.

http://pedestal.io/

pedestal.io

Pedestal

REFERENCE API GUIDES COOKBOOK SAMPLES COMMUNITY

Pedestal is a sturdy and reliable base for services and APIs.

Get Pedestal

What is Pedestal?

Pedestal is a set of libraries that we use to build services and applications. It runs in the back end and can serve up whole HTML pages or handle API requests.

There are a lot of tools in that space, so why did we build Pedestal? We had two main reasons:

- **Pedestal is designed for APIs first.** Most web app frameworks still focus on the "page model" and server side rendering. Pedestal lets you start simple and add that if you need it.
- **Pedestal makes it easy to create "live" applications.** Applications must respond with immediate feedback even while some back-end communication goes on. Pedestal makes it easy to deliver server-sent events and asynchronous updates.

```
(ns front-page
  (:require [io.pedestal.http :as http]))

(defn respond-hello [request]
  {:status 200
   :body   "Hello, world!"})

(def routes
  #{["/greet" :get `respond-hello]})
```

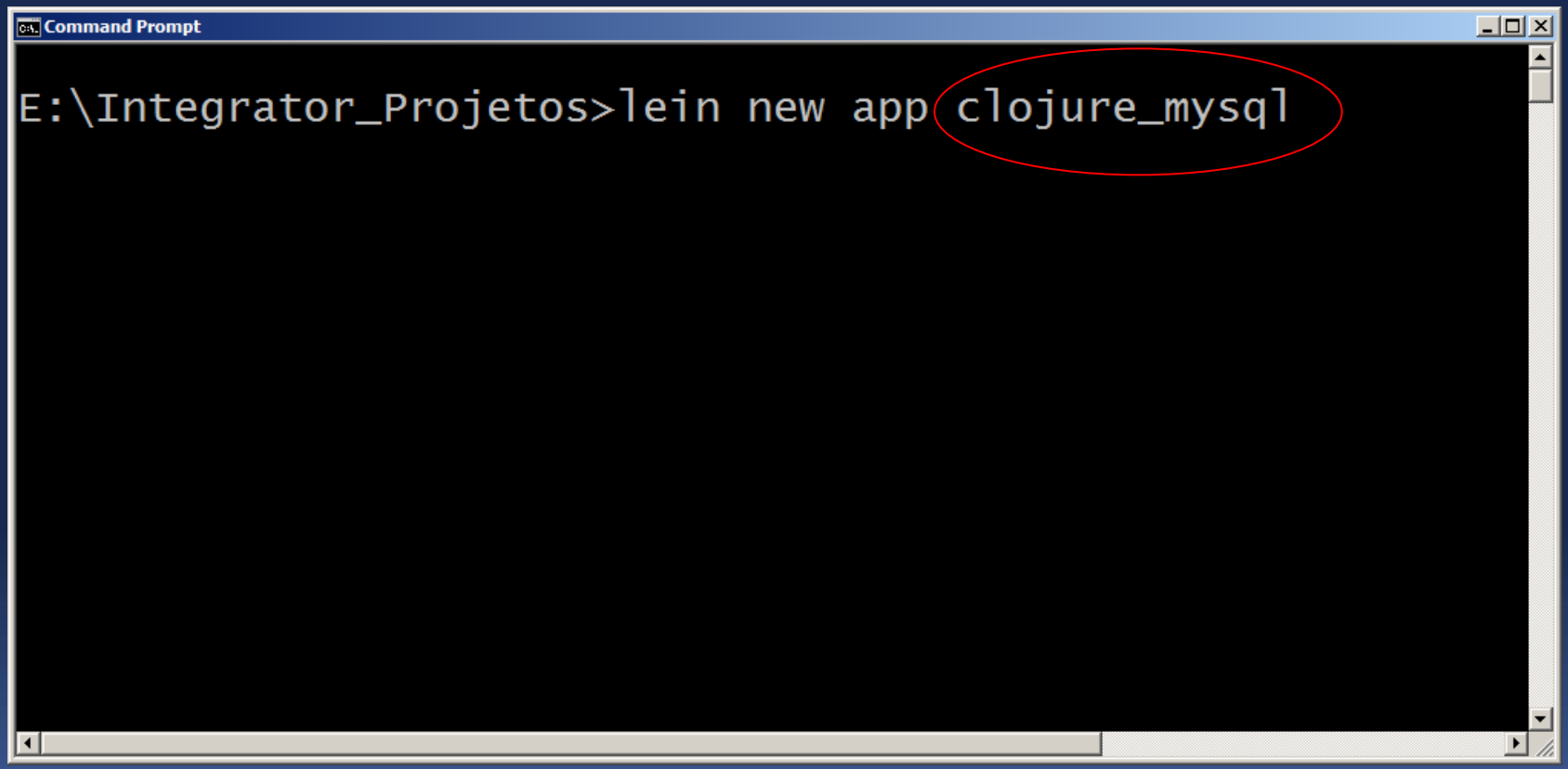
O que é Pedestal ?

- É um conjunto de bibliotecas que são usadas para se criar serviços e aplicações.
- Roda no back end e pode servir requests handle **API**;
- Desenvolveremos nesta unidade uma api simples para demonstrar o uso da biblioteca Pedestal;
- Inicialmente, faremos uma aplicação gerando o projeto a partir do zero !

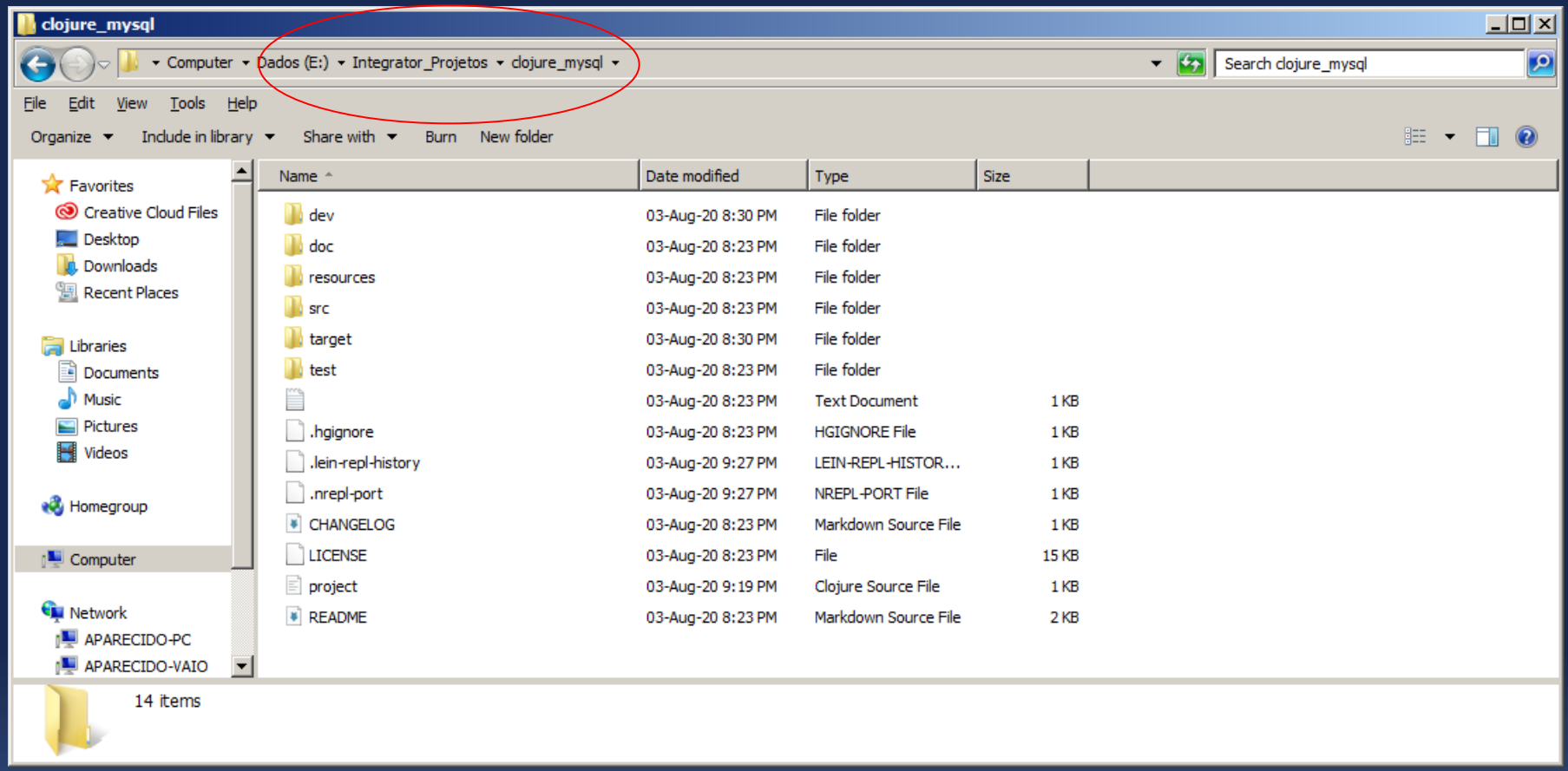
Acessando MySQL com projeto Leiningen

Recursos

- Criando o projeto com Lein



Projeto criado



Configurando o projeto

- Incluindo no arquivo **project.clj** as dependências do **jdbc** e do Driver jdbc **MySQL**

project.clj

```
(defproject clojure_mysql "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception-2.0"
            :url "https://www.eclipse.org/legal/epl-2.0/"}

  :repl-options {:init-ns user}

  :dependencies [ [org.clojure/clojure "1.10.1"]
                  [org.clojure/java.jdbc "0.7.9"]
                  [mysql/mysql-connector-java "8.0.21"]]

  :jvm-opts ["-Dclojure.server.myrepl={:port,5555,:accept,clojure.core.server/repl}"]

  :main ^:skip-aot clojure_mysql.core
  :target-path "target/%s"
  :profiles {:uberjar {:aot :all
                       :jvm-opts ["-Dclojure.compiler.direct-linking=true"] }
                       :dev {:dependencies []
                       :source-paths ["dev"]}})
```

Criando o Banco de Dados

- Banco de Dados: scpe
- Porta: 3307
- Host: localhost
- Tabela: cursos

```
use scpe;

describe curso;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
id_curso	int(11)	NO	PRI	NULL	auto_increment
datetime_curso	varchar(255)	YES		NULL	
nome_curso	varchar(255)	YES		NULL	
timestamp_curso	varchar(255)	YES		NULL	

```
select *  from curso;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

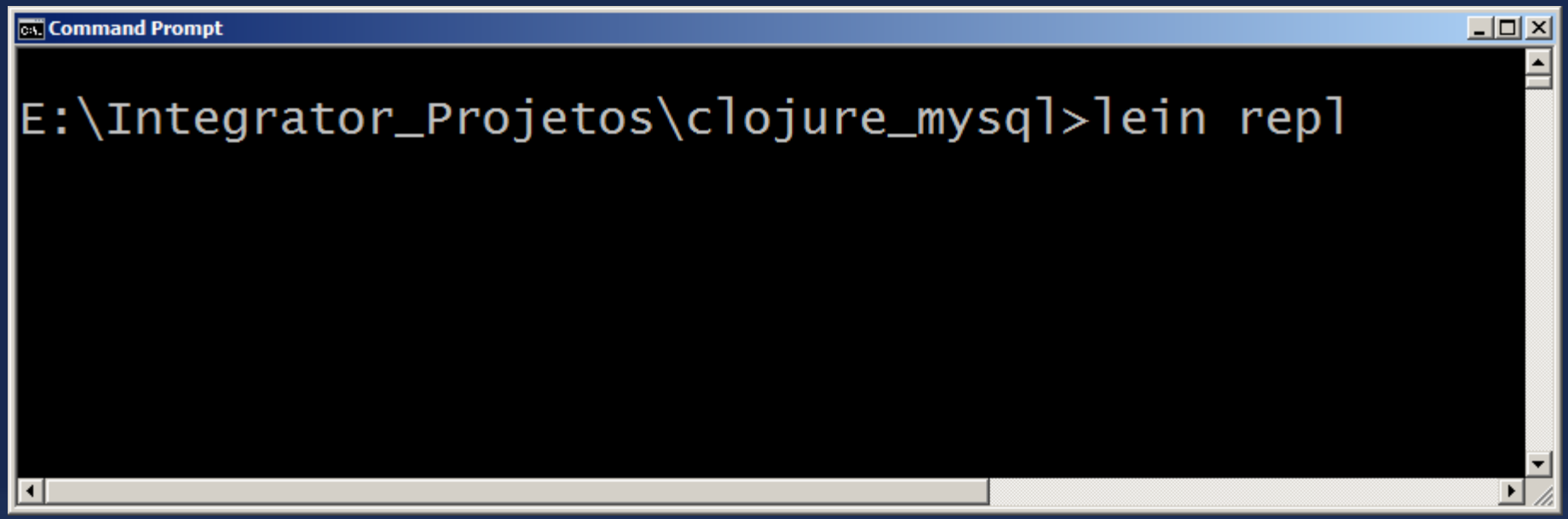
id_curso	datetime_curso	nome_curso	timestamp_curso
1	2020-07-20 21:51:15	Sistemas de Informação	2020-07-20 21:51:15
2	2020-07-20 21:51:18	Gestão de Tecnologia da Informação	2020-07-20 21:51:18
3	2020-07-20 21:51:21	Tecnologia ADS	2020-07-20 21:51:21

Código Clojure para acessar a tabela

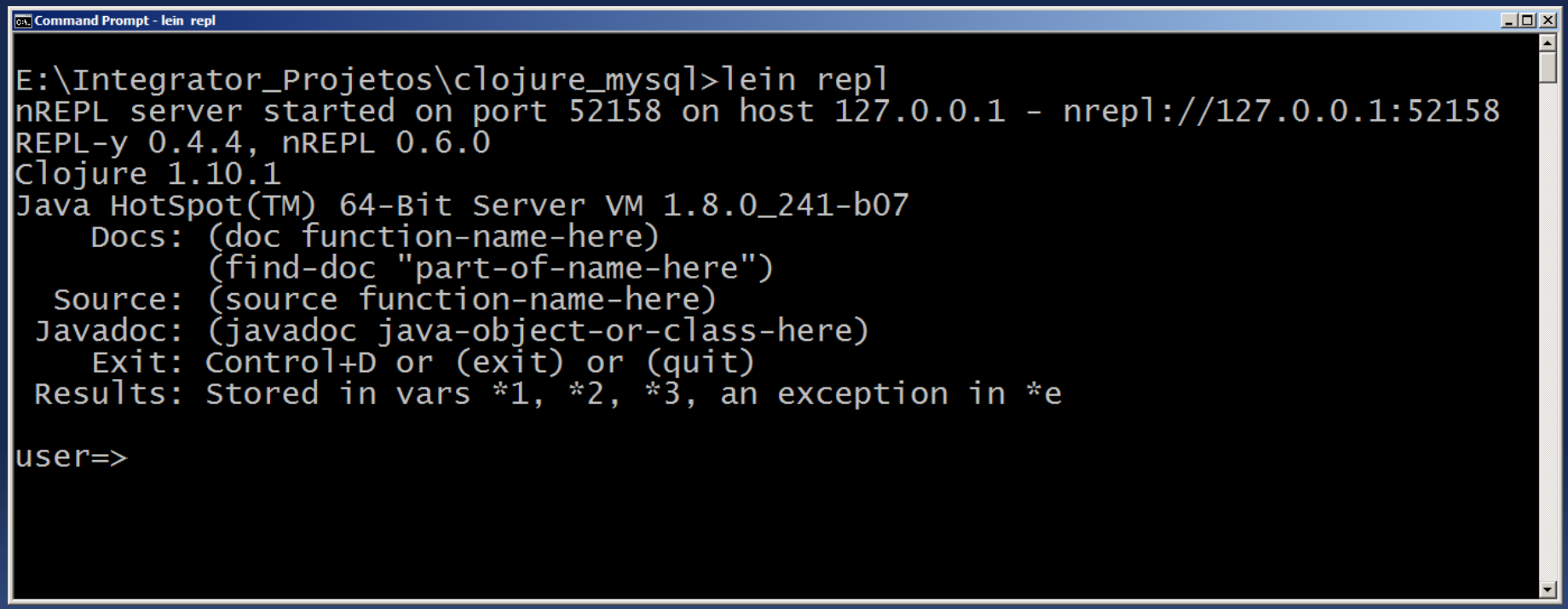
core.clj

```
(ns clojure_mysql.core
  (:gen-class))

(require '[clojure.java.jdbc :as jdbc])

(let [db-host "localhost"
      db-port 3307
      db-name "scpe"]

(def db {:classname "com.mysql.cj.jdbc.Driver"
         :subprotocol "mysql"
         :subname (str "//" db-host ":" db-port "/" db-name "?useTimezone=true&serverTimezone=UTC")
         :user "root"
         :password "maua"}))

(def resp (jdbc/query  db ["SELECT * FROM curso WHERE id_curso = 1"]))

(defn -main
  "Código clojure para acessar MySQL via jdbc"
  [& args]
  (println resp))
```

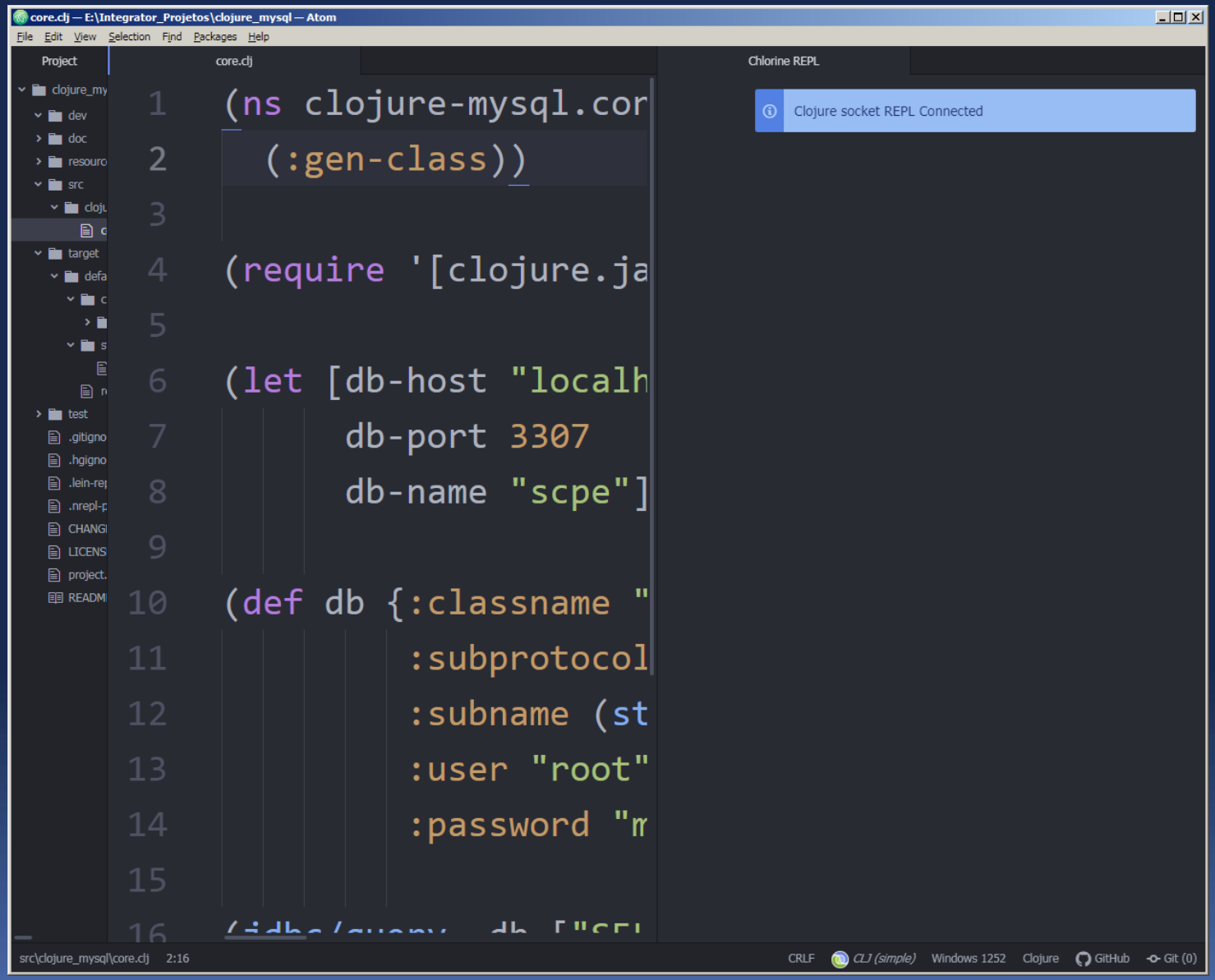
Executando sob REPL - Chlorine



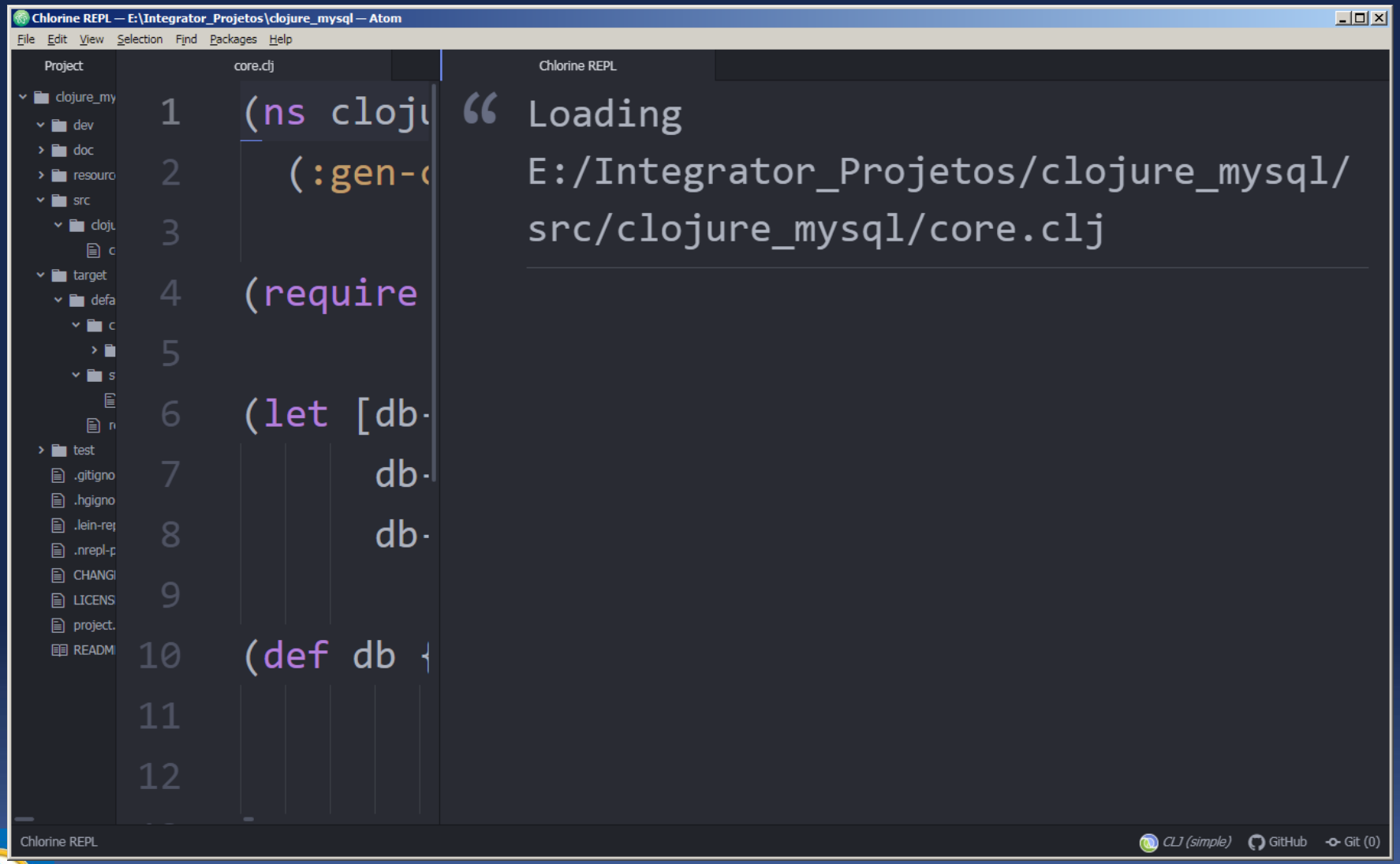
Servidor REPL ativo



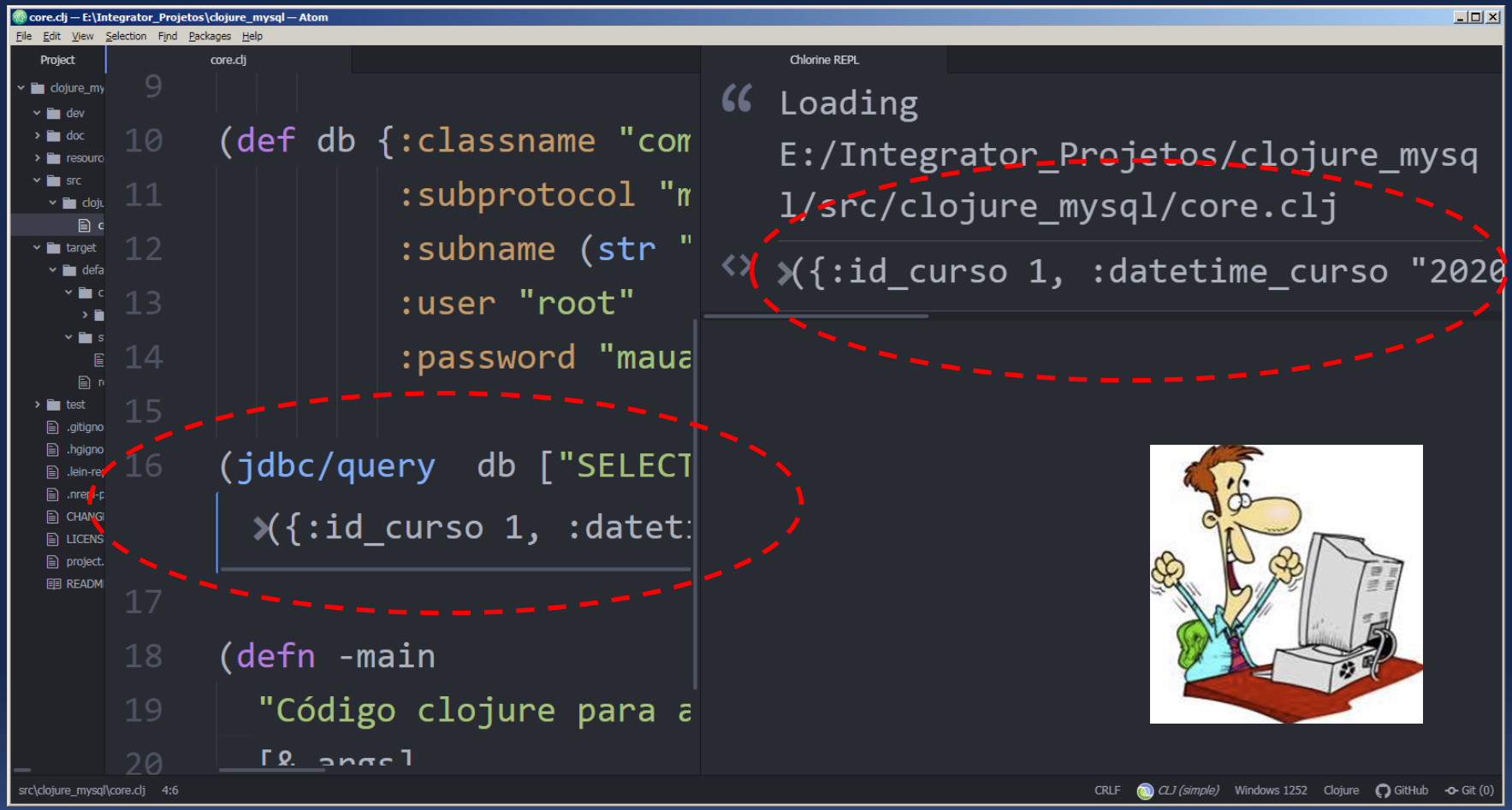
Conectando Atom Chlorine



Carregando código no REPL (Load-file)

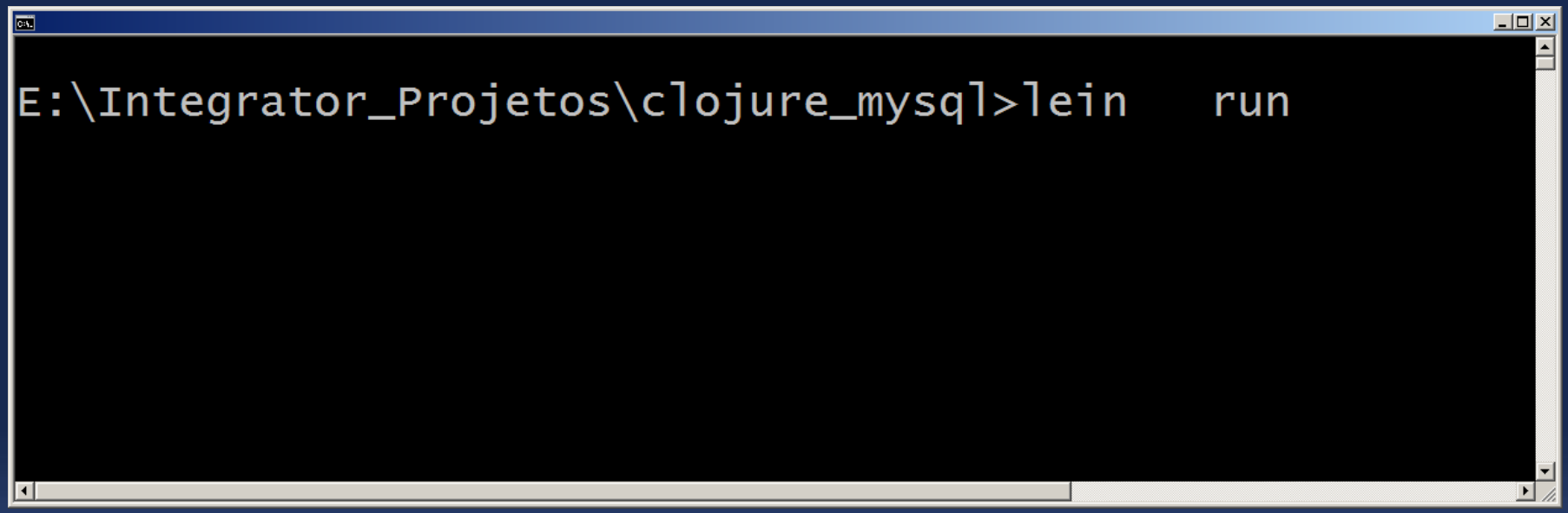


Executando sob REPl (Chlorine)



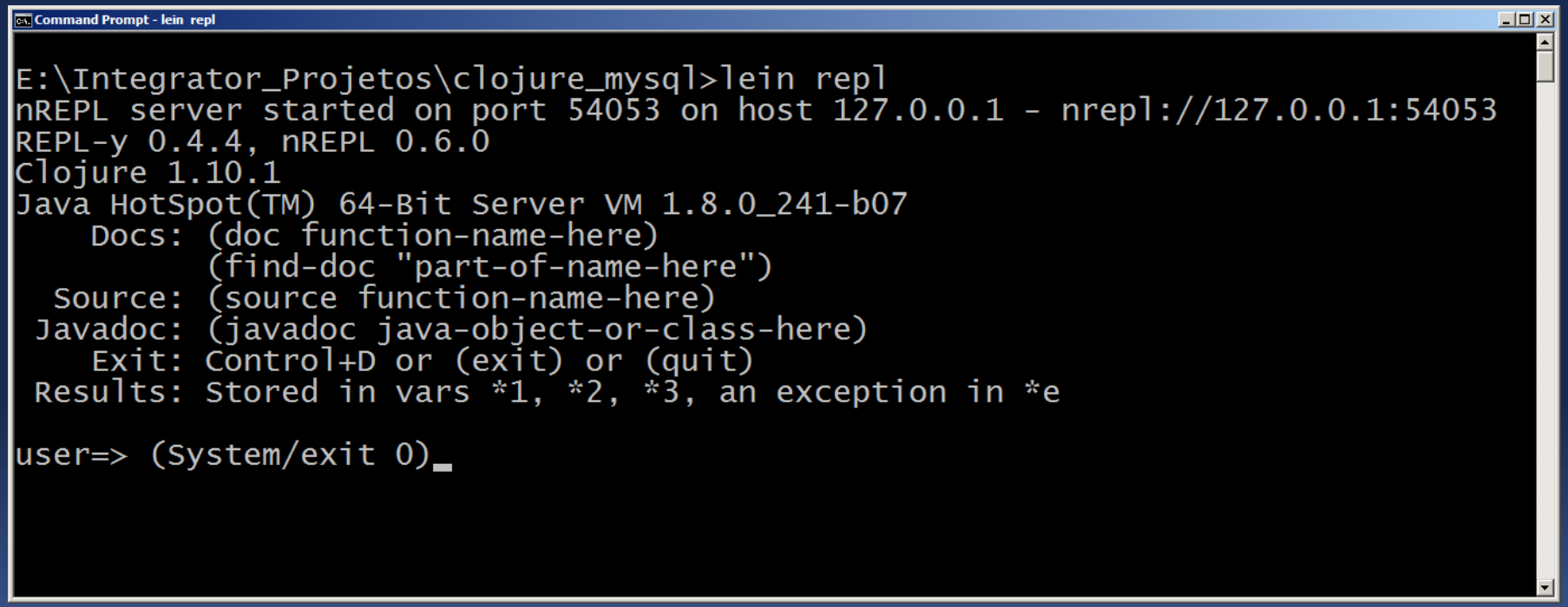
Executando – Linha de Comando REPL

No diretório do projeto > lein run



Executando – Linha de Comando REPL

- Para efetuar shutdown no REPL server => (System/exit 0)

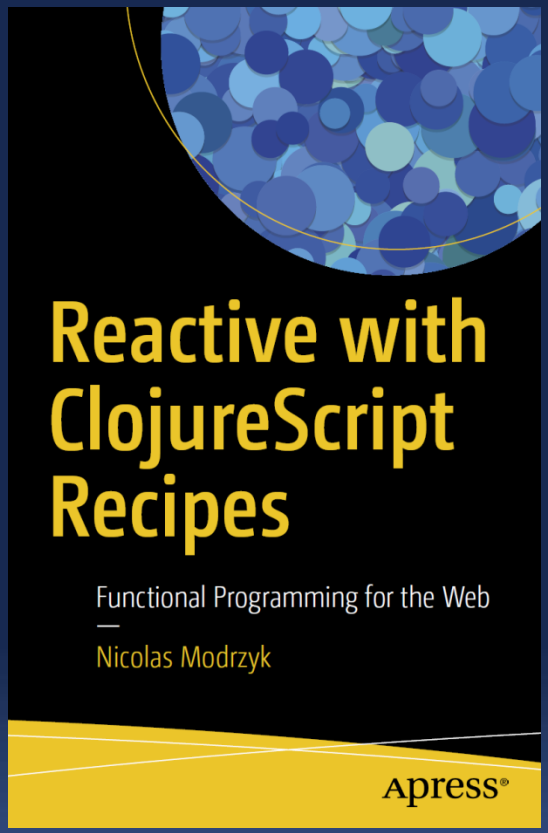


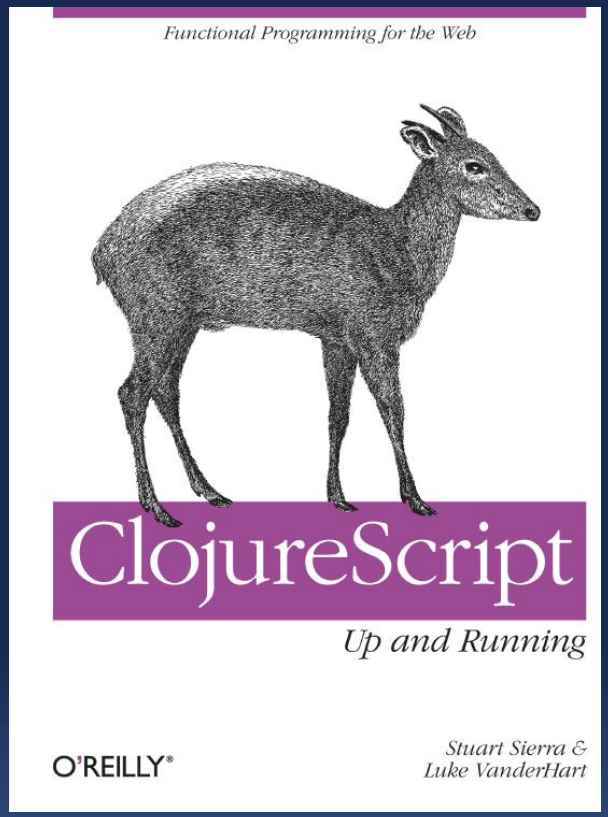
Executando – Linha de Comando REPL

```
Command Prompt

E:\Integrator_Projetos\clojure_mysql>lein run
({:id_curso 1, :datetime_curso 2020-07-20 21:51:15, :nome_curso Sistemas de Info
rma├°├úo, :timestamp_curso 2020-07-20 21:51:15})
E:\Integrator_Projetos\clojure_mysql>
```

Interação com JavaScript

ClojureScript





- ✓ aparecido.freitas@prof.uscs.edu.br
- ✓ aparecidovfreitas@gmail.com