



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



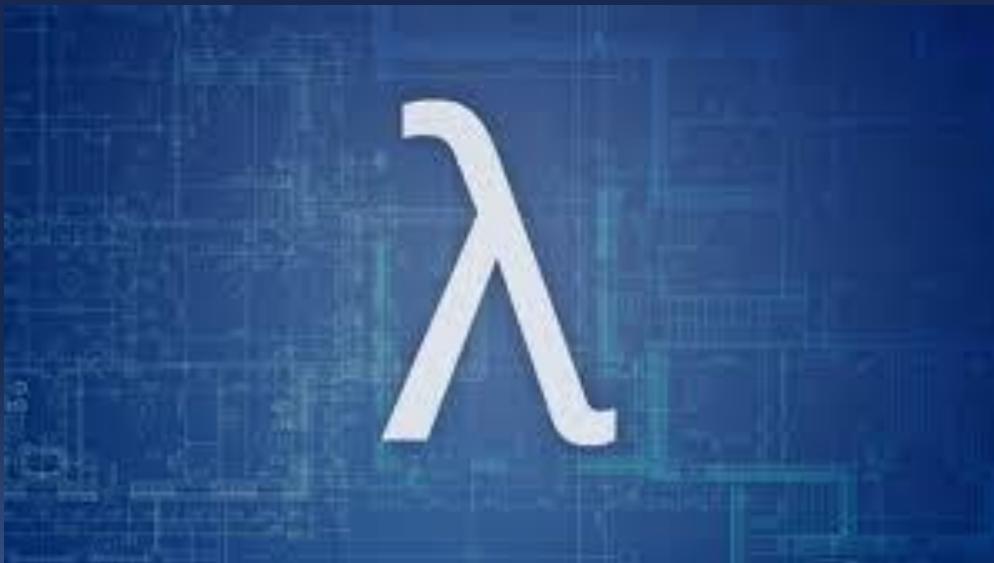


Outline

- ✓ 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**

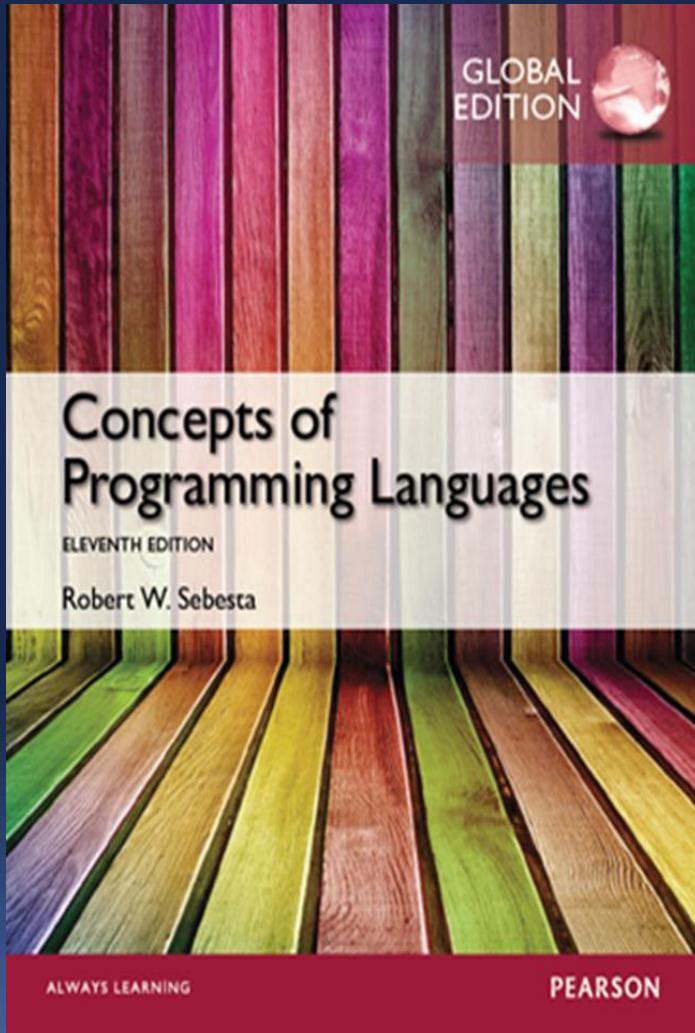


Clojure é uma Linguagem Puramente Funcional



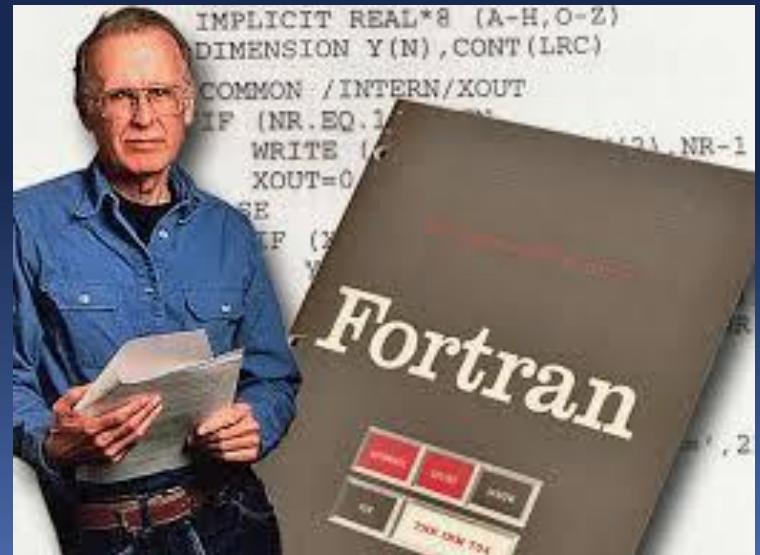
Recomendo ...

■ R. Sebesta - Concepts Of Programming Languages



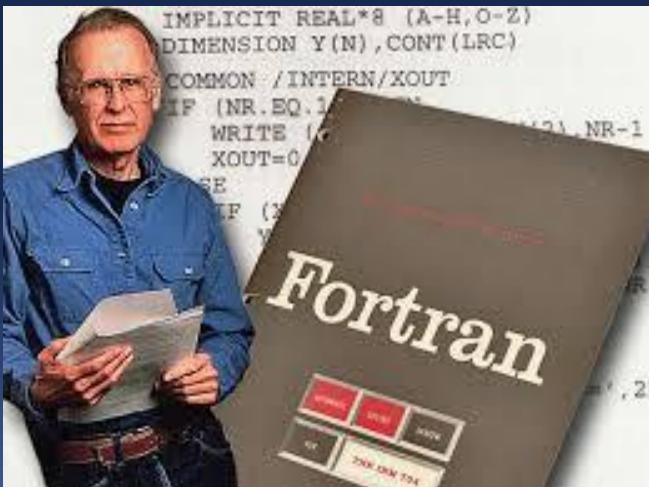
O Paradigma Funcional

- ✓ 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 ?



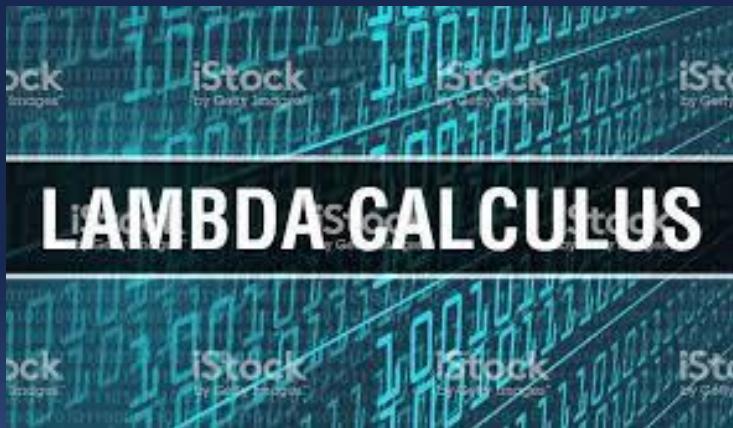
Linguagens Funcionais

- ✓ 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;



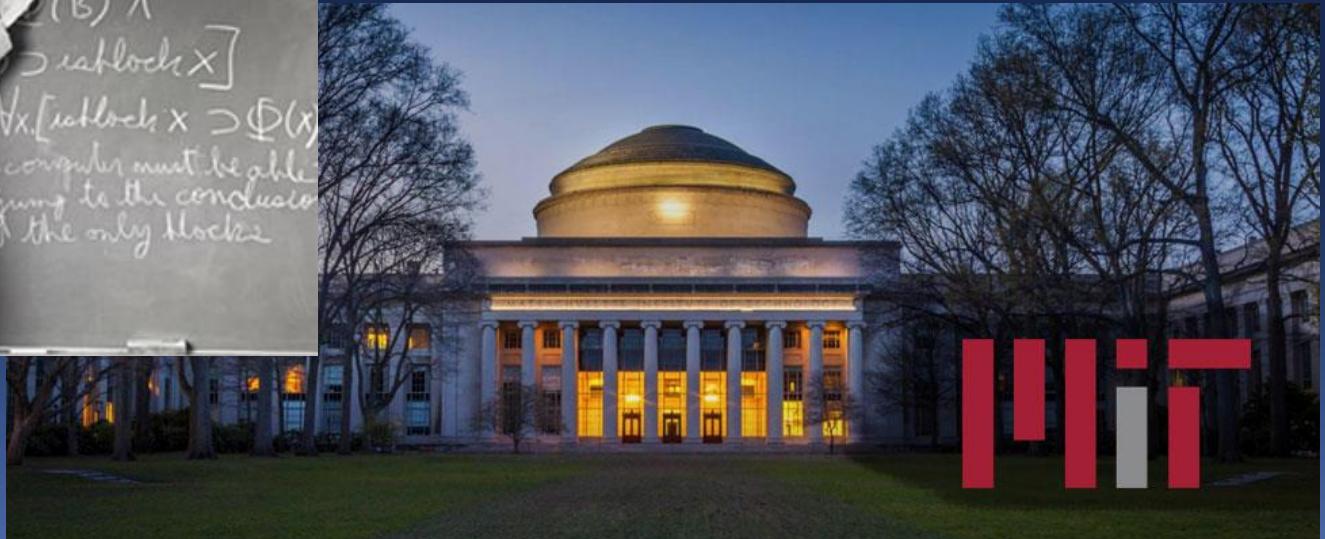
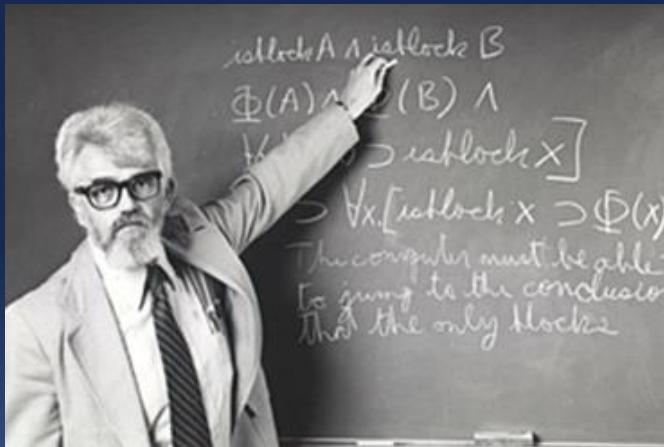


Clojure é um dialeto LISP



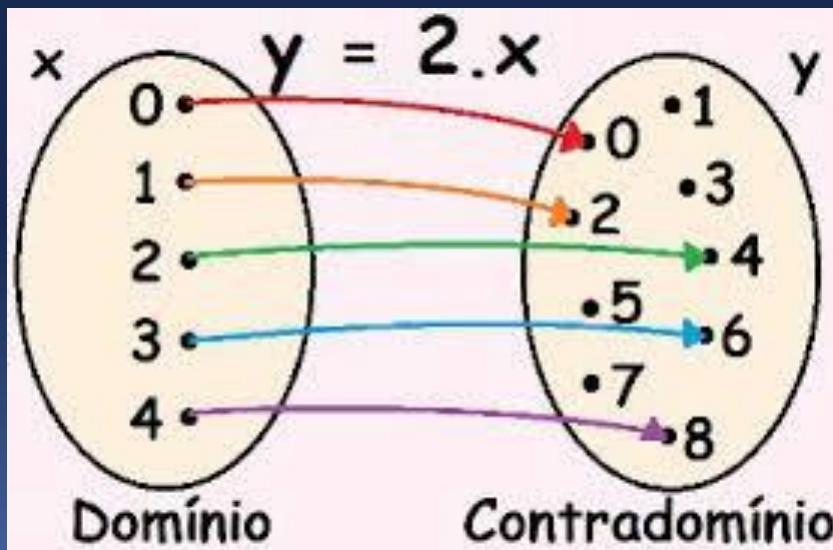
Linguagem de Programação Lisp

- ✓ 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.



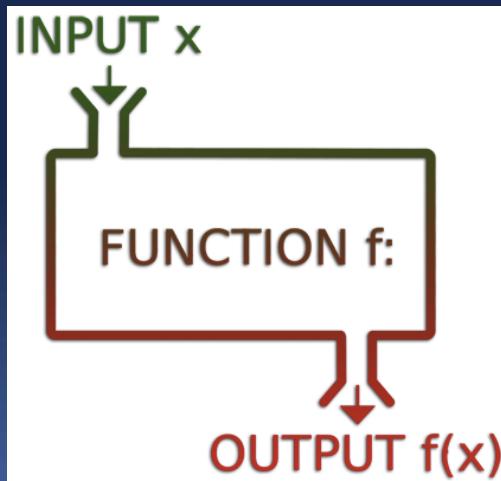
Funções Matemáticas

- ✓ 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**.

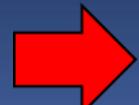
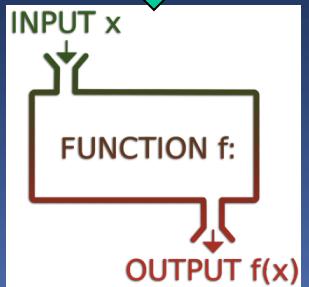
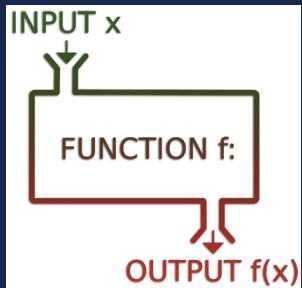
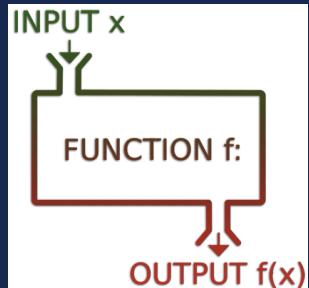


Paradigma Funcional

- ✓ 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**.



Entrada



Saída

✓ 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



```
REPL
user=>
user=> (defn funcao [x] (def x 99) (* x 2) (println x) )
#'user/funcao
user=>
user=>
user=>
user=>
user=>
user=>
user=>
```

A red arrow points to the opening bracket '[x]' in the code.

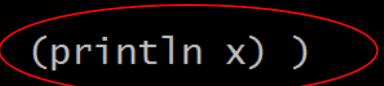
- ✓ O parâmetro **x** é será ligado (**bound**) à algum argumento durante a avaliação;
- ✓ Esse argumento passado à função é **imutável!**



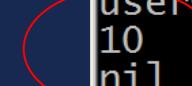


Exemplo - Clojure

```
REPL
user=>
user=> (defn  funcao [x]  (def x 99)  (* x 2) (println x) )
# 'user/funcao
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
```



```
REPL
user=>
user=>
user=>
user=> (funcao 10)
10
nil
user=>
user=>
user=> x
99
user=>
user=>
```



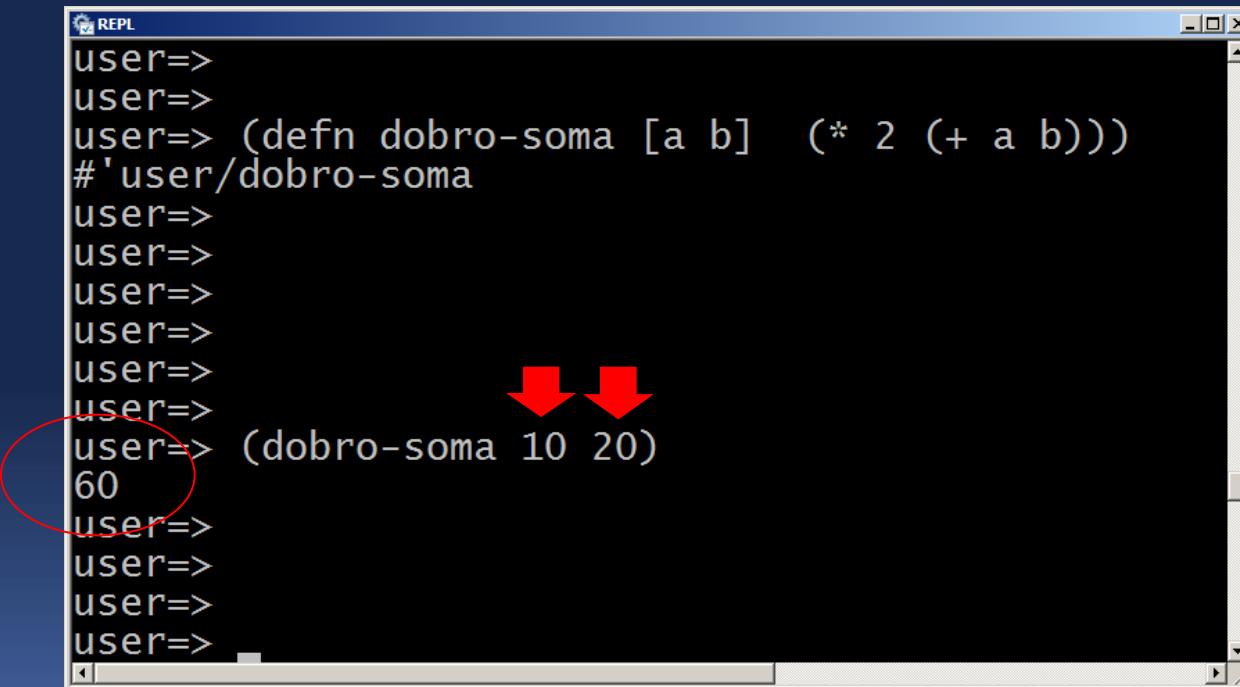
- ✓ O argumento **x** com o valor **10** é constante durante a avaliação.
- ✓ Embora tenha havido um binding anterior para **99**, a função **println** imprimiu, **por side effect**, o valor **10** (constante durante a avaliação).



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**.



The screenshot shows a Clojure REPL window with the following interaction:

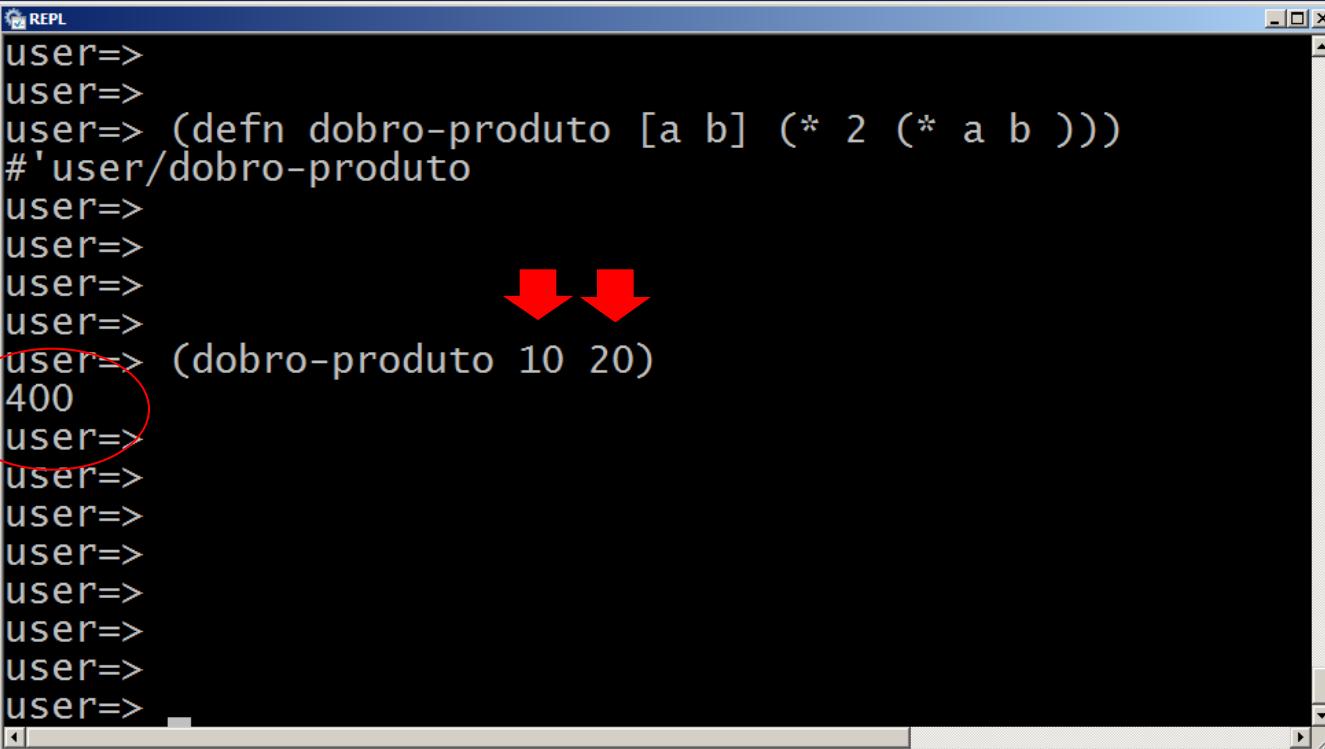
```
REPL
user=>
user=>
user=> (defn dobro-soma [a b] (* 2 (+ a b)))
#'user/dobro-soma
user=>
user=>
user=>
user=>
user=>
user=>
user=> (dobro-soma 10 20)
60
user=>
user=>
user=>
user=>
```

A red oval highlights the line '(dobro-soma 10 20)'. Two red arrows point from the bottom of the oval to the arguments '10' and '20' in the line. The number '60' is also circled in red.

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**



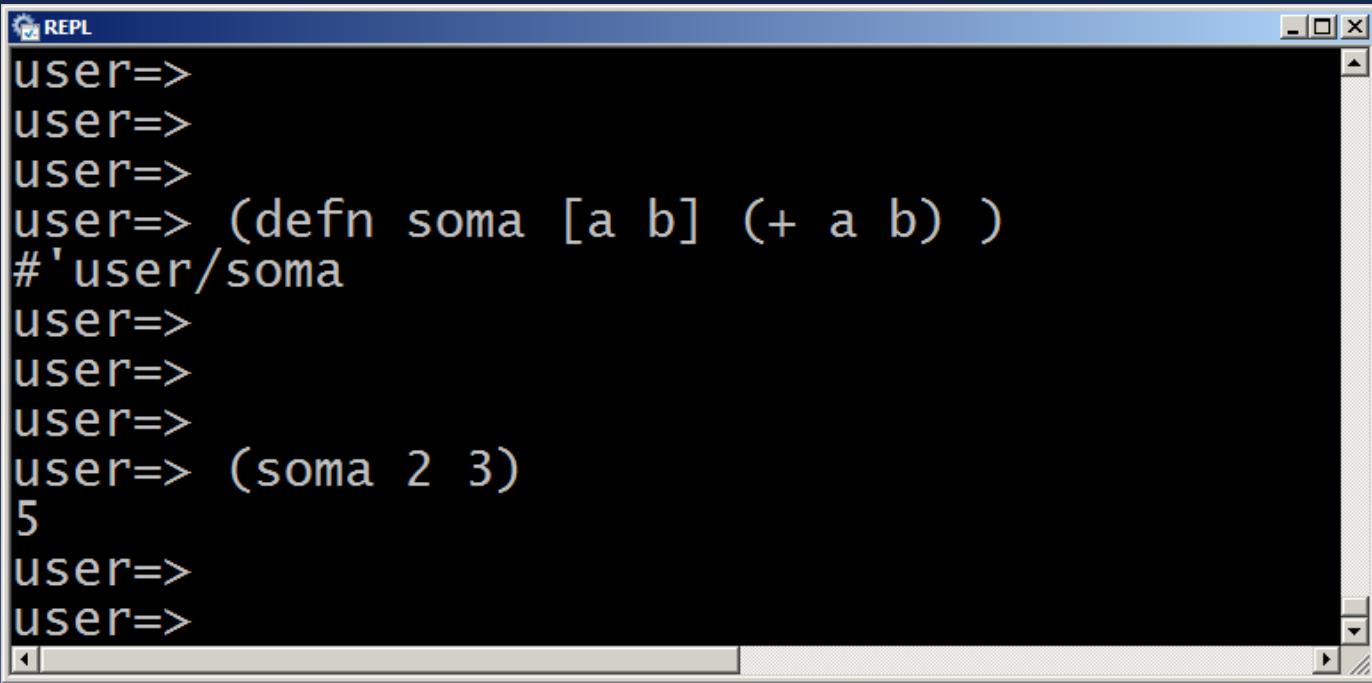
The screenshot shows a Clojure REPL window with the following interaction:

```
REPL
user=>
user=>
user=> (defn dobro-produto [a b] (* 2 (* a b )))
#'user/dobro-produto
user=>
user=>
user=>
user=>
user=> (dobro-produto 10 20)
400
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
```

A red circle highlights the number 400, and two red arrows point from the bottom of the screen towards the arguments 10 and 20 in the call to the function.

Higher Order Function - Clojure

- ✓ As funções **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**.

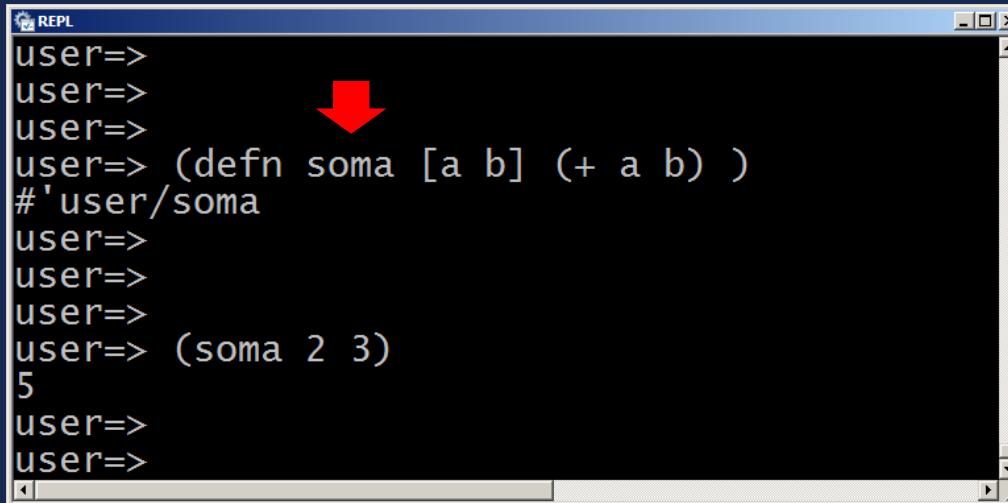
A screenshot of a Java-based REPL (Read-Eval-Print Loop) window titled "REPL". The window has a dark background and displays the following Clojure code:

```
user=>
user=>
user=>
user=> (defn soma [a b] (+ a b) )
#'user/soma
user=>
user=>
user=>
user=> (soma 2 3)
5
user=>
user=>
```

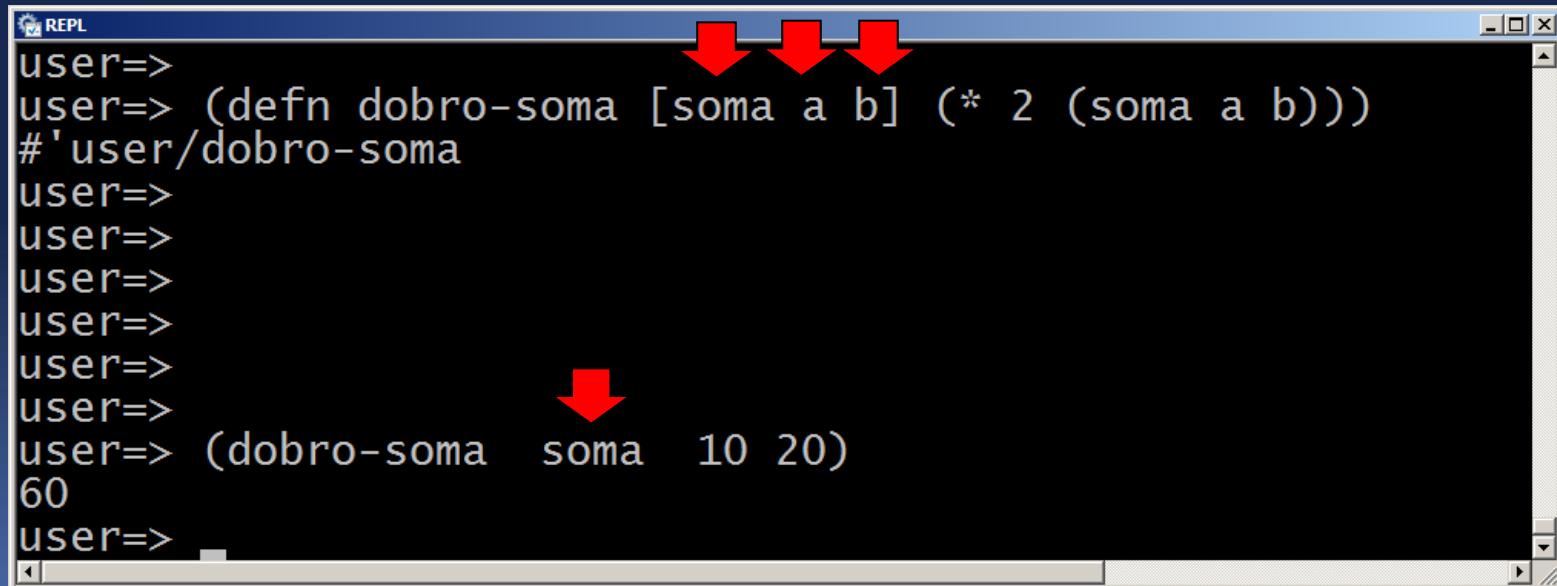
The code defines a function "soma" that takes two arguments and adds them together. It then calls "soma" with arguments 2 and 3, resulting in the output "5".

Higher Order Function - Clojure

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



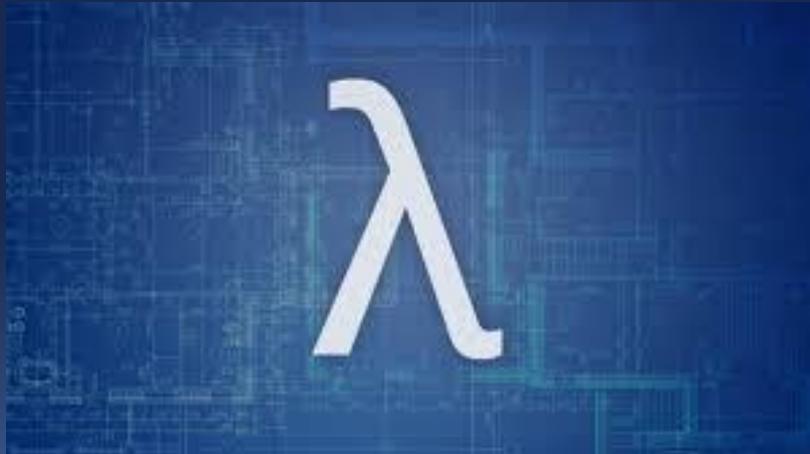
```
REPL
user=>
user=>
user=>
user=> (defn soma [a b] (+ a b) )
#'user/soma
user=>
user=>
user=>
user=> (soma 2 3)
5
user=>
user=>
```



```
REPL
user=>
user=> (defn dobro-soma [soma a b] (* 2 (soma a b)))
#'user/dobro-soma
user=>
user=>
user=>
user=>
user=>
user=> (dobro-soma soma 10 20)
60
user=>
```

Objetivos da Programação Funcional

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



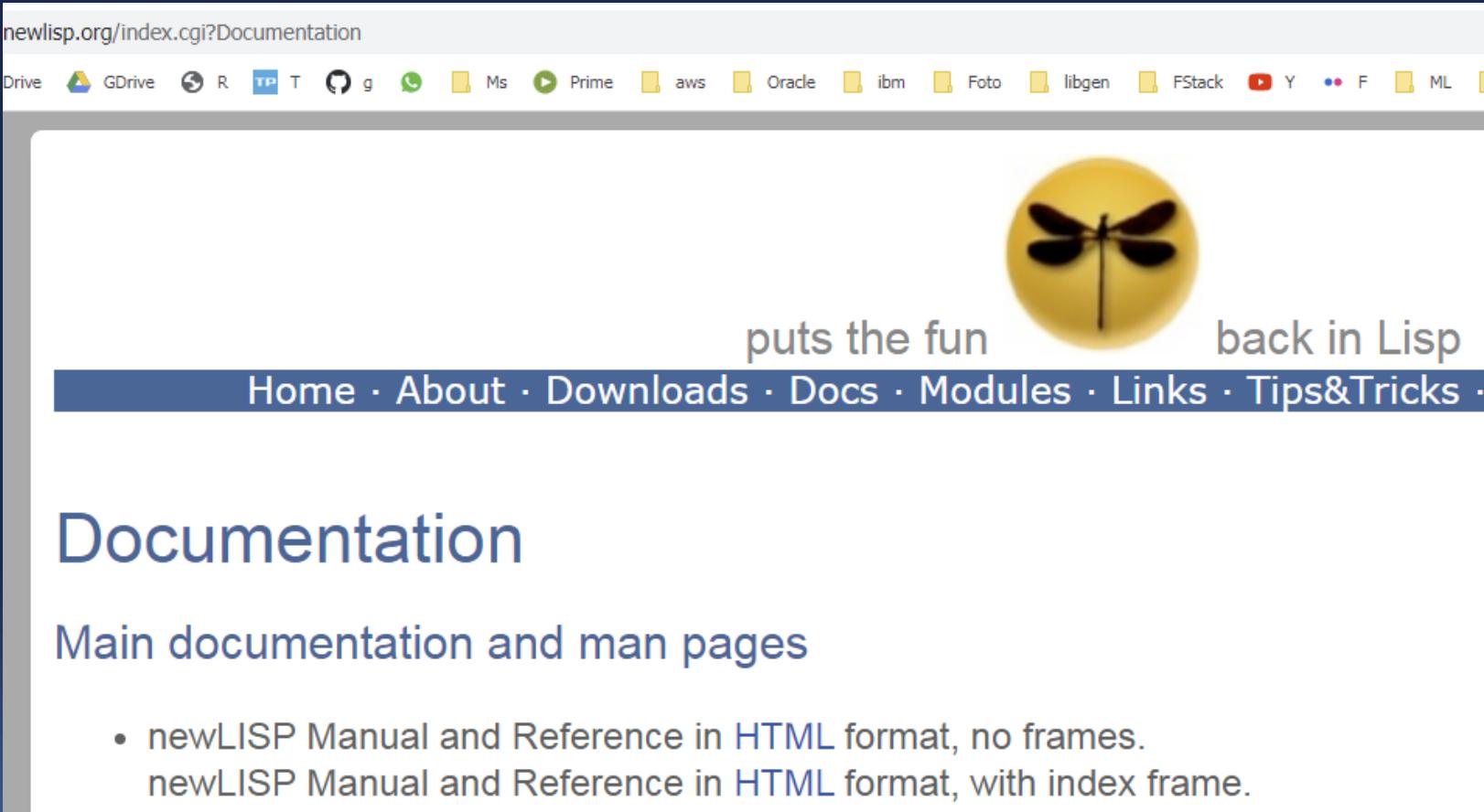
A Linguagem Lisp

- ✓ 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>

A screenshot of a web browser showing the newLISP documentation homepage. The address bar shows "newlisp.org/index.cgi?Documentation". The top navigation bar includes links for Drive, GDrive, R, TP, T, g, Ms, Prime, aws, Oracle, ibm, Foto, libgen, FStack, Y, F, ML, and a search icon. The main content area features a large yellow circular logo with a black dragonfly in the center. Below the logo, the text "puts the fun back in Lisp" is displayed. A horizontal menu bar contains links for Home, About, Downloads, Docs, Modules, Links, and Tips&Tricks. The page title is "Documentation".

newlisp.org/index.cgi?Documentation

Drive GDrive R TP T g Ms Prime aws Oracle ibm Foto libgen FStack Y F ML

puts the fun back in Lisp

Home · About · Downloads · Docs · Modules · Links · Tips&Tricks ·

Documentation

Main documentation and man pages

- newLISP Manual and Reference in [HTML](#) format, no frames.
newLISP Manual and Reference in [HTML](#) format, with index frame.

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++**.

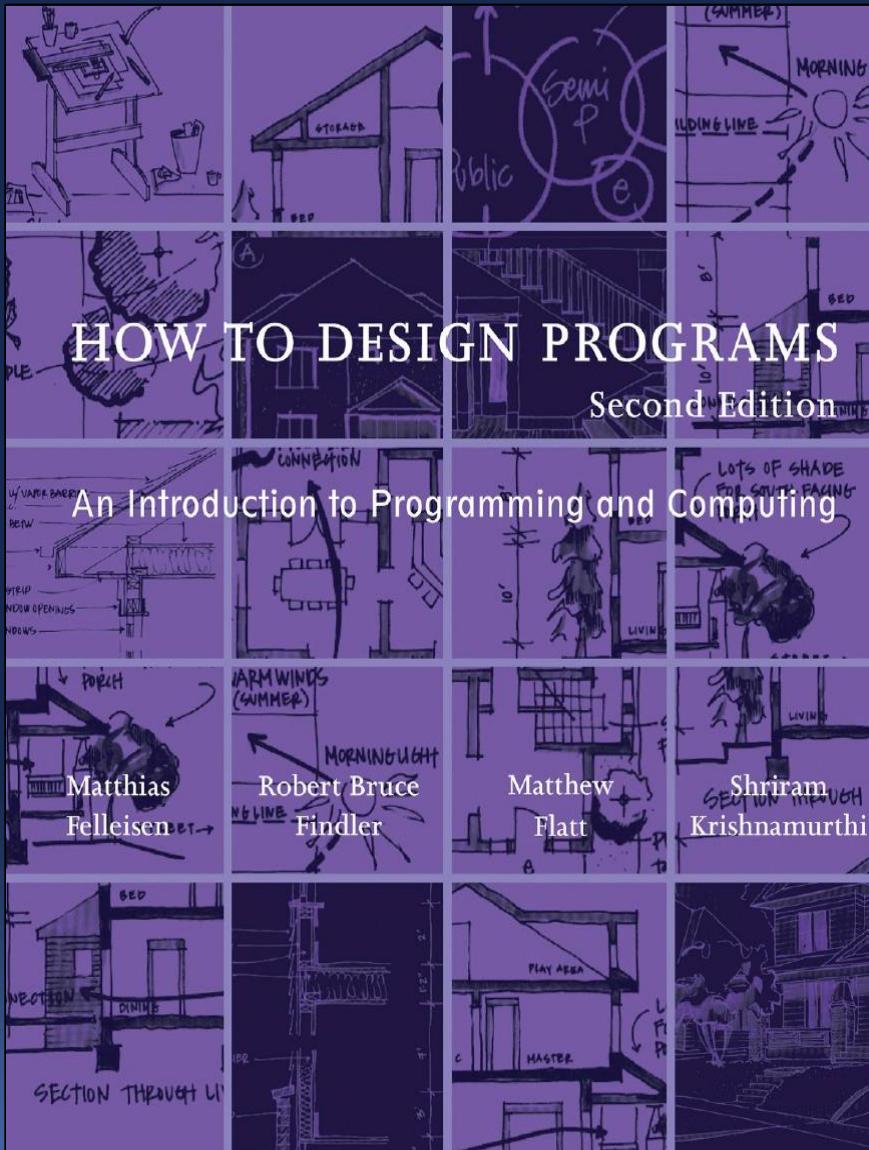


Linguagem Racket

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

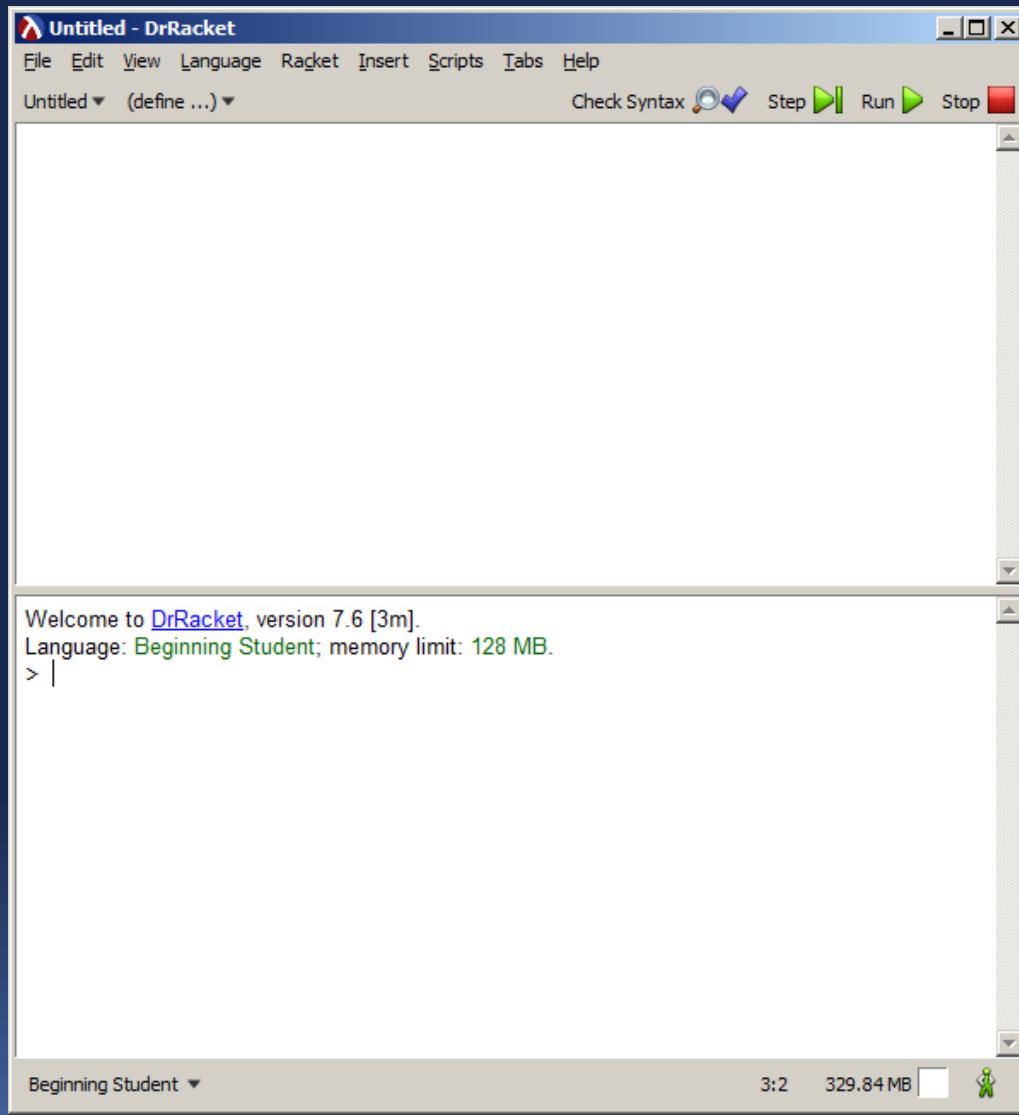


Racket





Racket

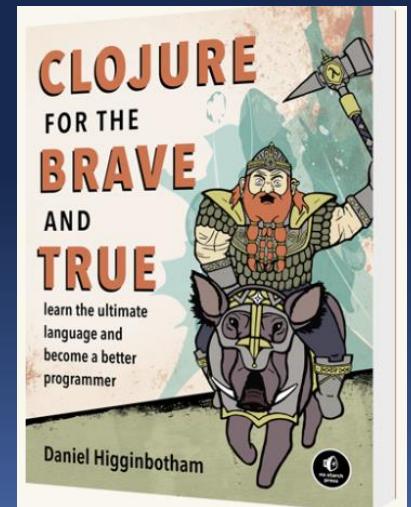
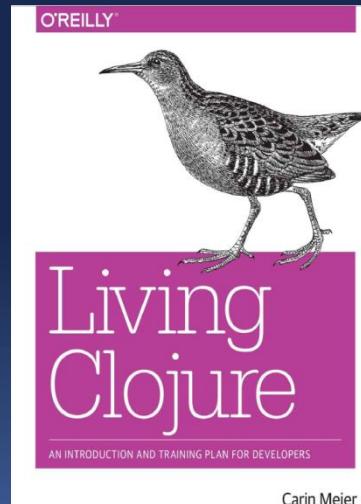
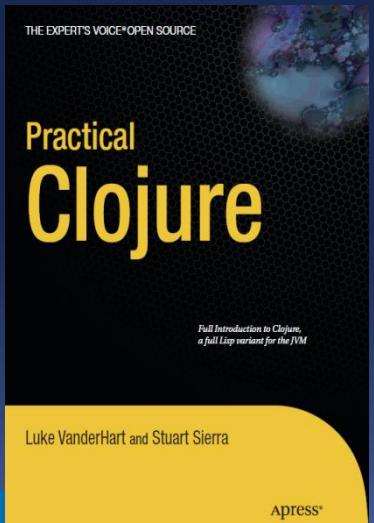
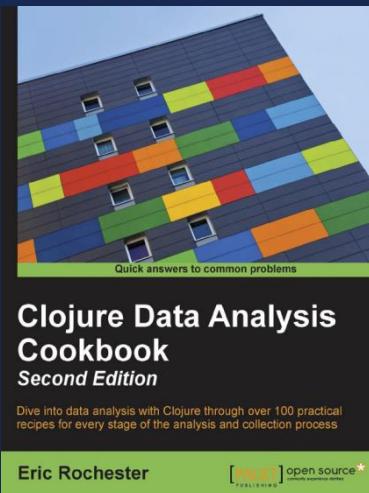
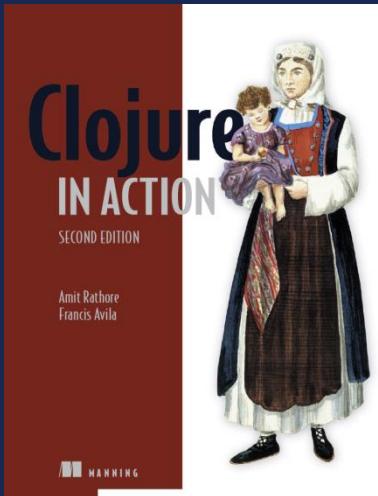
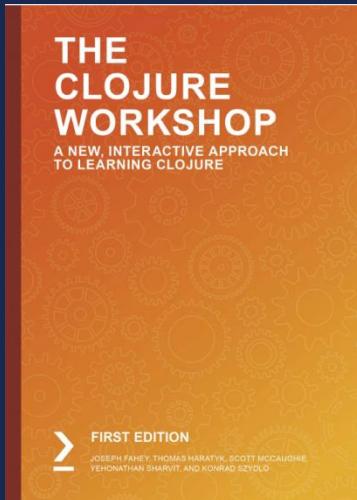


Linguagem Clojure

- ✓ 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/>

The Clojure Programming Language

Clojure is a dynamic, general-purpose programming language, combining the approachability and interactive development of a scripting language with an efficient and robust infrastructure for multithreaded programming. Clojure is a compiled language, yet remains completely dynamic – every feature supported by Clojure is supported at runtime. Clojure provides easy access to the Java frameworks, with optional type hints and type inference, to ensure that calls to Java can avoid reflection.

Clojure is a dialect of Lisp, and shares with Lisp the code-as-data philosophy and a powerful macro system. Clojure is predominantly a functional programming language, and features a rich set of immutable, persistent data structures. When mutable state is needed, Clojure offers a software transactional memory system and reactive Agent system that ensure clean, correct, multithreaded designs.

I hope you find Clojure's combination of facilities elegant, powerful, practical and fun to use.

Rich Hickey
author of Clojure and CTO Cognitect

Companies Succeeding with Clojure

"Our Clojure system just handled its first Walmart black Friday and came out without

"Clojure is a functional programming language from top to bottom. This means

"We discussed the existing Clojure community: the maturity of the language

Learn More

Rationale
A brief overview of Clojure and the features it includes

Getting Started
Resources for getting Clojure up and running

Reference
Grand tour of all that Clojure has to offer

Guides
Walkthroughs to help you learn along the way

Community
We have a vibrant, flourishing community. Join us!

Clojure TV

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:\closure

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window has a black background and white text. The text shows the command "E:\>cd closure" being entered, followed by a new line and the prompt "E:\closure>".

```
E:\>cd closure
E:\closure>
```





Baixando clojure

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text:

```
E:\>cd clojure  
E:\Clojure>git clone https://github.com/clojure/clojure.git
```

The window has a standard blue title bar and a black background for the command area. The text is white and clearly legible.



Baixando clojure

```
C:\ Command Prompt
E:\>cd clojure

E:\clojure>git clone https://github.com/clojure/clojure.git
Cloning into 'clojure'...
remote: Enumerating objects: 41, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 32714 (delta 11), reused 29 (delta 9), pack-reused 3267
Receiving objects: 100% (32714/32714), 14.92 MiB | 7.40 MiB/s, done.
Resolving deltas: 100% (19735/19735), done.

E:\clojure>_
```



Pasta Clojure

Name	Date modified	Type	Size
.git	20-May-20 2:00 PM	File folder	
.idea	20-May-20 2:00 PM	File folder	
doc	20-May-20 2:00 PM	File folder	
src	20-May-20 2:00 PM	File folder	
test	20-May-20 2:00 PM	File folder	
	20-May-20 2:00 PM	Text Document	1 KB
antsetup	20-May-20 2:00 PM	Shell Script	1 KB
build	20-May-20 2:00 PM	XML Document	9 KB
changes	20-May-20 2:00 PM	Markdown Source File	115 KB
clojure.iml	20-May-20 2:00 PM	IML File	2 KB
CONTRIBUTING	20-May-20 2:00 PM	Markdown Source File	1 KB
epl-v10	20-May-20 2:00 PM	Chrome HTML Docu...	13 KB
pom	20-May-20 2:00 PM	XML Document	11 KB
readme	20-May-20 2:00 PM	Text Document	14 KB

Building Clojure com Maven

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

```
Command Prompt
Downloaded from central: https://repo.maven.apache.org/maven2/org/vafer/jdependency/1.2/jdependency-1.2.jar (22 kB at 20 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/19.0/guava-19.0.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-analysis/6.0_BETA/asm-analysis-6.0_BETA.jar (21 kB at 16 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/jdom/jdom/1.1.3/jdom-1.1.3.jar (151 kB at 117 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/eclipse/aether/aether-util/0.9.0.M2/aether-util-0.9.0.M2.jar (134 kB at 100 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-util/6.0_BETA/asm-util-6.0_BETA.jar (47 kB at 35 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/com/google/guava/guava/19.0/guava-19.0.jar (2.3 MB at 391 kB/s)
[INFO] Including org.clojure:spec.alpha:jar:0.2.187 in the shaded jar.
[INFO] Including org.clojure:core.specs.alpha:jar:0.2.44 in the shaded jar.
[INFO] Including org.clojure:test.check:jar:0.9.0 in the shaded jar.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:28 min
[INFO] Finished at: 2020-05-20T14:12:28-03:00
[INFO] -----
```

E:\clojure>



Iniciando REPL

java -jar clojure.jar

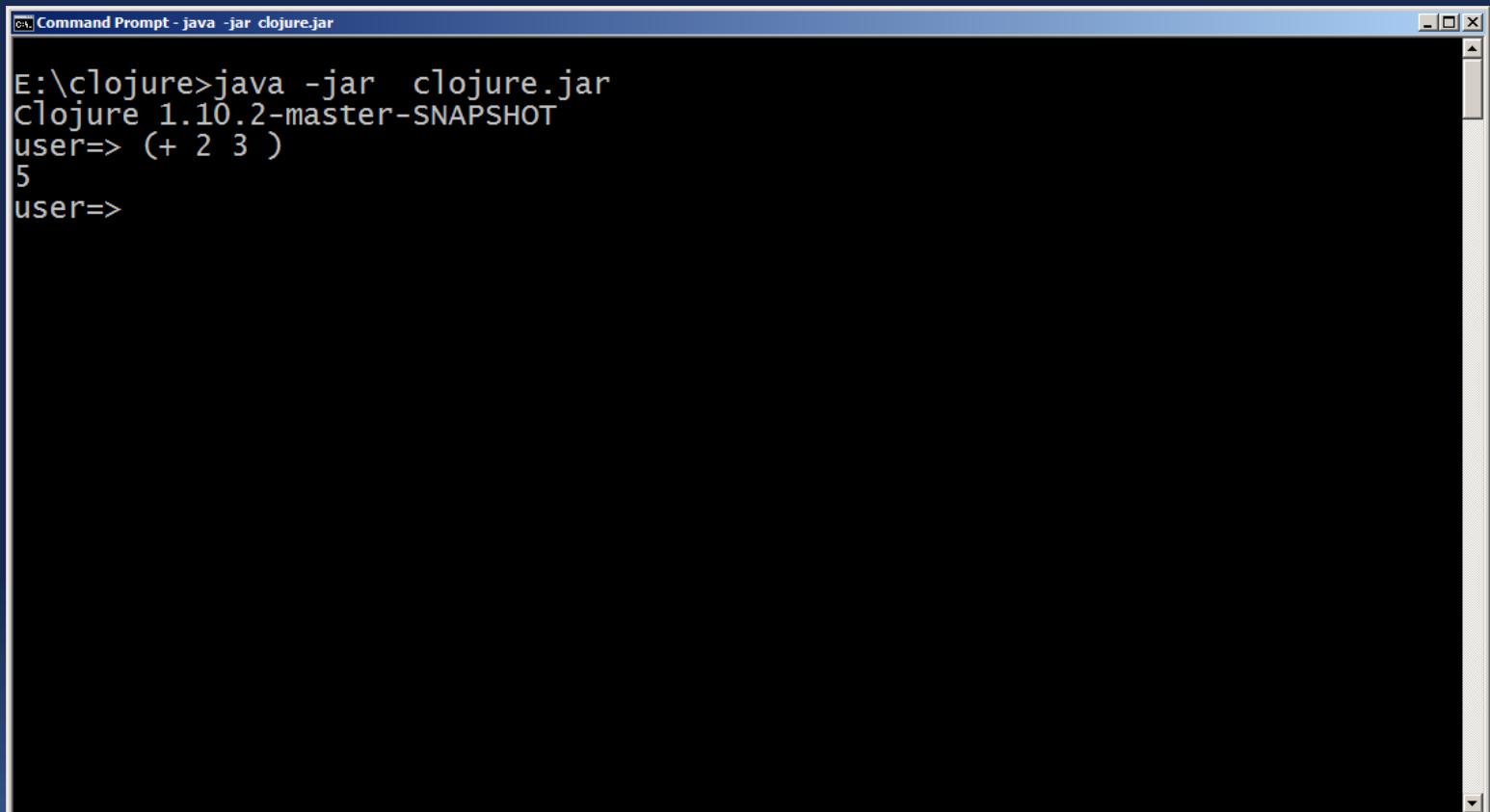
A screenshot of a Windows Command Prompt window titled "Command Prompt - java -jar clojure.jar". The window shows the command "E:\clojure>java -jar clojure.jar" being run, followed by the Clojure version "Clojure 1.10.2-master-SNAPSHOT" and the prompt "user=>". The window has a standard Windows title bar and scroll bars on the right side.

```
E:\clojure>java -jar clojure.jar
Clojure 1.10.2-master-SNAPSHOT
user=>
```



Testando REPL

java -jar clojure.jar

A screenshot of a Windows Command Prompt window titled "Command Prompt - java -jar clojure.jar". The window shows the following text:

```
E:\clojure>java -jar clojure.jar
Clojure 1.10.2-master-SNAPSHOT
user=> (+ 2 3 )
5
user=>
```

The window has a standard Windows title bar and scroll bars on the right side.



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

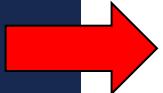
Make sure [PowerShell 5](#) (or later, include PowerShell Core) and [.NET Core SDK 2.1+](#) or [.NET Framework 4.5+](#) (or later) are installed. Then run:

```
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
```

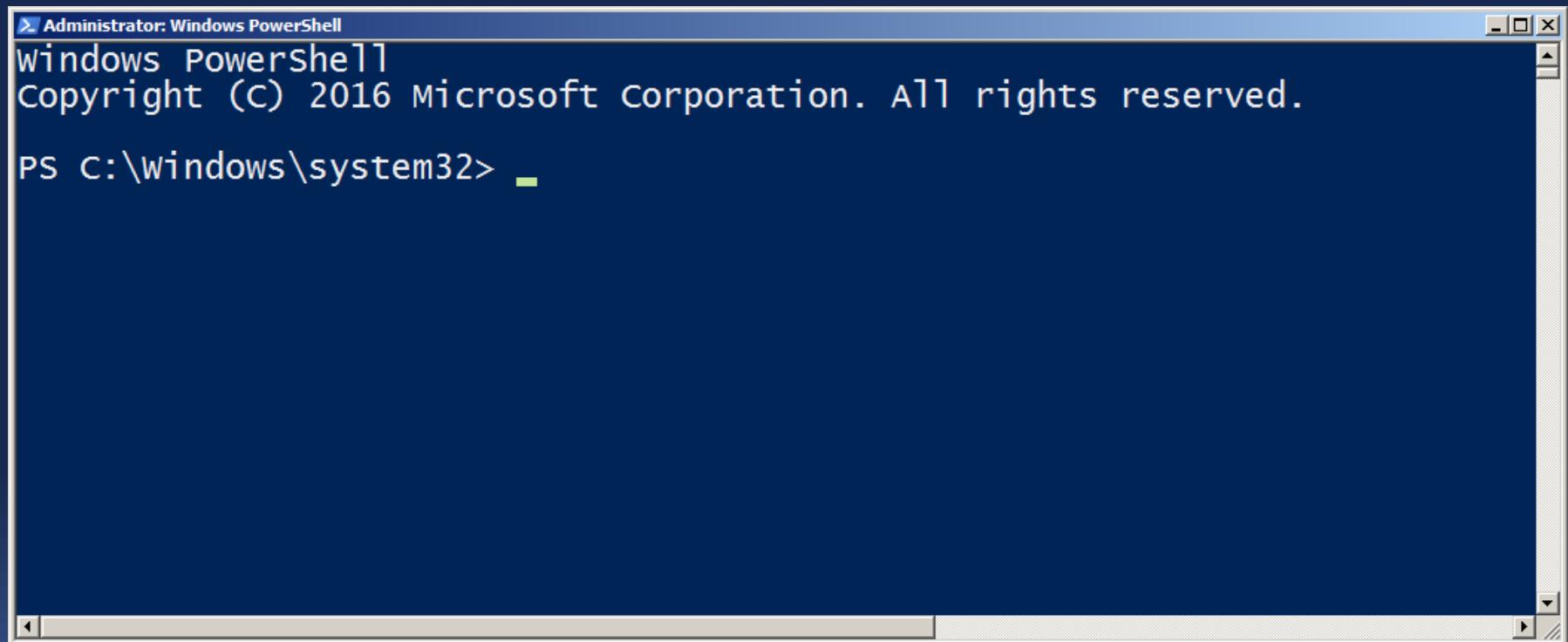
```
Cleaning up existing install
Installing PowerShell module
Removing download
Clojure now installed. Use "clj -h" for help.
```

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;



A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The title bar also includes the text "Windows PowerShell" and "Copyright (c) 2016 Microsoft Corporation. All rights reserved.". The command prompt shows "PS C:\windows\system32> -". The window has a standard Windows look with a blue header bar and a dark blue background.

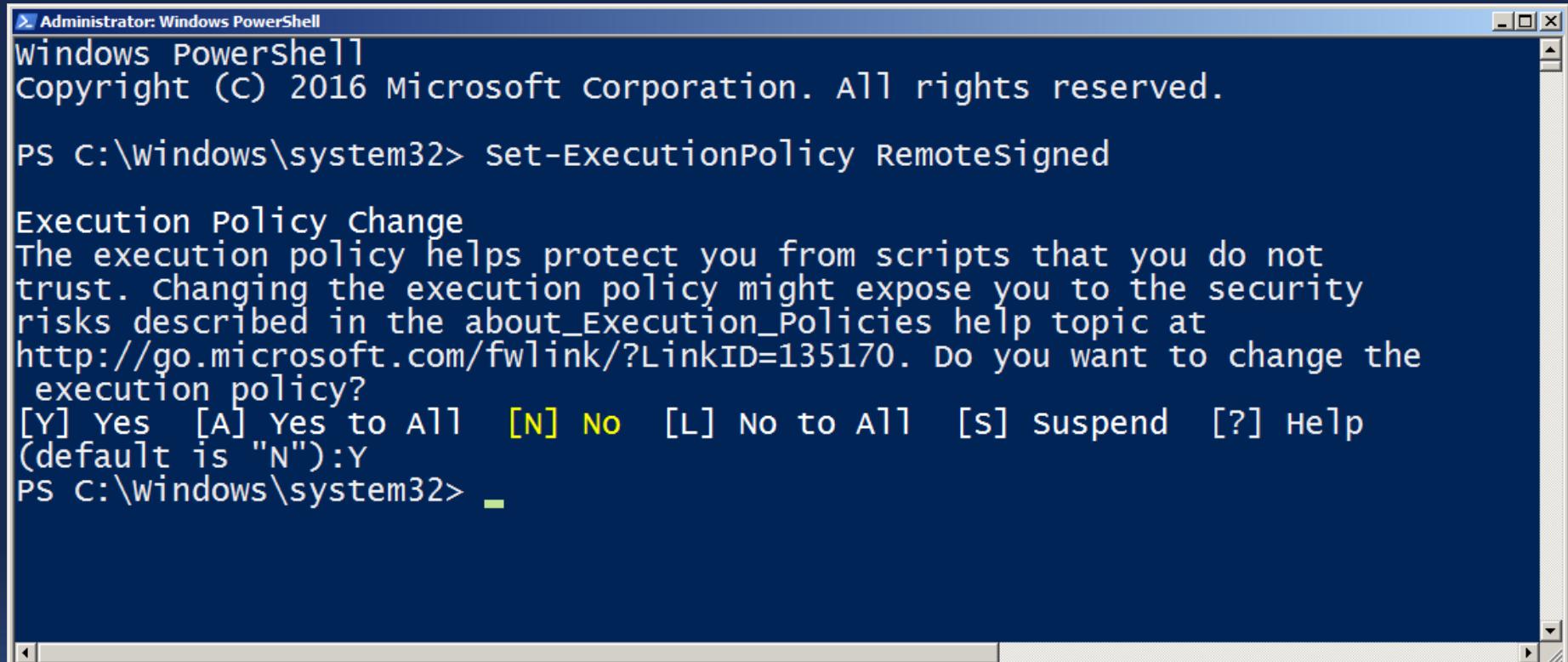


Configurações de segurança - Powershell



Habilitar PowerShell para execução

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



A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The window shows the command "Set-ExecutionPolicy RemoteSigned" being run, followed by a confirmation message about changing the execution policy. The user is prompted to choose between [Y] Yes, [A] Yes to All, [N] No, [L] No to All, [S] Suspend, and [?] Help, with the default being "N". The user types "Y" and presses Enter.

```
Administrator: Windows PowerShell
windows Powershell
Copyright (c) 2016 Microsoft Corporation. All rights reserved.

PS C:\windows\system32> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not
trust. Changing the execution policy might expose you to the security
risks described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the
execution policy?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help
(default is "N"):Y
PS C:\windows\system32>
```



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.

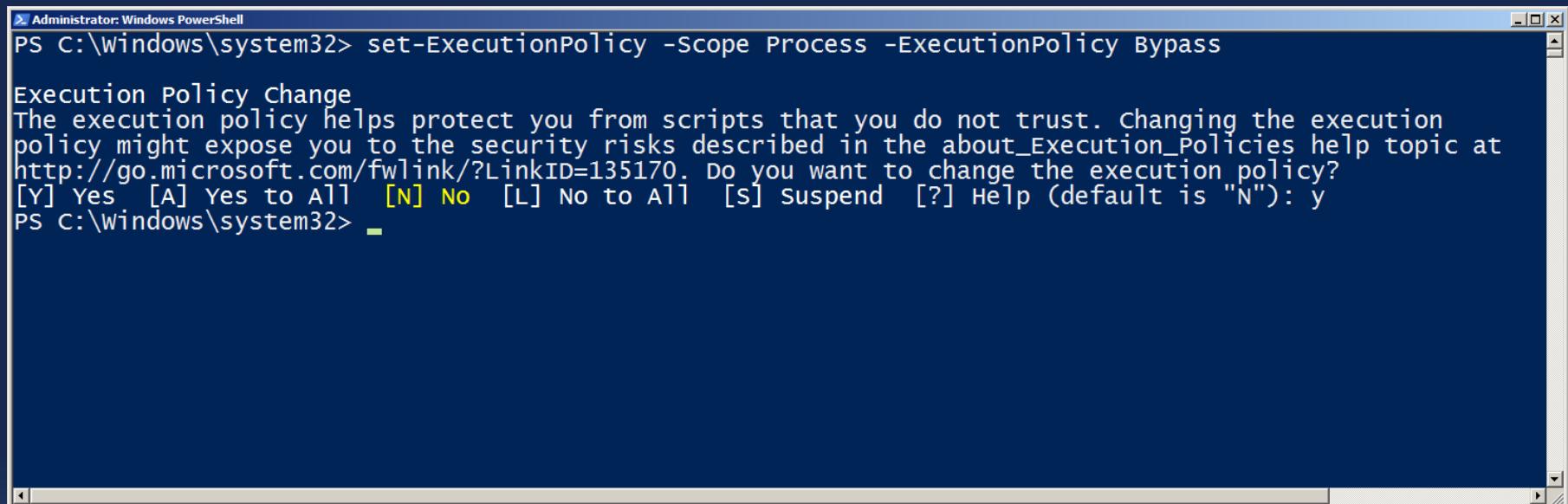
```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

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



```
Administrator: Windows PowerShell
PS C:\Windows\system32> set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution
policy might expose you to the security risks described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkId=135170. Do you want to change the execution policy?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): y
PS C:\Windows\system32> _
```

A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The command `set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass` is entered at the prompt. A confirmation message appears, stating: "Execution Policy Change. The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose you to the security risks described in the about_Execution_Policies help topic at http://go.microsoft.com/fwlink/?LinkId=135170. Do you want to change the execution policy?". The user types "y" to confirm. The window has standard Windows UI elements like title bar, buttons, and scroll bars.



clj on Windows

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

A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The window shows the command "PS C:\Windows\System32> c:\win-install-1.10.1.561.ps1" being typed into the command line. The background of the slide is a dark blue gradient.



clj on Windows

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

A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The command entered is "PS C:\Windows\System32> c:\win-install-1.10.1.561.ps1". The output shows the process of downloading Clojure tools, followed by messages about writing a web request and a request stream. The window has a dark blue background and standard Windows-style borders.

```
Administrator: Windows PowerShell
PS C:\Windows\System32> c:\win-install-1.10.1.561.ps1
Downloading Clojure tools

Writing web request
Writing request stream... (Number of bytes written: 5617888)
```





clj on Windows

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

```
Administrator: Windows PowerShell
PS C:\Windows\System32> c:\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) 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
```



clj on Windows

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

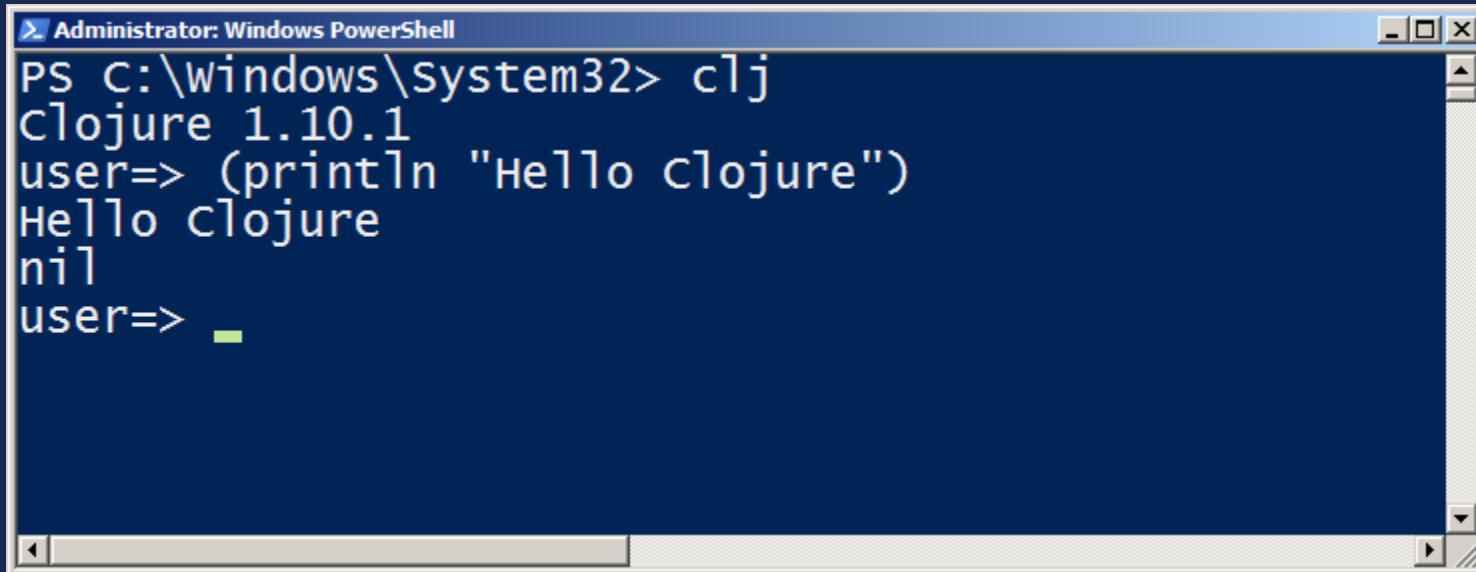
```
Administrator: Windows PowerShell
Downloading clojure tools
WARNING: Clojure will install as a module in your PowerShell module path.

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



A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The command "PS C:\windows\system32> clj" is entered, followed by the Clojure version "clojure 1.10.1". A user interaction "user=>" is shown with the expression "(println "Hello Clojure")" and its output "Hello Clojure". The response "nil" is also displayed. The window has a standard Windows title bar and scroll bars.

```
PS C:\windows\system32> clj
clojure 1.10.1
user=> (println "Hello Clojure")
Hello Clojure
nil
user=> _
```



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**



```
C:\Windows\system32\cmd.exe - powershell -command clj
C:\>powershell -command clj
clojure 1.10.1
user=>
```

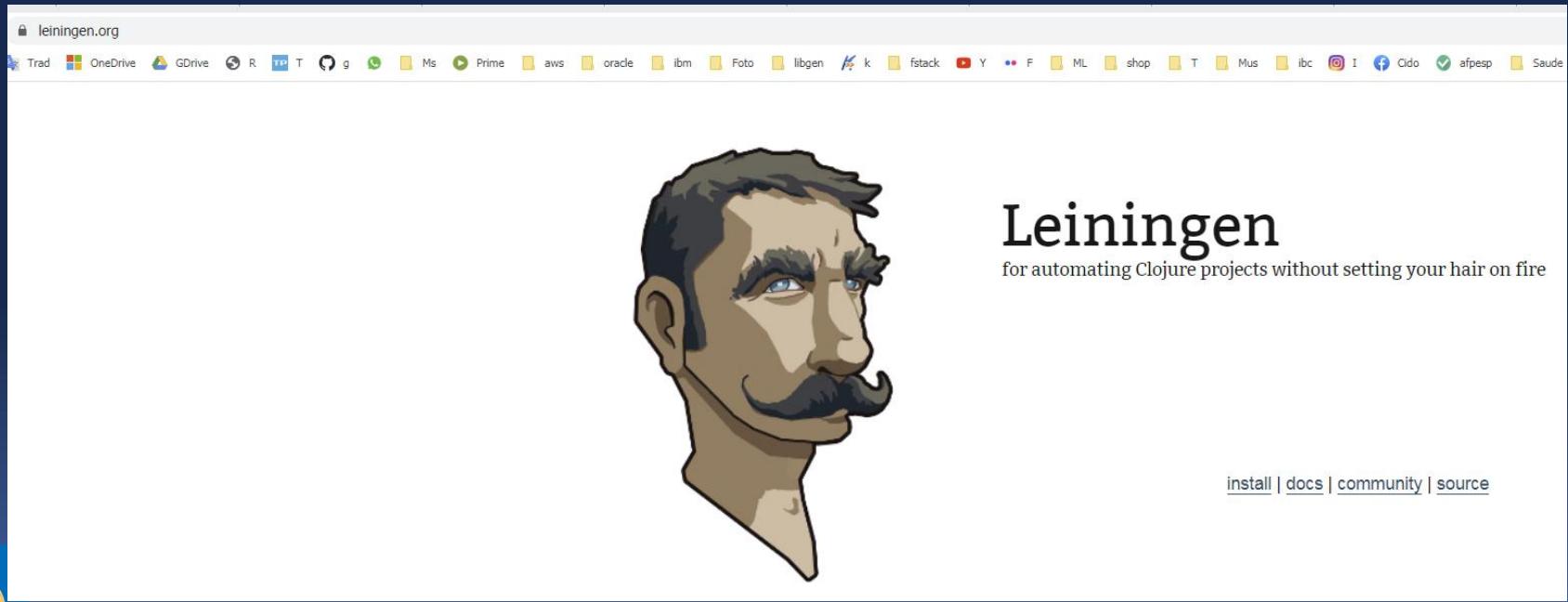


Instalação com Leiningen



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

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

Install

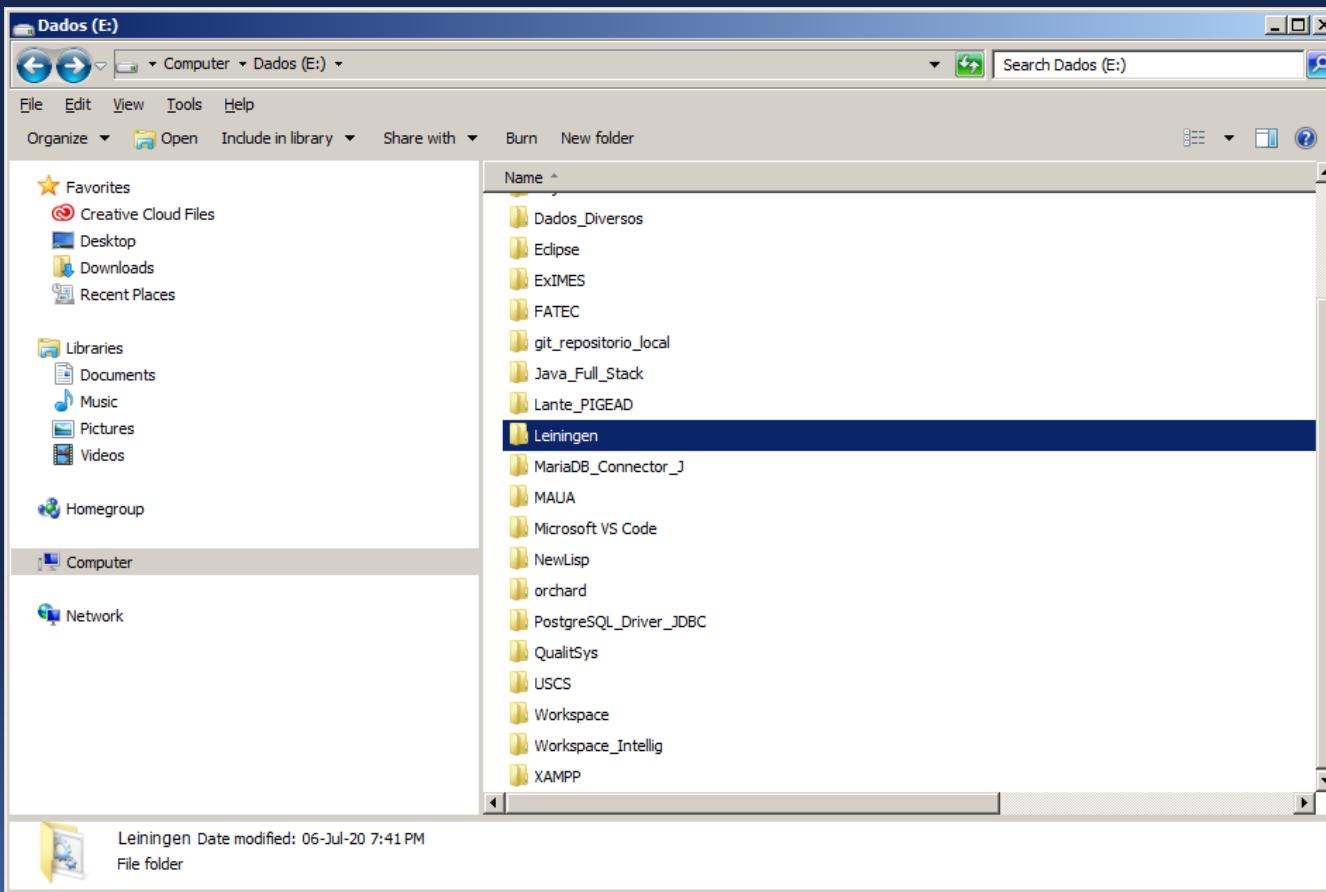
1. Download the [lein script](#) (or on Windows [lein.bat](#))
2. Place it on your \$PATH where your shell can find it (eg. ~/bin)
3. Set it to be executable (`chmod a+x ~/bin/lein`)
4. Run it (`lein`) and it will download the self-install package

You can check your [package manager](#) as well..



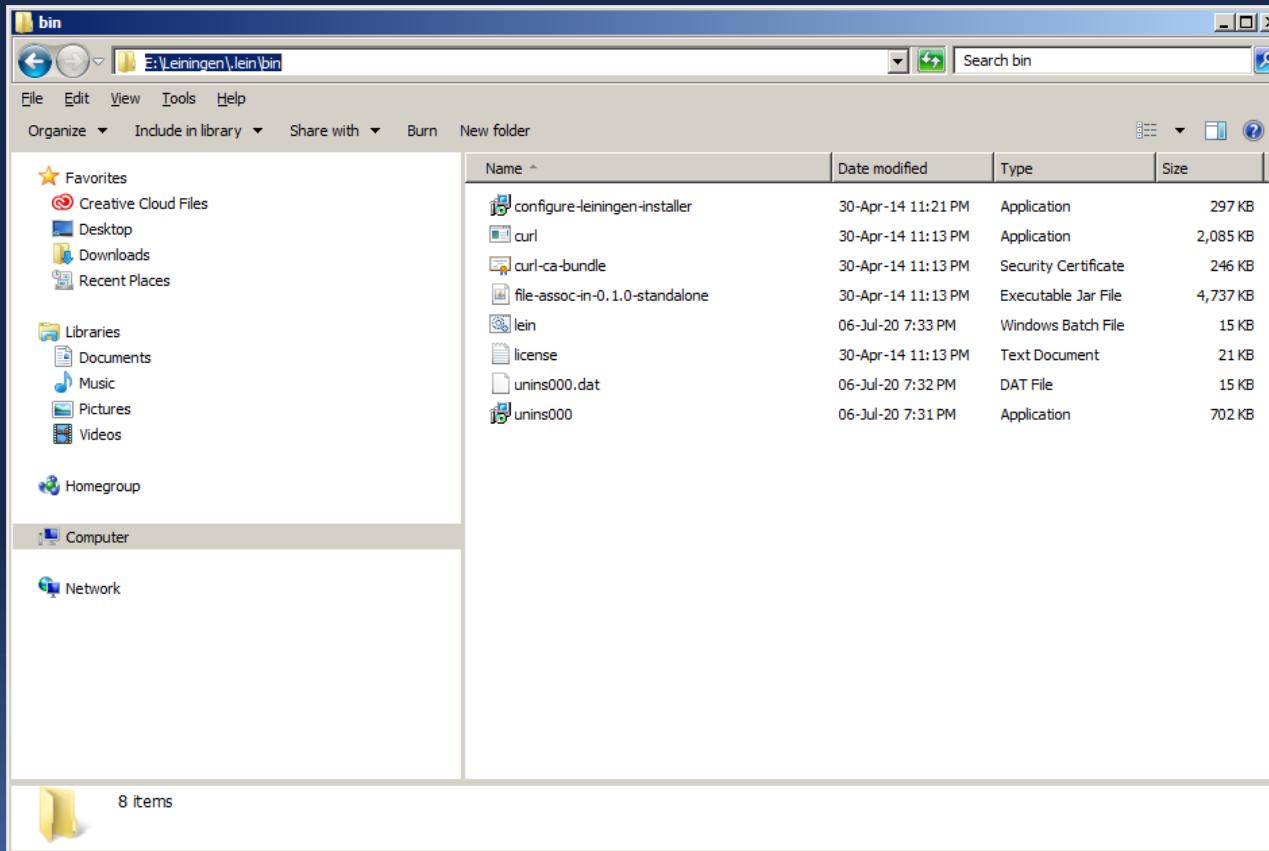
Leiningen - Install

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



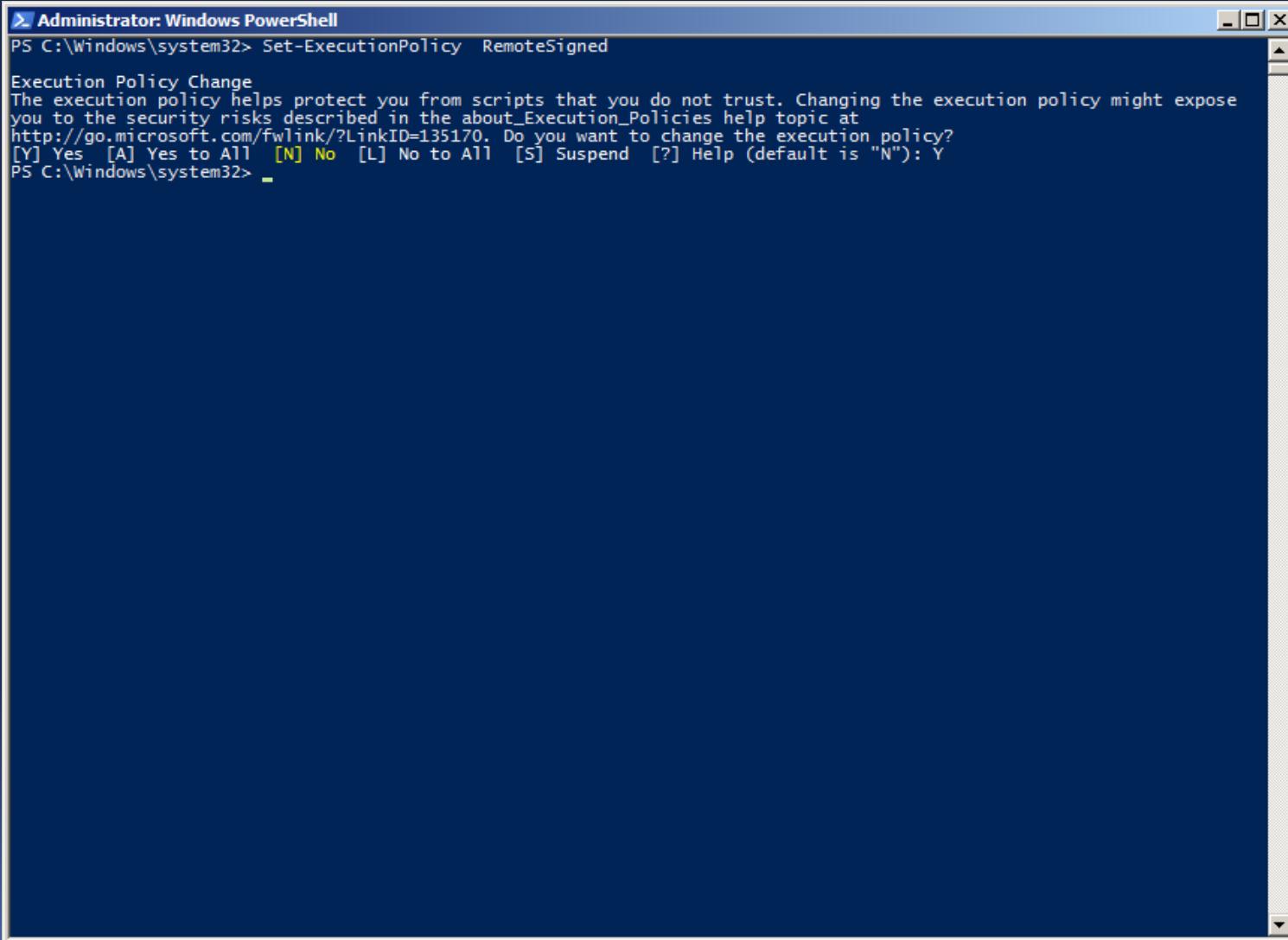
Leiningen - Install

- ✓ 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**



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command entered is "PS C:\Windows\system32> Set-ExecutionPolicy RemoteSigned". A warning message about the execution policy follows, stating: "Execution Policy Change. The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose you to the security risks described in the about_Execution_Policies help topic at http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy? [Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): Y". The user has responded with "Y".



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



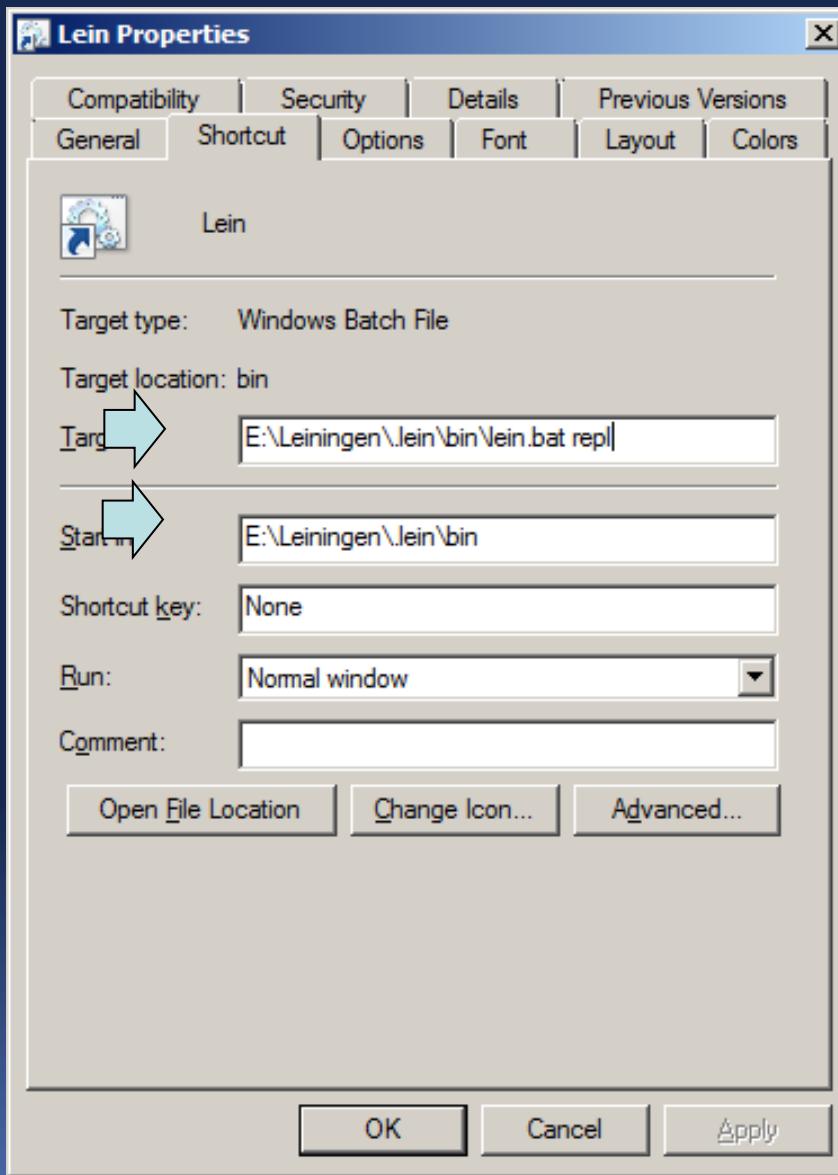
When you run a .ps1 PowerShell script you might get the message saying
“.ps1 is not digitally signed. The script will not execute on the system.”

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

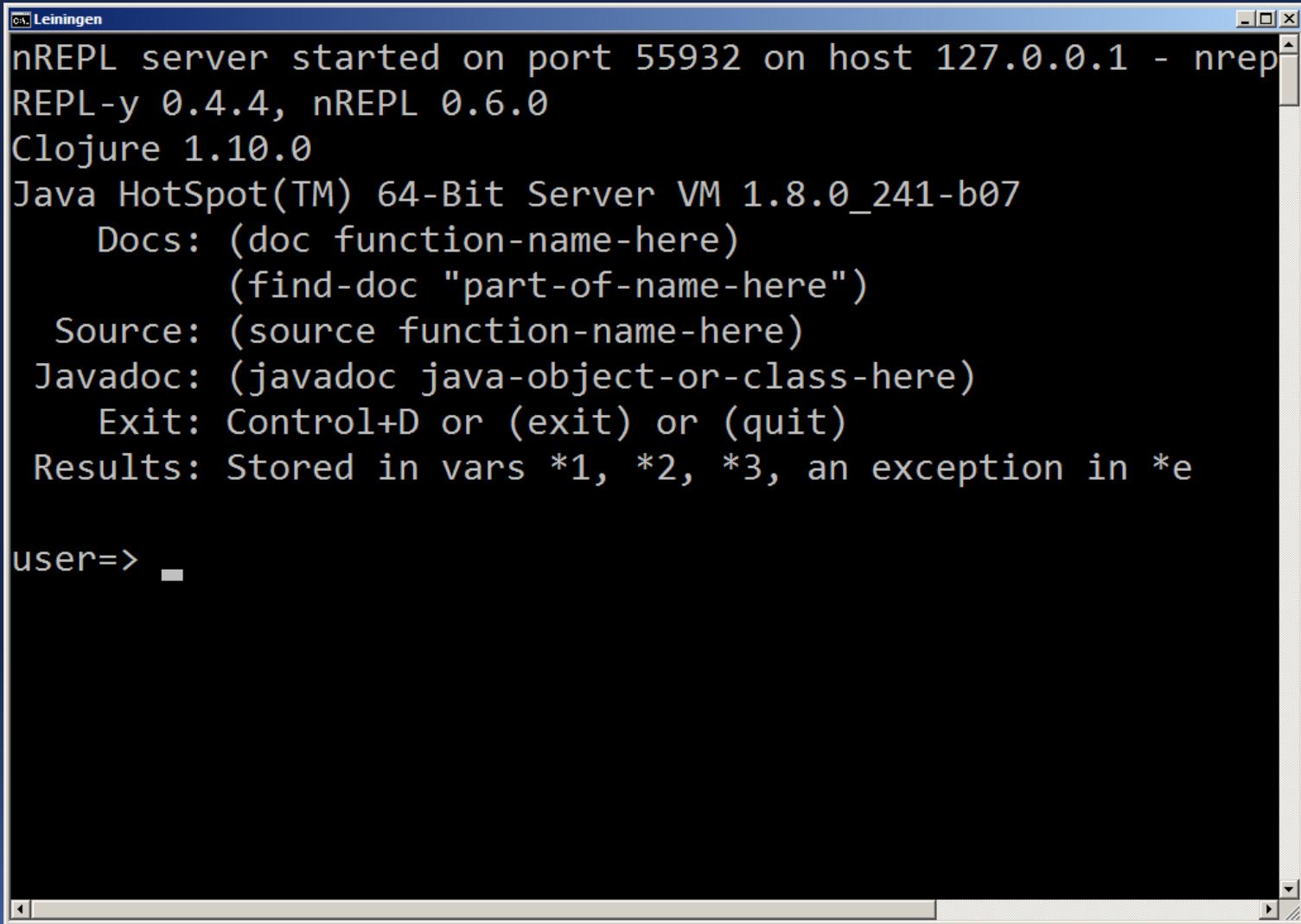
```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

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



A screenshot of a terminal window titled "Leiningen". The window displays the following text:

```
nREPL server started on port 55932 on host 127.0.0.1 - nrepl-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
        (find-doc "part-of-name-here")
  Source: (source function-name-here)
Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
Results: Stored in vars *1, *2, *3, an exception in *e

user=> ■
```

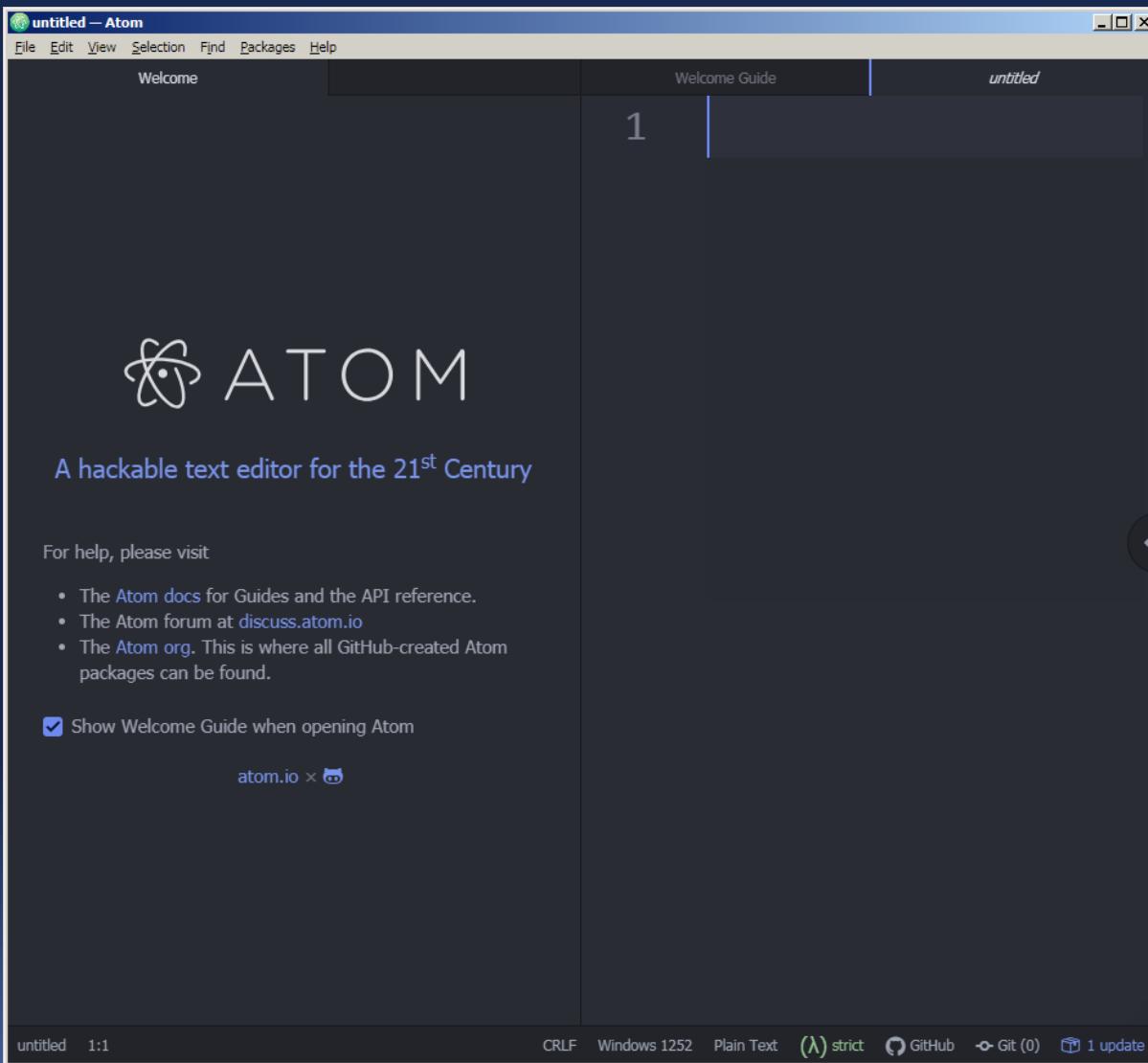


Desenvolvimento Clojure com a IDE Atom





Código Clojure no Atom





Atom – Package Chlorine

✓ **Ctrl + Shift + p** > Install Packages and Themes

A screenshot of the Atom code editor interface. The title bar reads "Welcome Guide — E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_1S\Modelos_Linguagens_Programacao\Fontes_Clojure — Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The main window shows a "Project" sidebar with a "Fontes_Clojure" folder containing "Clojure_1_01.clj" and "Clojure_1_02.clj". A central modal dialog titled "install packages" is open, displaying the "Settings View: Install Packages And Themes" option, which is highlighted with a blue border. Below this are other options: "Settings View: Uninstall Packages" and "Settings View: View Installed Packages". The background of the Atom interface is dark-themed. At the bottom, there are status icons for CRLF, GitHub, Git, and update.





Atom – Package Chlorine

✓ Instalar => package **Chlorine**

Settings — E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_15\Modelos_Linguagens_Programacao\Fontes_Clojure — Atom

File Edit View Selection Find Packages Help

Project Clojure_1_01.clj Clojure_1_02.clj keymap.cson Welcome Guide Welcome Settings

Fontes_Clojure

- Clojure_1_01.clj
- Clojure_1_02.clj

Core Editor URI Handling System Keybindings Packages Themes Updates

+ Install Open Config Folder

+ Install Packages

Packages Themes

chlorine 0.8.1 6,327

Socket REPL client for Clojure and ClojureScript mauricieszabo

Settings Uninstall Disable

tdl-coloring 0.4.2 143

Adds simple support for TDL (Type Description Language) syntax as described in Copestake 2002 lemontheme

Install

exploringly-find-and-replace 0.215.6 63

Replace without moving the cursor. exploringly

Install

colorio 0.2.1 6,631

An Atom package that look for color cheat sheet nju33

Install

doctrine 0.3.4 2,272

Doctrine support for Atom Cronos87

Install

github-authoring 0.1.0 142

A short description of your package

CRLF (A) strict GitHub Git (0) 1 update

Settings

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

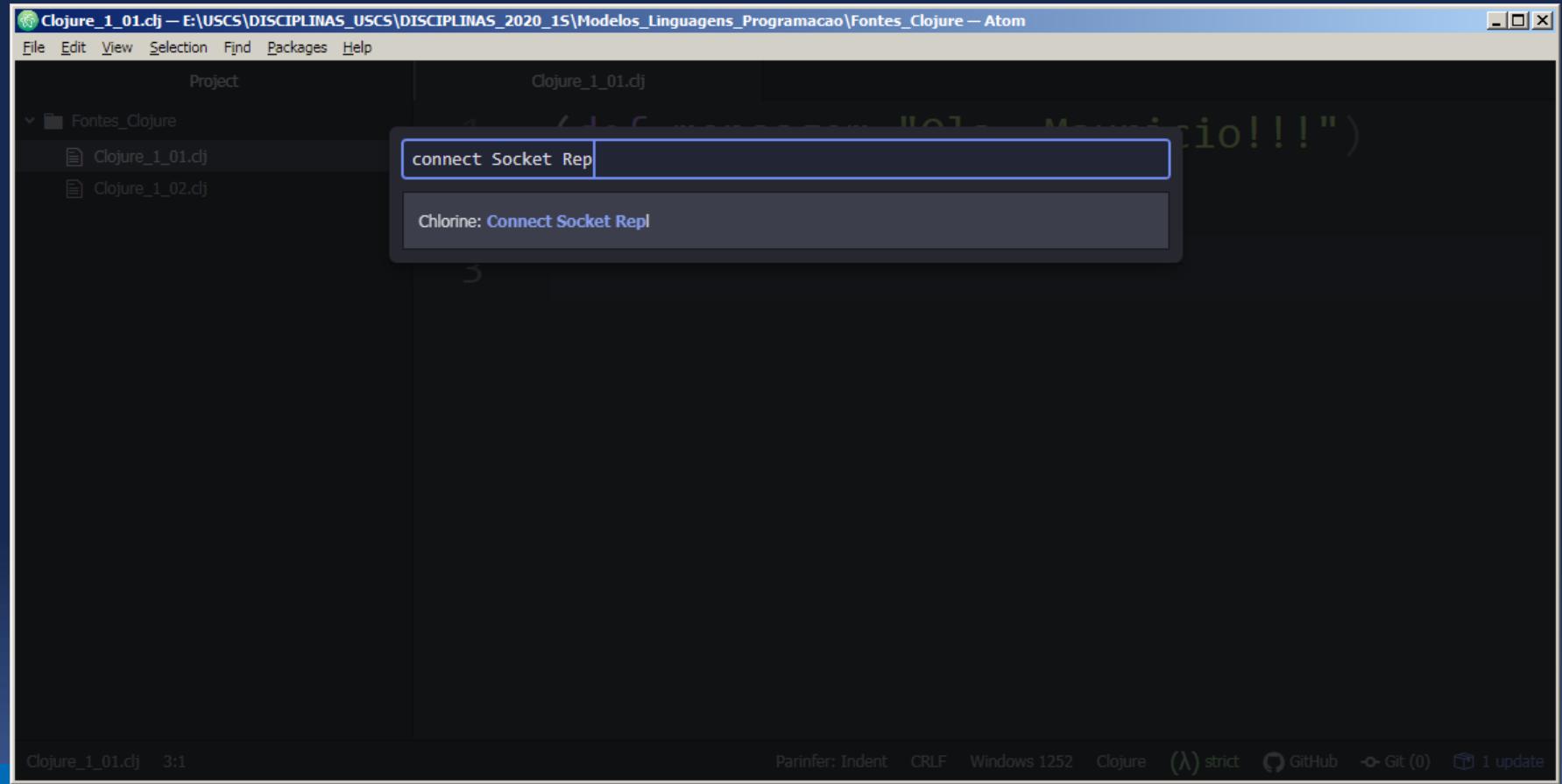


Chlorine – Configuração de Hot Keys

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

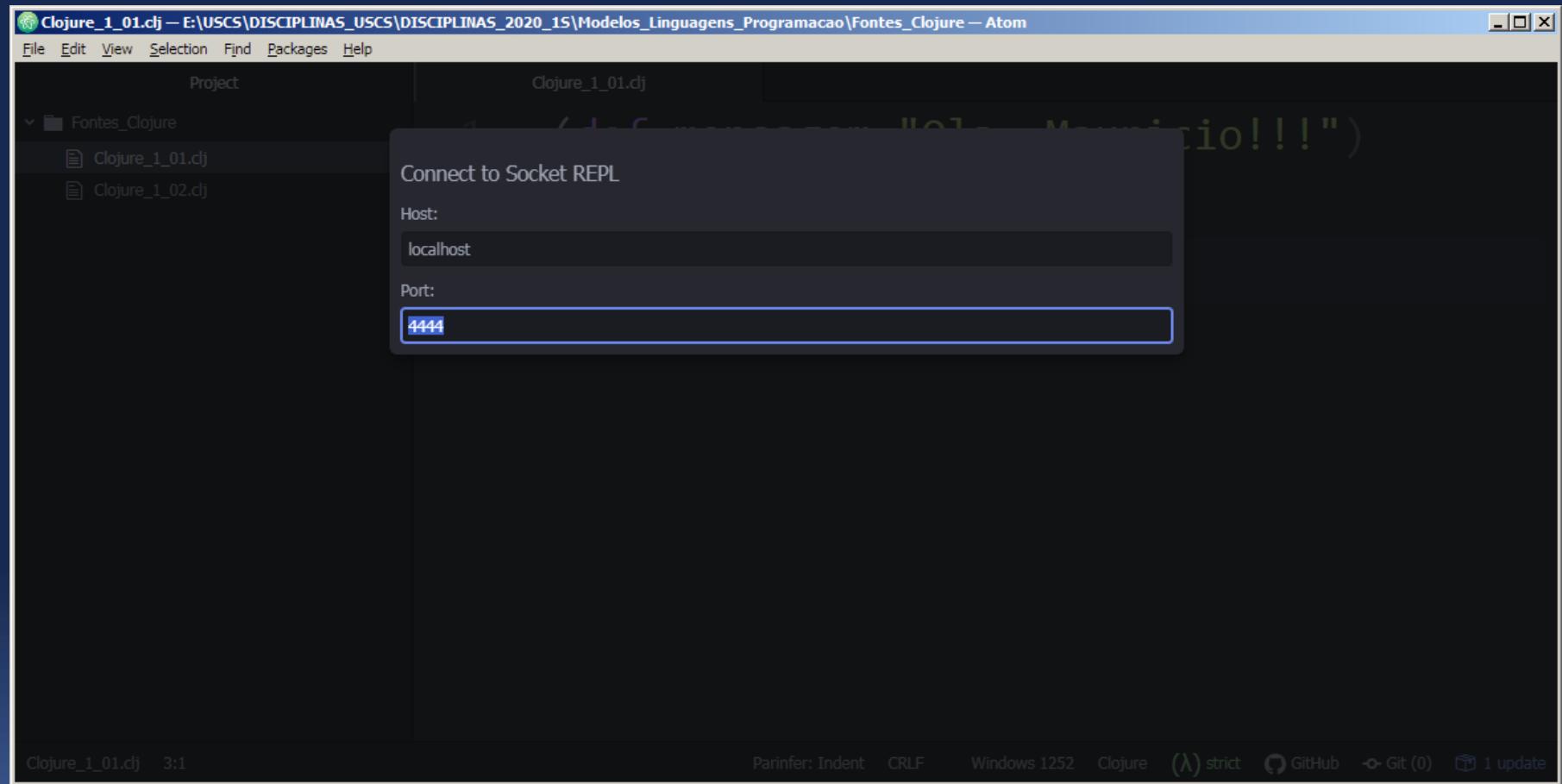
Atom – Package Chlorine

- ✓ 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 !!!

The screenshot shows the Atom code editor interface. On the left, the Project panel lists a folder named 'Fontes_Clojure' containing two files: 'Clojure_1_01.clj' and 'Clojure_1_02.clj'. The main editor area displays the following Clojure code:

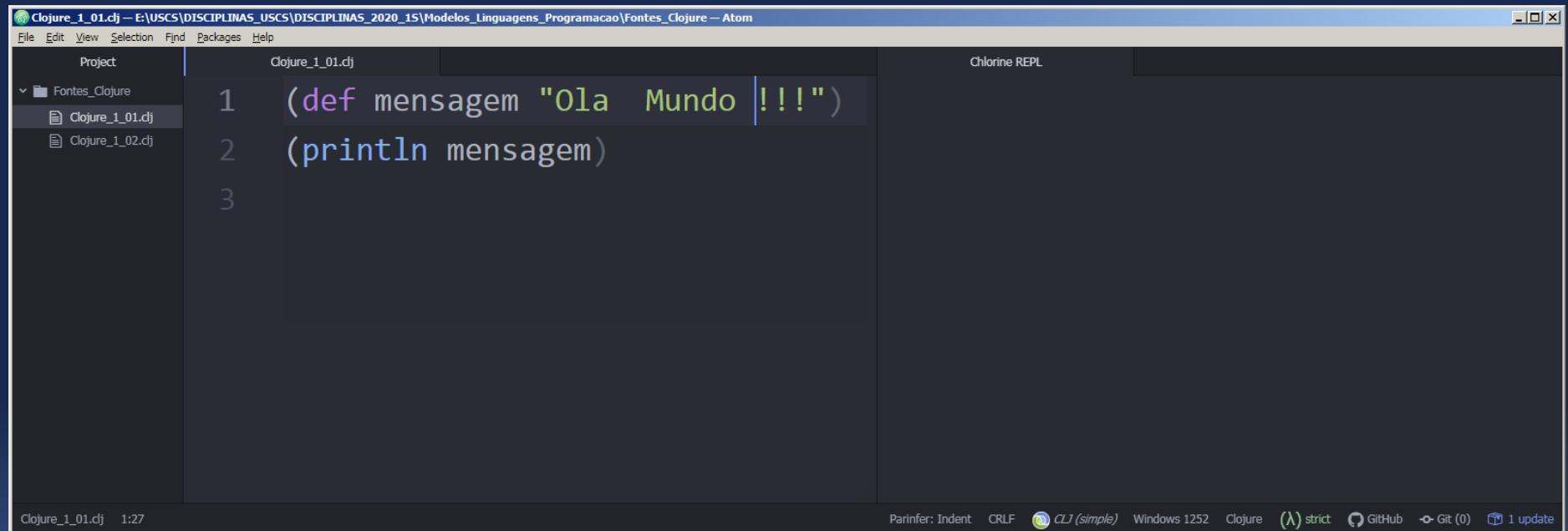
```
1 (def mensagem "Olá")
2 (println mensagem)
```

A yellow warning dialog box titled 'Chlorine REPL' is overlaid on the editor. It contains the message: 'REPL already connected' and 'REPL is already connected. Please, disconnect the current REPL if you want to connect to another.' The status bar at the bottom of the Atom window shows the file path 'Clojure_1_01.clj', line '3:1', and various tool icons.



Atom – Package Chlorine

✓ Escrever código *Clojure* (extensão .clj)



The screenshot shows the Atom code editor interface. The title bar reads "Clojure_1_01.clj – E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_1S\Modelos_Linguagens_Programacao\Fontes_Clojure – Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The left sidebar shows a "Project" tree with "Fontes_Clojure" expanded, containing "Clojure_1_01.clj" and "Clojure_1_02.clj". The main editor area displays the following Clojure code:

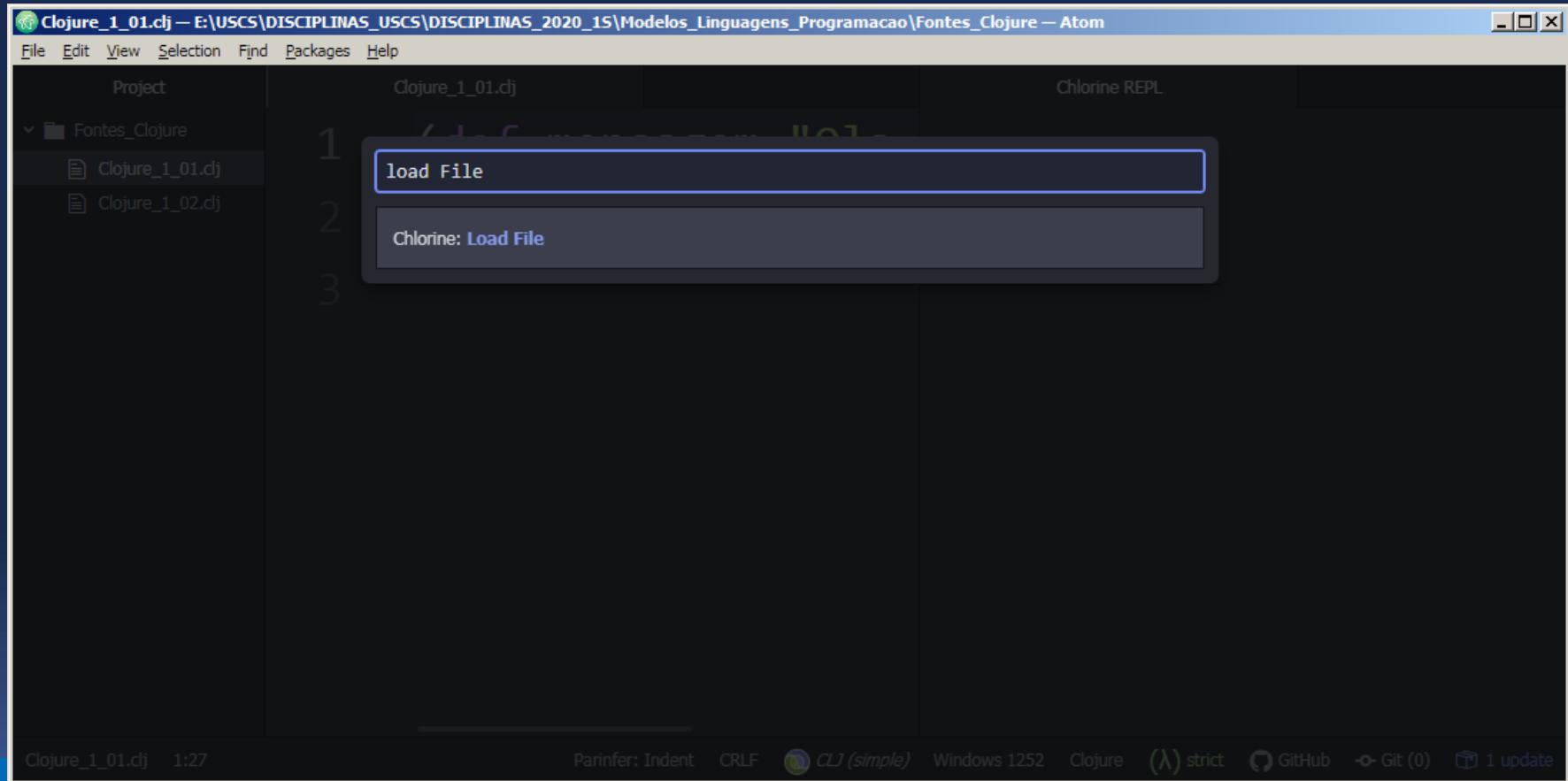
```
1 (def mensagem "Olá Mundo !!!")
2 (println mensagem)
3
```

The right panel is titled "Chlorine REPL" and is currently empty. The bottom status bar shows "Clojure_1_01.clj 1:27", "Parinfer: Indent CRLF", "CLJ (simple)", "Windows 1252", "Clojure (λ) strict", "GitHub", "Git (0)", and "1 update".



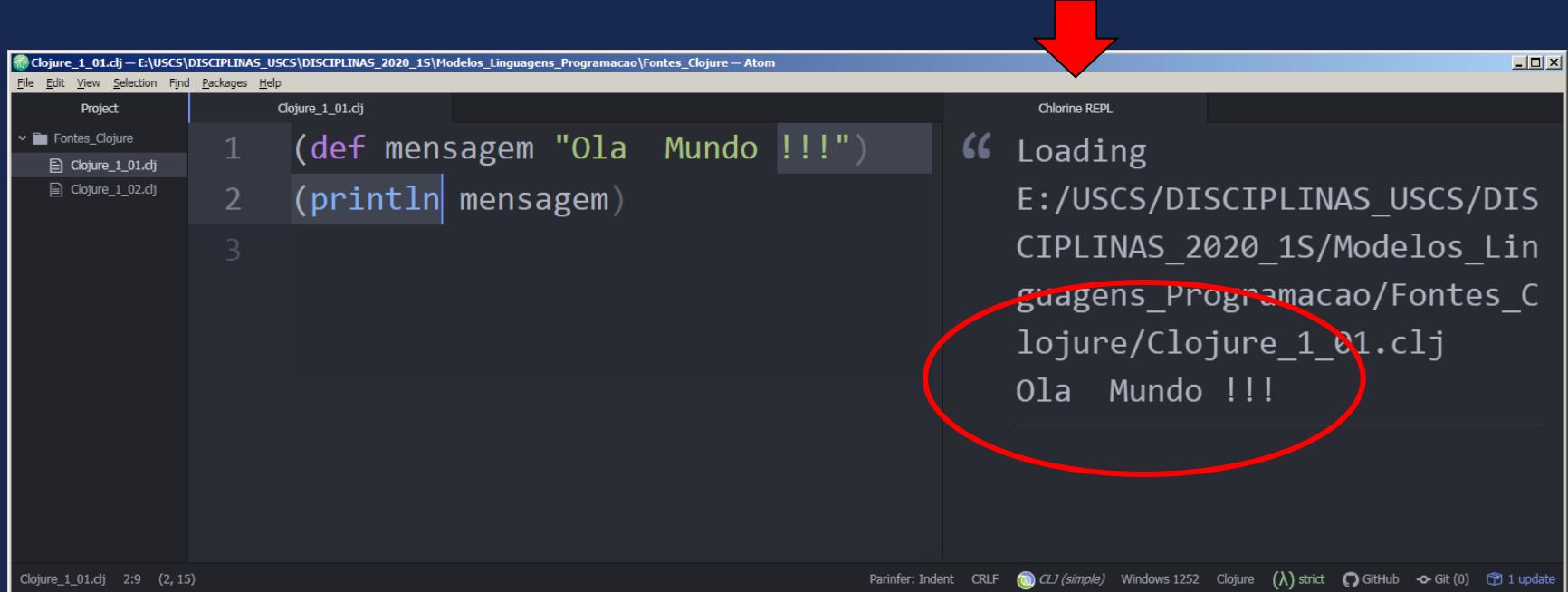
Atom – Package Chlorine

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



Atom – Package Chlorine

✓ Carga do código Clojure para o REPL



A screenshot of the Atom code editor interface. The title bar reads "Clojure_1_01.clj - E:\USCS\DISCIPLINAS_USCS\DISCIPLINAS_2020_1S\Modelos_Linguagens_Programacao\Fontes_Clojure - Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The left sidebar shows a "Project" tree with "Fontes_Clojure" expanded, containing "Clojure_1_01.clj" and "Clojure_1_02.clj". The main editor area displays the following Clojure code:

```
1 (def mensagem "Olá Mundo !!!")
2 (println mensagem)
3
```

The right panel is titled "Chlorine REPL" and shows the output of the code execution:

```
“ Loading
E:/USCS/DISCIPLINAS_USCS/DISCIPLINAS_2020_1S/Modelos_Linguagens_Programacao/Fontes_Clojure/Clojure_1_01.clj
Olá Mundo !!!
```

A large red arrow points from the text "Carga do código Clojure para o REPL" above to the "Chlorine REPL" output panel. A red oval highlights the output text "Olá Mundo !!!".

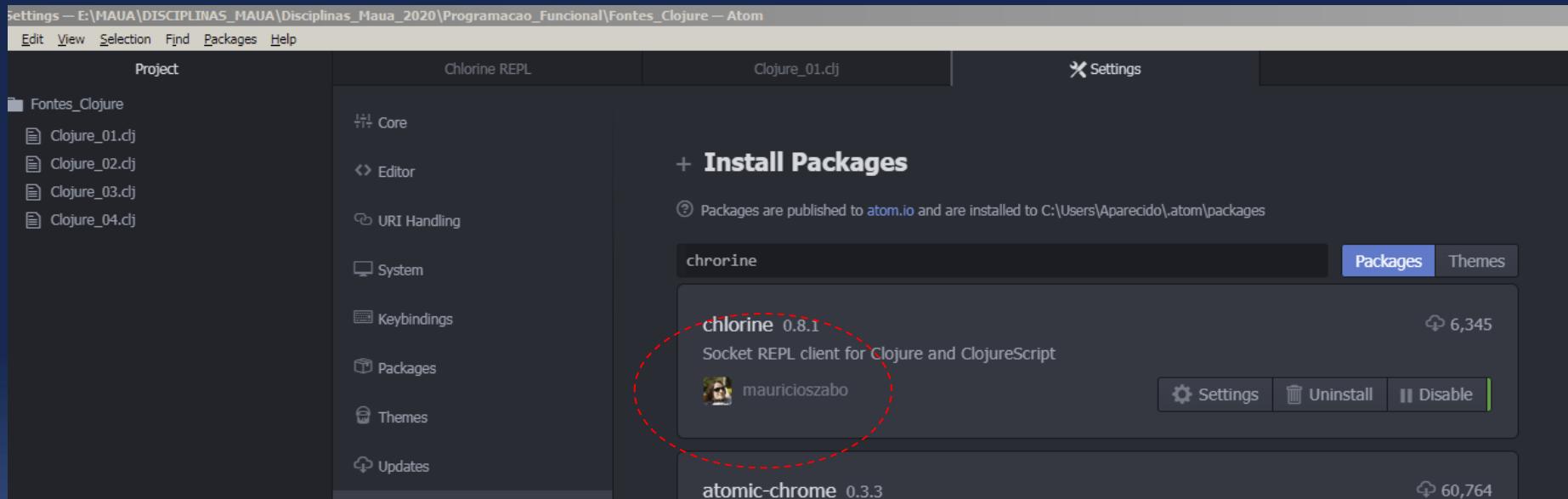


Atom – Package Chlorine + nREPL (Leiningen)



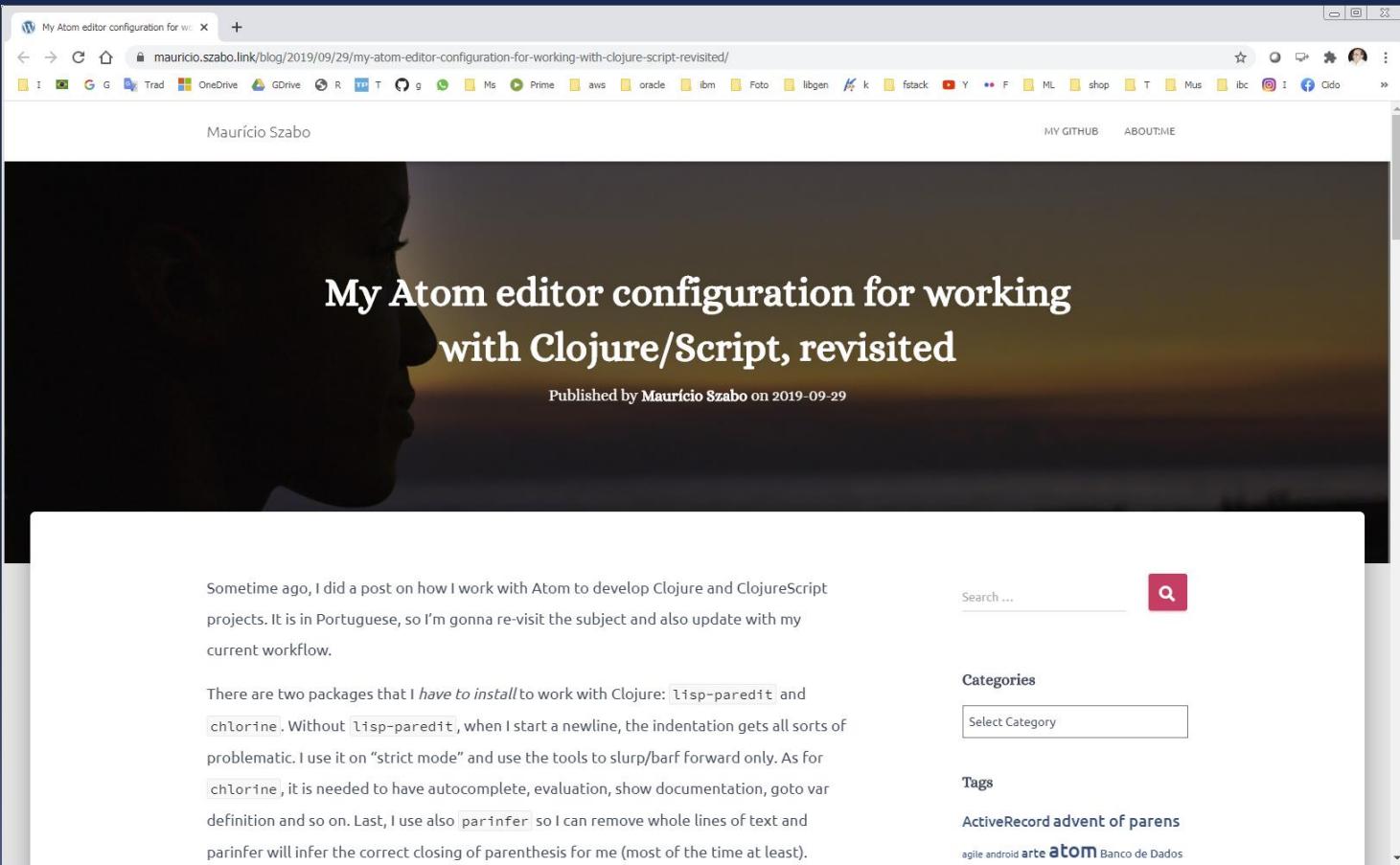
Configuração do Package Chlorine no Atom

- ✓ 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/>



My Atom editor configuration for working with Clojure/Script, revisited

Published by Maurício Szabo on 2019-09-29

Sometime ago, I did a post on how I work with Atom to develop Clojure and ClojureScript projects. It is in Portuguese, so I'm gonna re-visit the subject and also update with my current workflow.

There are two packages that I *have to install* to work with Clojure: `lisp-paredit` and `chlorine`. Without `lisp-paredit`, when I start a newline, the indentation gets all sorts of problematic. I use it on "strict mode" and use the tools to slurp/barf forward only. As for `chlorine`, it is needed to have autocomplete, evaluation, show documentation, goto var definition and so on. Last, I use also `parinfer` so I can remove whole lines of text and `parinfer` will infer the correct closing of parenthesis for me (most of the time at least).

Categories

Select Category

Tags

ActiveRecord advent of parens
agile android arte atom Banco de Dados

Configuração do Package Chlorine no Atom

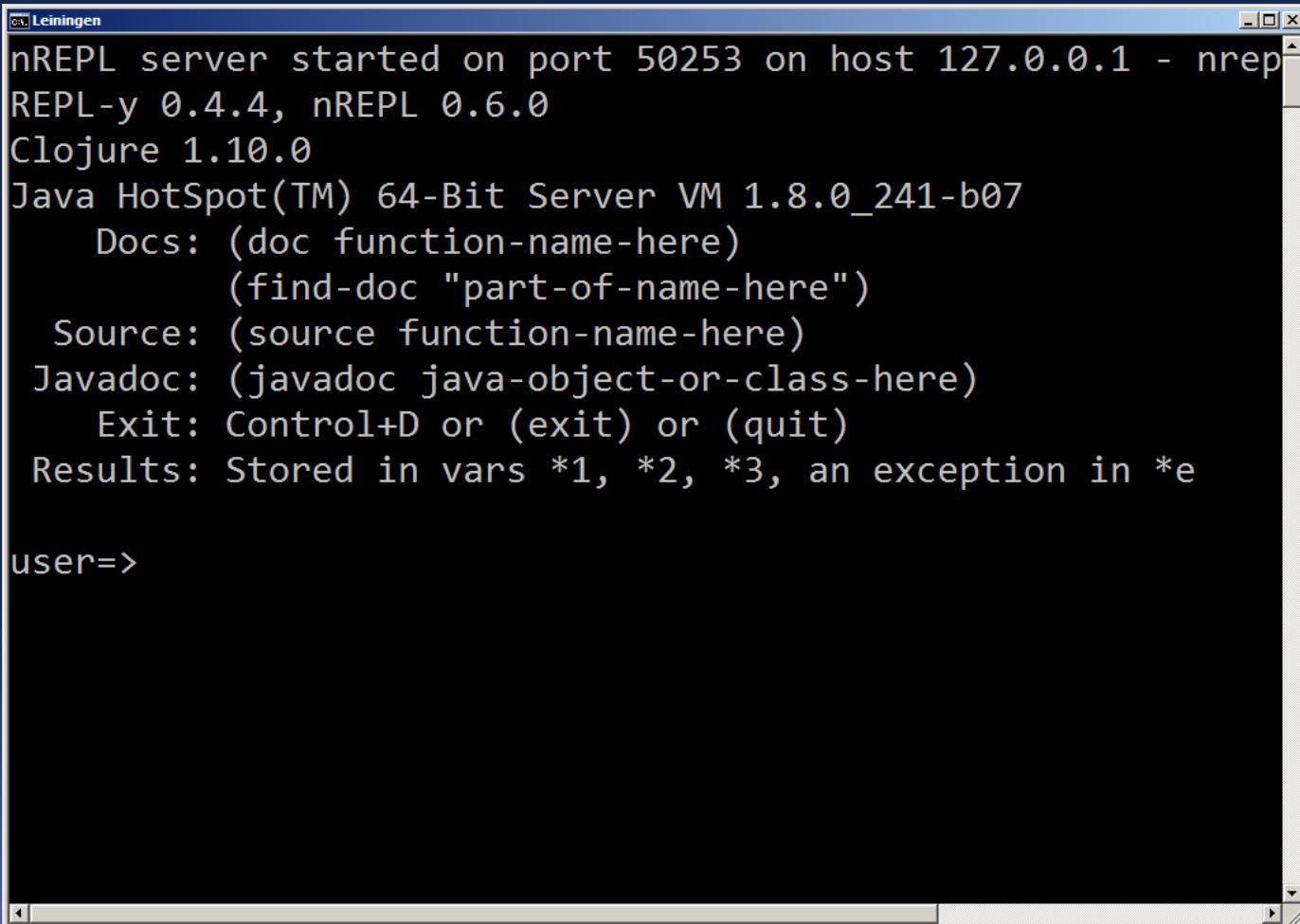
- ✓ Sob **Atom**, teclar **Ctrl + Shift + p** e procurar “**Open your keymap**”;
- ✓ Após abertura do arquivo para customizar **keys**, incluir:

```
'atom-text-editor':  
  'ctrl-k':          'chlorine:clear-console'  
  'ctrl-shift-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**

A screenshot of a terminal window titled "Leiningen". The window contains the following text:

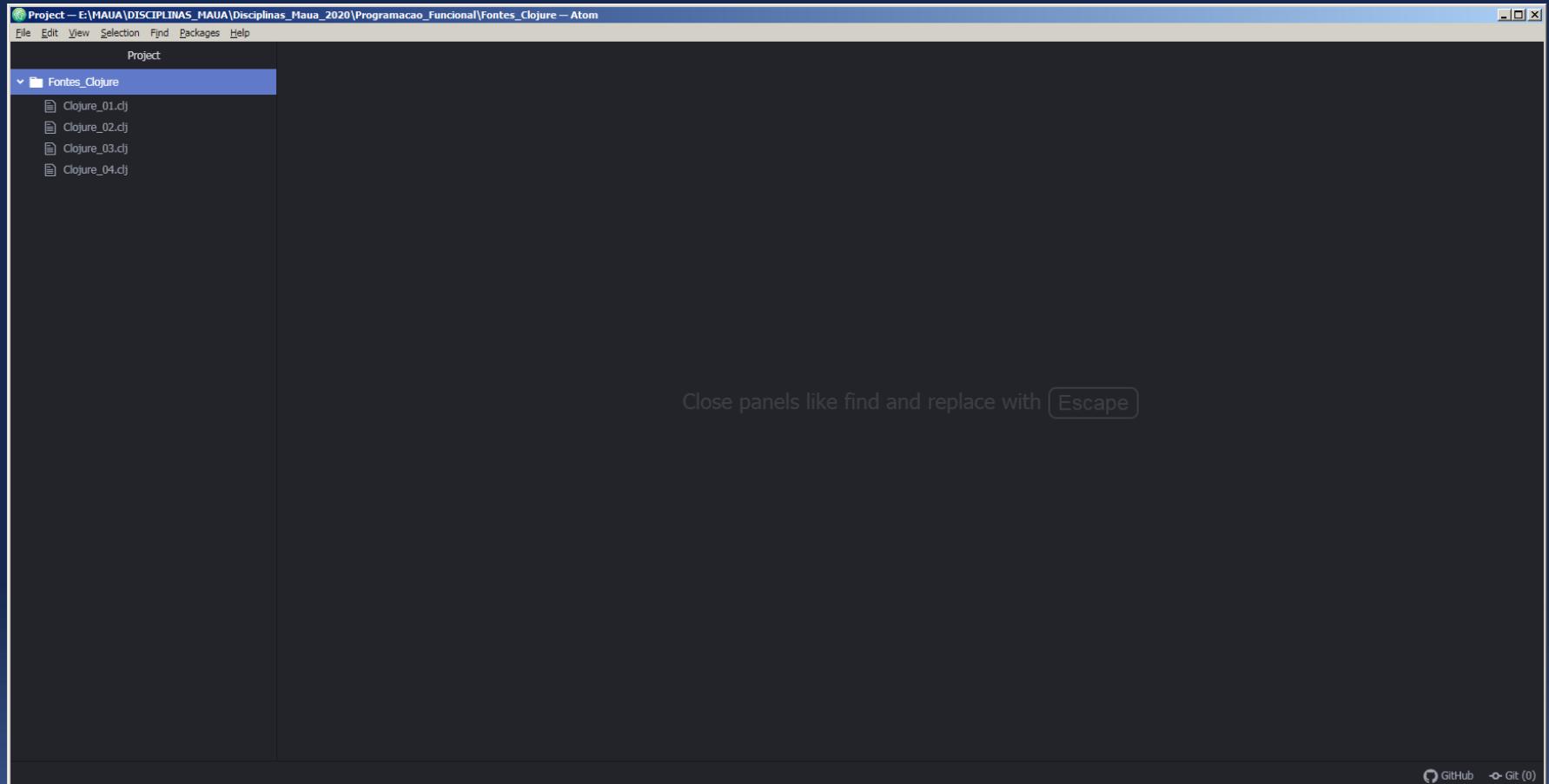
```
nREPL server started on port 50253 on host 127.0.0.1 - nrepl
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
    Docs: (doc function-name-here)
          (find-doc "part-of-name-here")
    Source: (source function-name-here)
Javadoc: (javadoc java-object-or-class-here)
    Exit: Control+D or (exit) or (quit)
Results: Stored in vars *1, *2, *3, an exception in *e

user=>
```

The terminal has a standard Windows-style title bar and scroll bars on the right side.

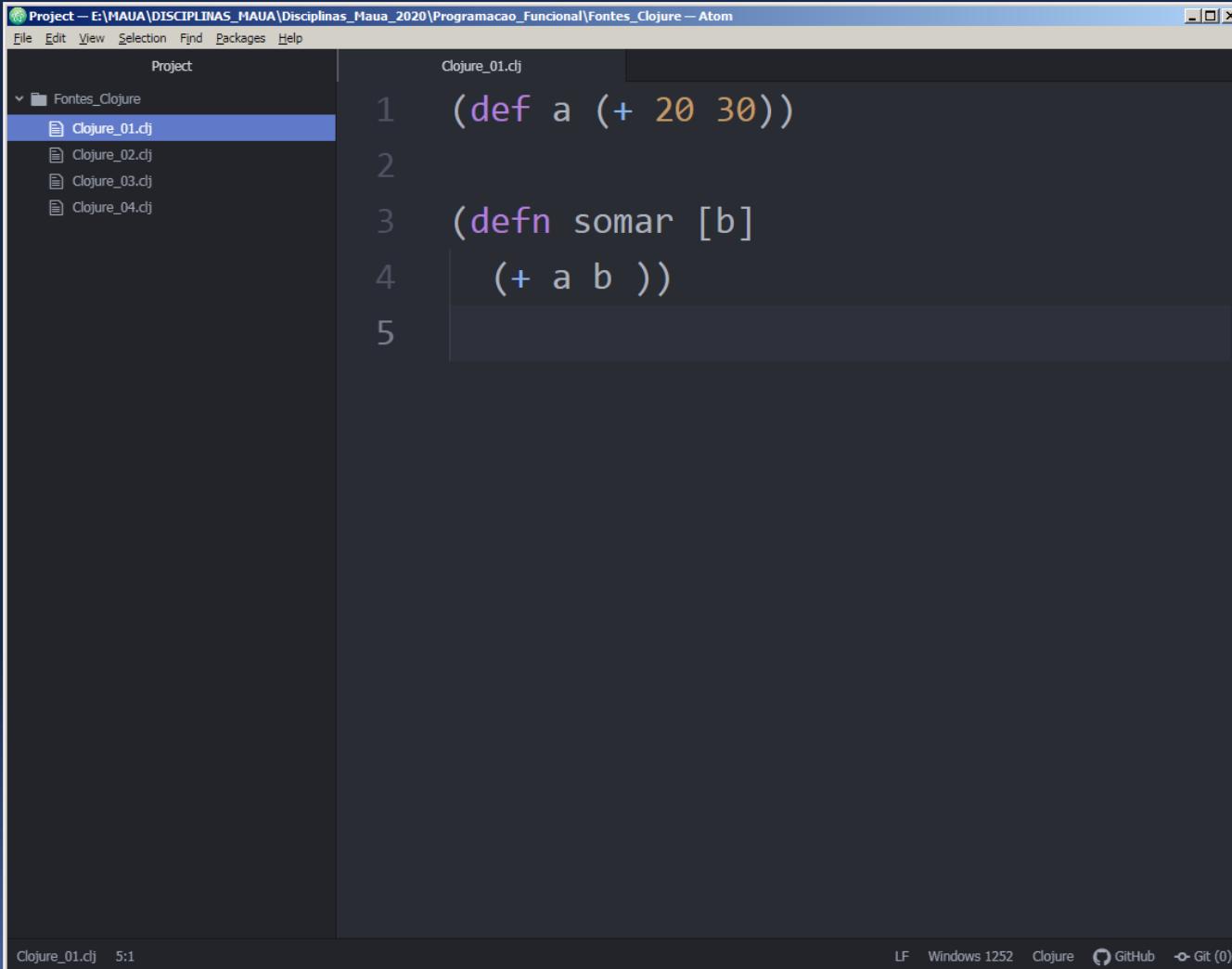
Ativar Atom

✓ File > Open Folder



Ativar Atom

✓ Escrevendo código Clojure no Atom



The screenshot shows the Atom code editor interface. The title bar reads "Project - E:\MAUA\DISCIPLINAS_MAU\Disciplinas_Maua_2020\Programacao_Funcional\Fontes_Clojure - Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The left sidebar is titled "Project" and shows a folder structure under "Fontes_Clojure": "Clojure_01.clj" (selected), "Clojure_02.clj", "Clojure_03.clj", and "Clojure_04.clj". The main editor area displays the following Clojure code:

```
(def a (+ 20 30))
(defn somar [b]
  (+ a b))
```

The status bar at the bottom shows "Clojure_01.clj 5:1" and icons for LF, Windows 1252, Clojure, GitHub, and Git (0).

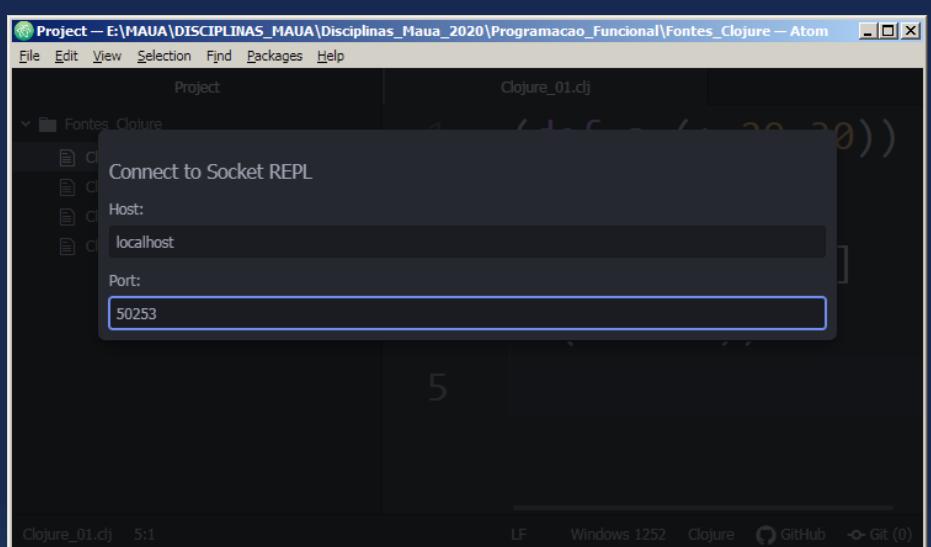


Ativar Atom

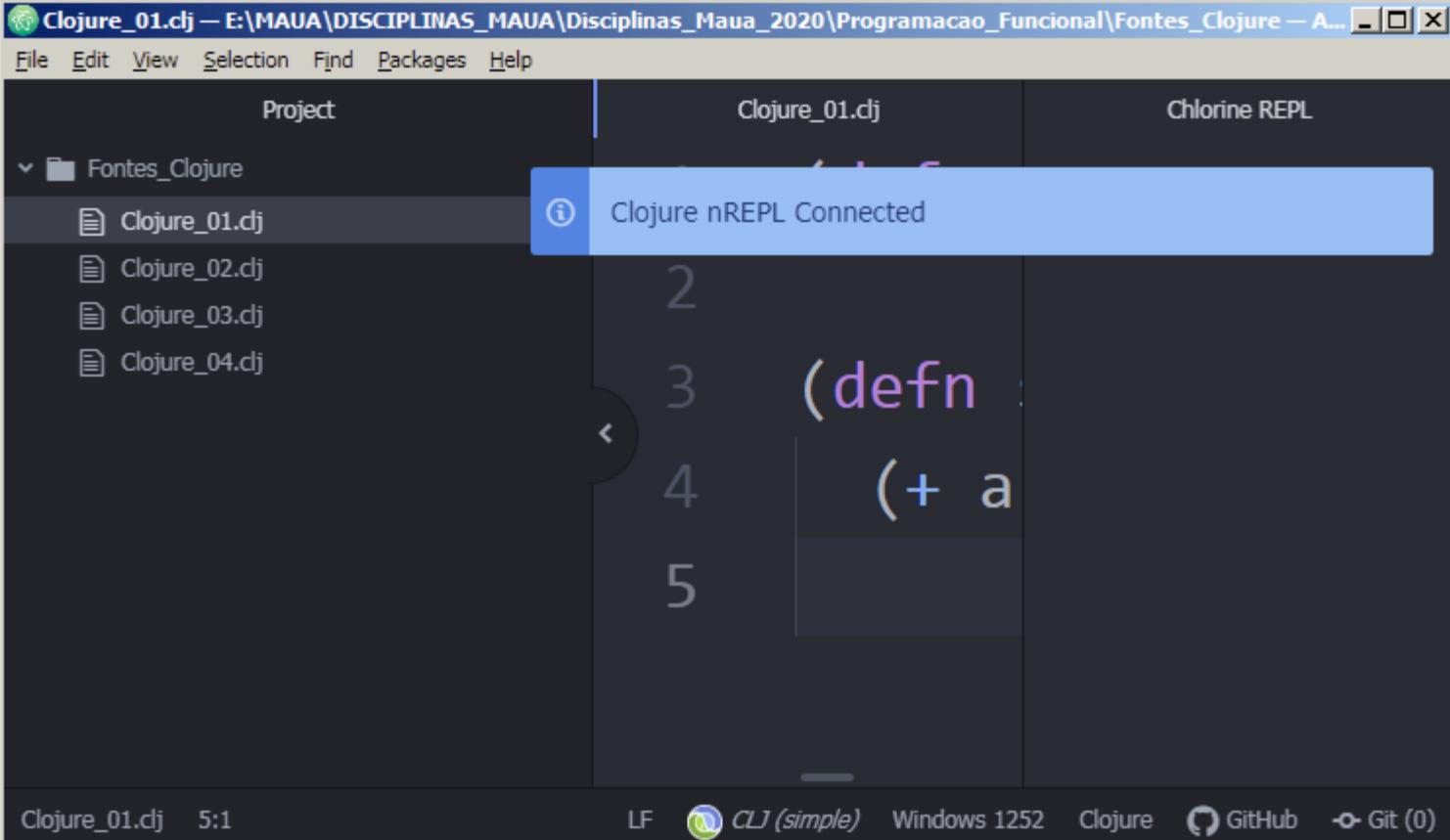
- ✓ Ativando **nREPL** (Leiningen)
- ✓ **Ctrl + Shift + p** > Connect to Socket REPL
 - > Host: **localhost**
 - > Port: aquela fornecida pela nREPL (Leiningen), por exemplo: **50253**

```
Leiningen
nREPL server started on port 50253 on host 127.0.0.1 - nrepl
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
         (find-doc "part-of-name-here")
  Source: (source function-name-here)
Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
Results: Stored in vars *1, *2, *3, an exception in *e

user=> 50253
```



Clojure nREPL Connected



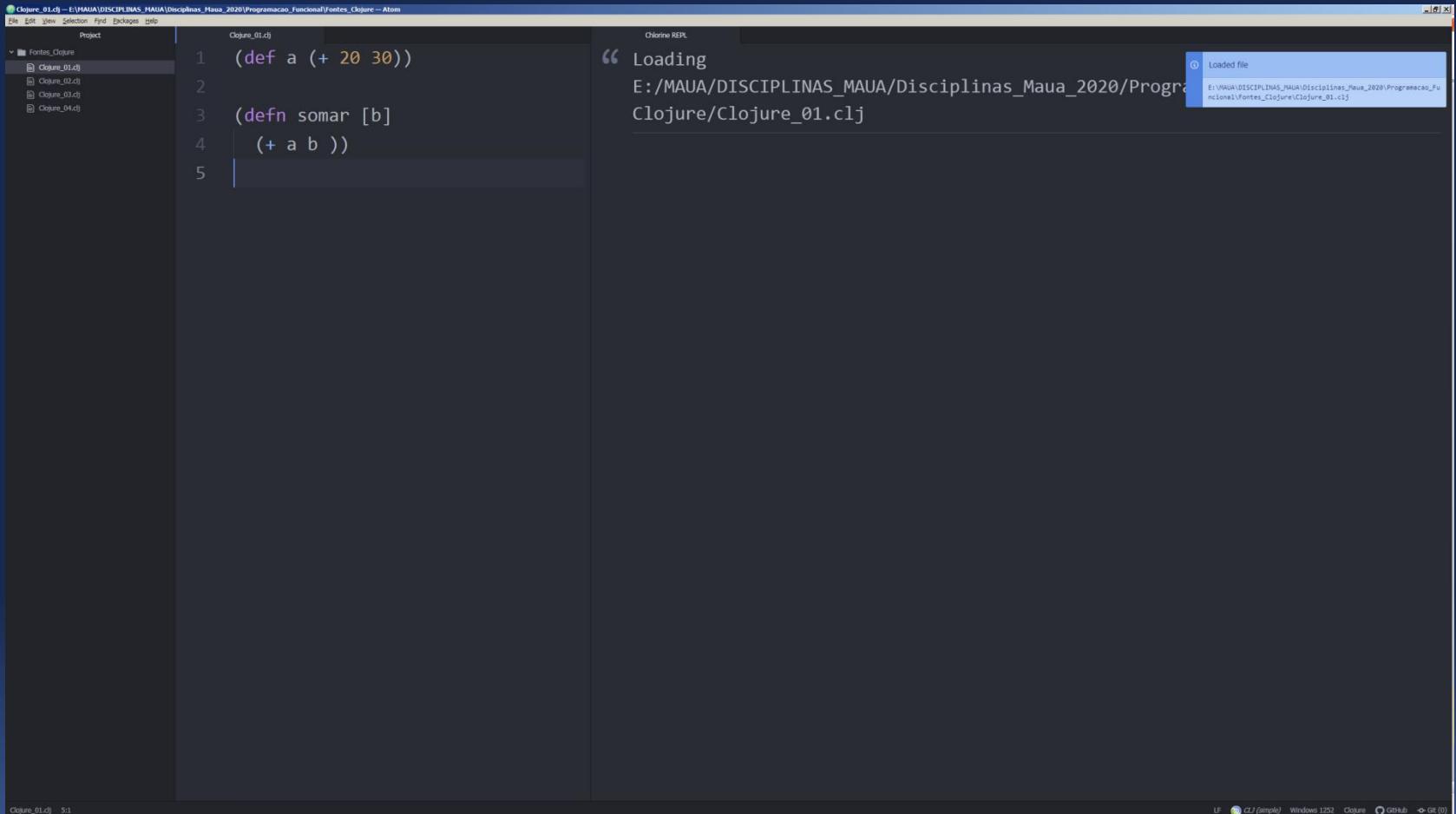
The screenshot shows a Clojure development environment. On the left, the Project view lists files: Fontes_Clojure, Clojure_01.clj, Clojure_02.clj, Clojure_03.clj, and Clojure_04.clj. The Clojure_01.clj file is selected. In the center, the code editor displays the following Clojure code:

```
2
3 (defn
4   (+ a
5
```

A tooltip for the 'defn' keyword is visible, stating "Clojure nREPL Connected". On the right, the Chlorine REPL panel shows the prompt "Clojure nREPL Connected". At the bottom, the status bar indicates "Clojure_01.clj 5:1", "LF", "CLJ (simple)", "Windows 1252", "Clojure", "GitHub", and "Git (0)".

Carregando Código para nREPL

✓ Ctrl + Shift +p > Chroline: Load File



The screenshot shows the Atom code editor with a Clojure project open. The file 'Clojure_01.clj' contains the following code:

```
(def a (+ 20 30))
(defn somar [b]
  (+ a b))
```

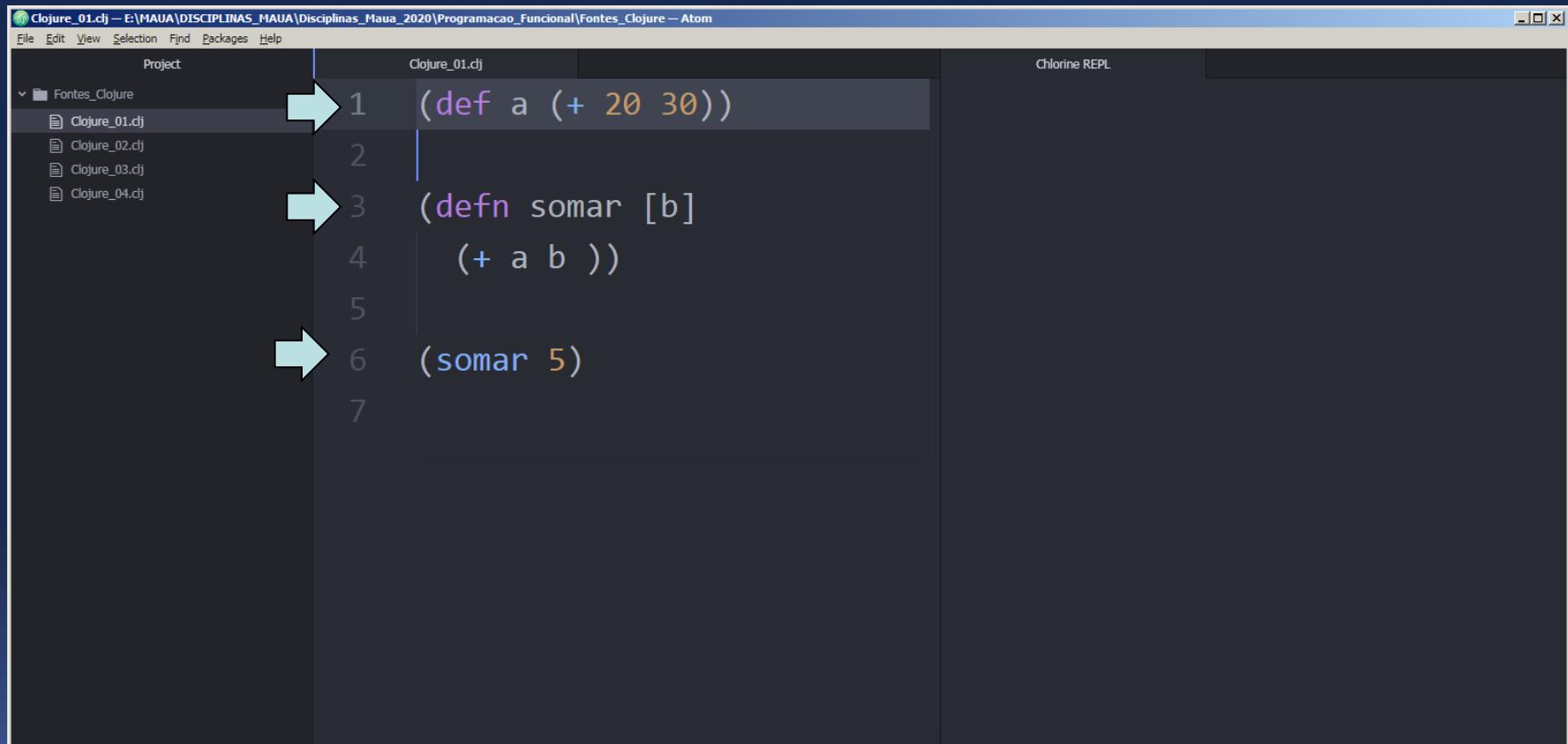
To the right, the 'Chroline REPL' panel shows the output of the 'load' command:

```
“ Loading
E:/MAUA/DISCIPLINAS_MAUΑ/Disciplinas_Maua_2020/Programacao_Funcional/Fontes_Clojure/Clojure_01.clj
Loaded file
E:/MAUA/DISCIPLINAS_MAUΑ/Disciplinas_Maua_2020/Programacao_Funcional/Fontes_Clojure/Clojure_01.clj
```

At the bottom of the screen, there are several status icons: LF, CLJ (simple), Windows 1252, Clojure, GitHub, and Git (0).

Evaluate "top-blocks"

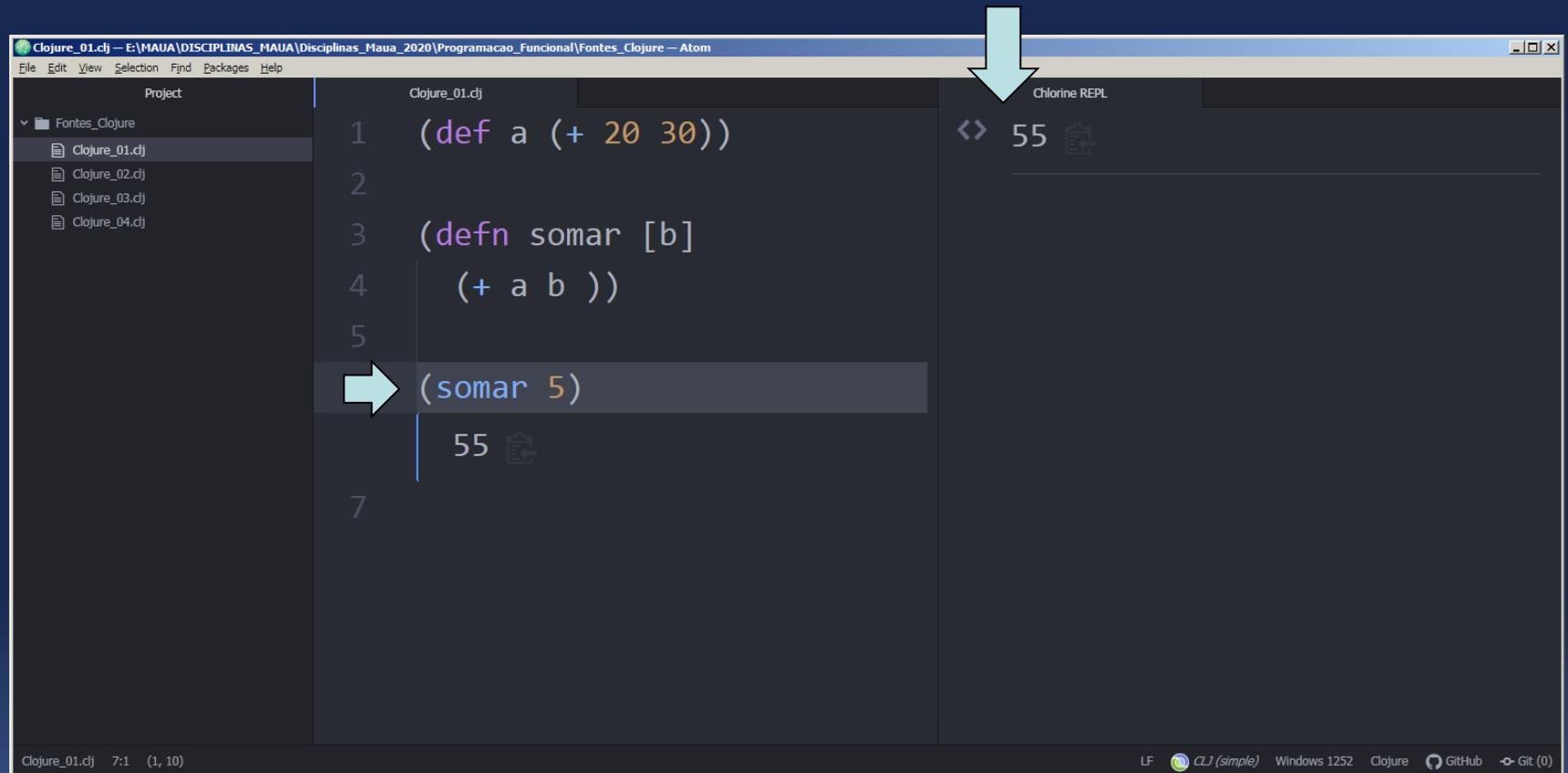
- ✓ Hot Key: **Ctrl + Enter**
- ✓ Selecionar o top block desejado e teclar => **Ctrl + Enter**



```
(def a (+ 20 30))
(defn somar [b]
  (+ a b ))
(somar 5)
```

Evaluate "top-blocks"

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



The screenshot shows the Atom code editor interface. On the left, the project structure includes files like Clojure_01.clj, Clojure_02.clj, Clojure_03.clj, and Clojure_04.clj. The main editor pane displays Clojure code:

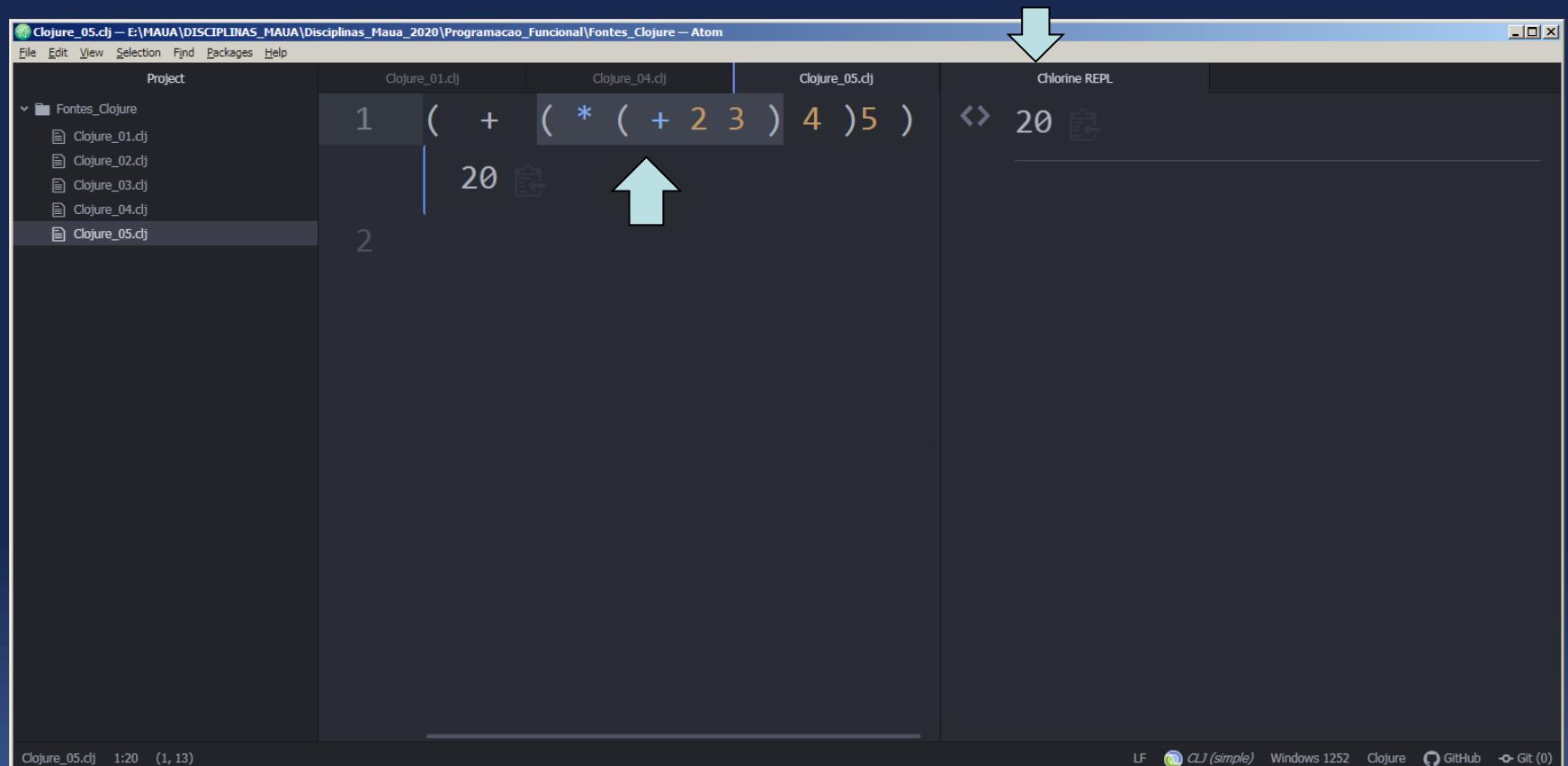
```
1 (def a (+ 20 30))
2
3 (defn somar [b]
4   (+ a b ))
5
6 (somar 5)
7
```

A light blue arrow points from the line '(somar 5)' in the editor to the same line in the 'Chlorine REPL' panel on the right, which shows the result '55'. The REPL panel also has a small icon of a clipboard with a checkmark.

At the bottom of the Atom window, status icons include LF, CLJ (simple), Windows 1252, Clojure, GitHub, and Git (0).

Evaluate “blocks”

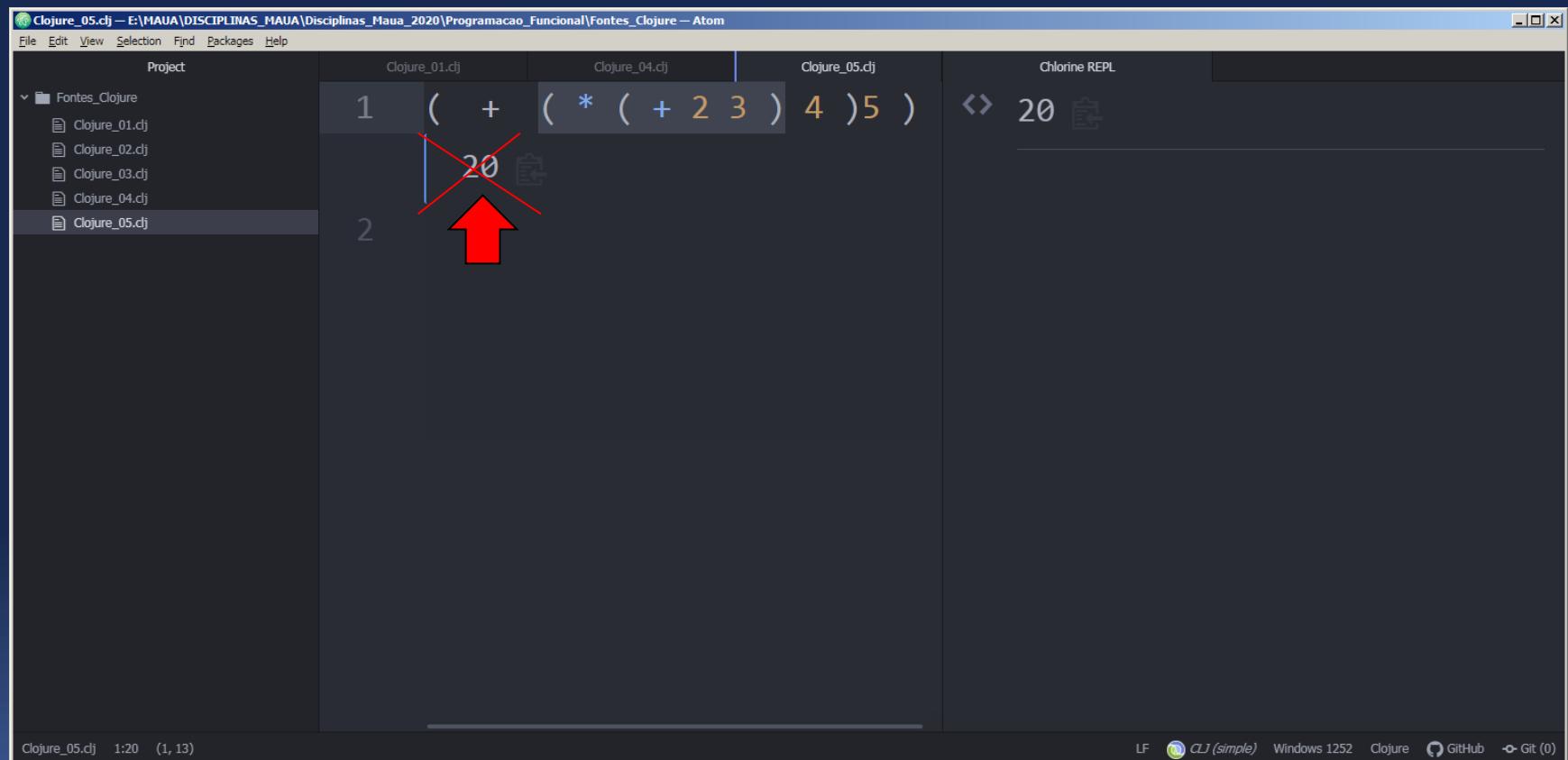
- ✓ Hot Key: **Ctrl + Shift + Enter**
- ✓ Selecionar o block desejado e teclar => **Ctrl + Shift + Enter**



A screenshot of the Atom code editor interface. The title bar reads "Clojure_05.clj – E:\MAUA\DISCIPLINAS_MAU\Disciplinas_Maua_2020\Programacao_Funcional\Fontes_Clojure – Atom". The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. On the left, a sidebar titled "Project" shows a folder named "Fontes_Clojure" containing files: Clojure_01.clj, Clojure_02.clj, Clojure_03.clj, Clojure_04.clj, and Clojure_05.clj, with Clojure_05.clj currently selected. The main editor area displays three lines of Clojure code:
1 (+ (* (+ 2 3) 4) 5)
2 20
3
A large blue arrow points upwards from the number 20 towards the selection bar. The selection bar itself has a blue background and highlights the expression "(* (+ 2 3) 4) 5 ". To the right of the editor is a "Chlorine REPL" panel showing the result "20". The status bar at the bottom shows "Clojure_05.clj 1:20 (1, 13)" and various system and application icons.

Clear Inline Results

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



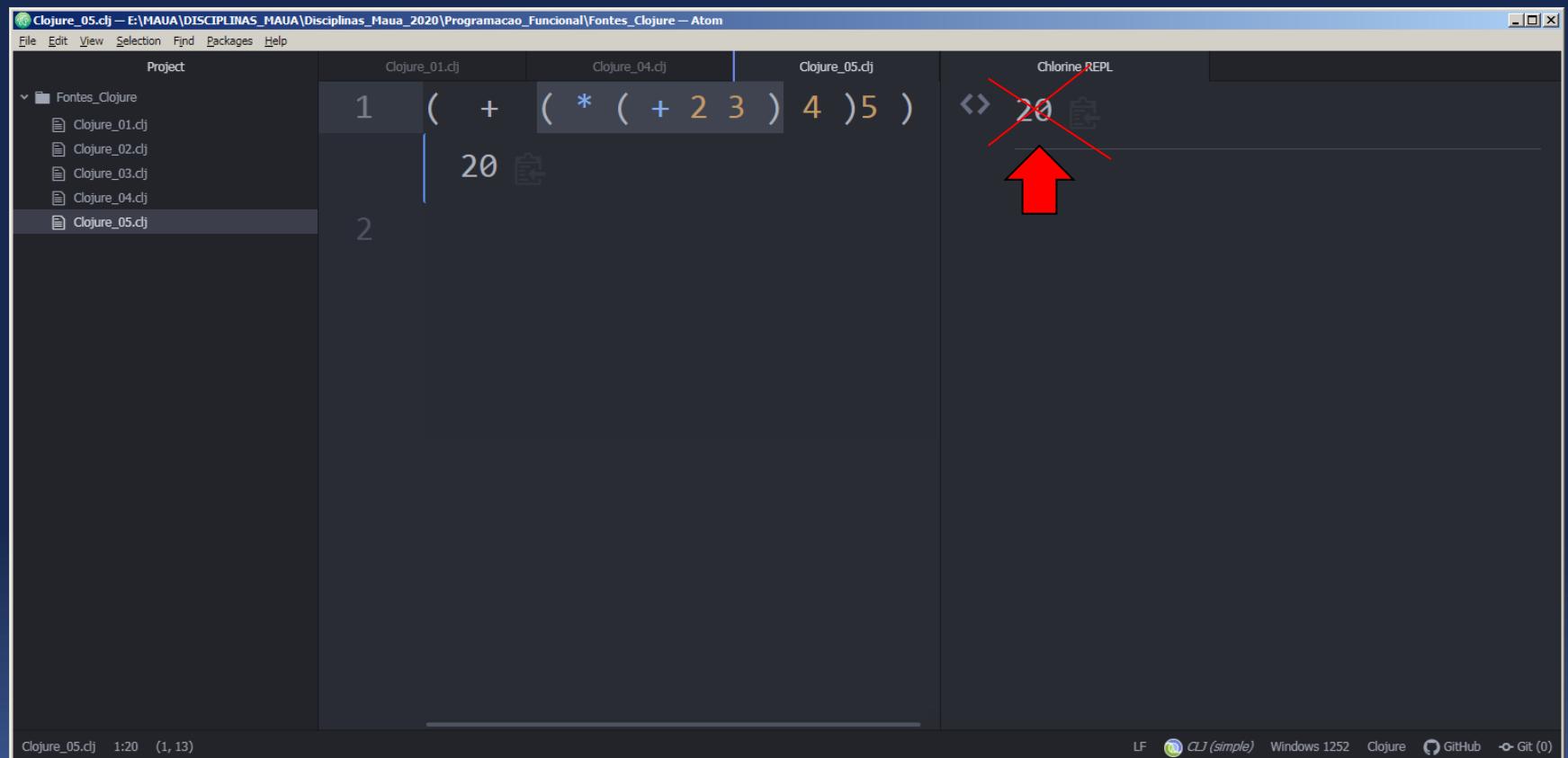
The screenshot shows the Atom code editor interface. On the left, the Project sidebar lists files: Clojure_01.clj, Clojure_02.clj, Clojure_03.clj, Clojure_04.clj, and Clojure_05.clj (which is currently selected). The main editor area displays Clojure code:

```
1 ( + (* (+ 2 3) 4) 5 )  
2 20
```

A red arrow points upwards from the number 20 in line 2 towards the inline result. A red 'X' is drawn over the inline result '20'. The right panel shows the Chlorine REPL output: 20. The status bar at the bottom indicates: Clojure_05.clj 1:20 (1, 13), LF, CLJ (simple), Windows 1252, Clojure, GitHub, Git (0).

Clear Console

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



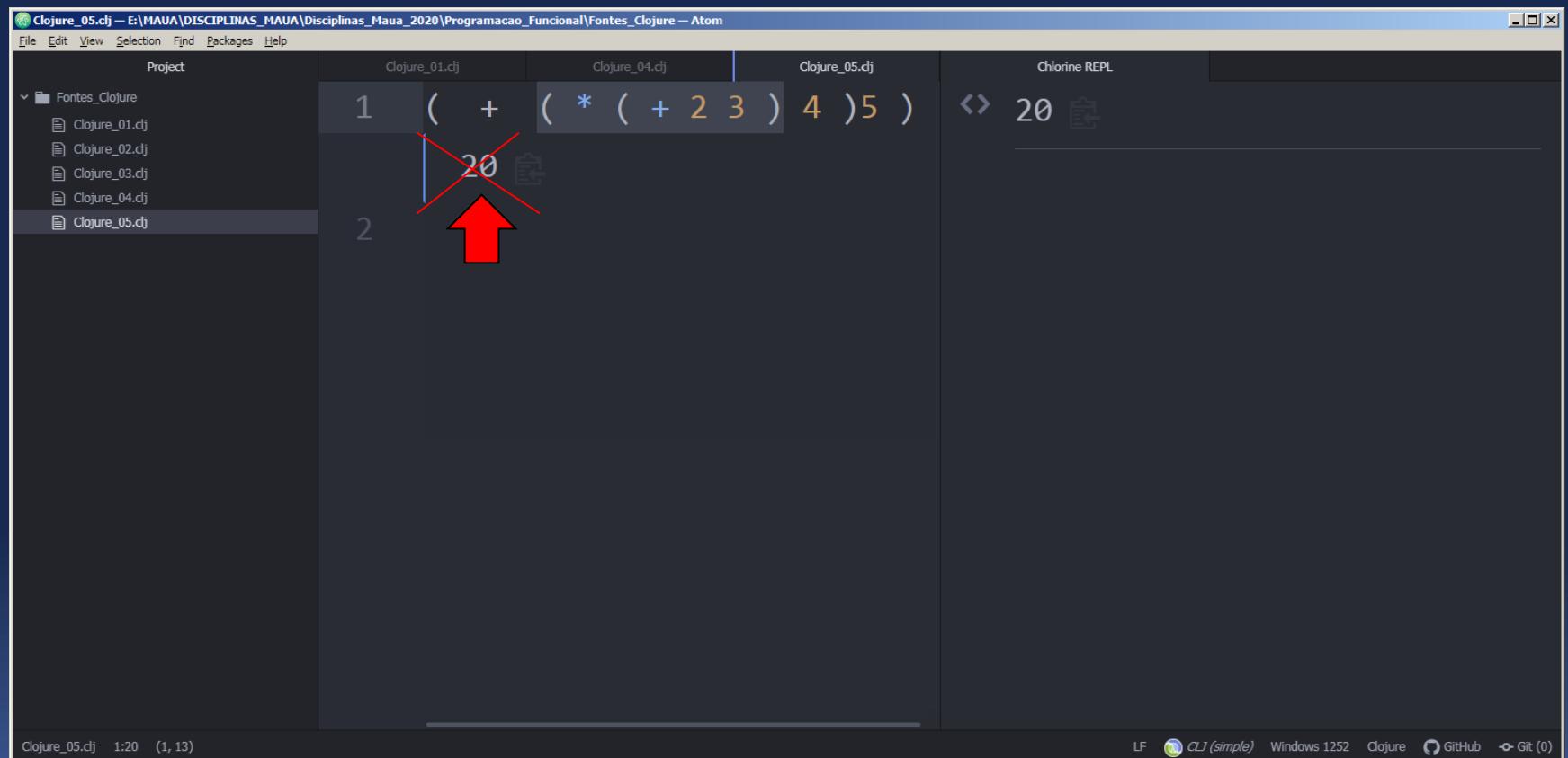
The screenshot shows the Atom code editor interface. On the left, the project tree displays files under 'Fontes_Clojure': Clojure_01.clj, Clojure_02.clj, Clojure_03.clj, Clojure_04.clj, and Clojure_05.clj (which is currently selected). The main workspace contains three tabs: Clojure_01.clj, Clojure_04.clj, and Clojure_05.clj. The Clojure_05.clj tab shows the following code:

```
1 ( + ( * ( + 2 3 ) 4 ) 5 )
2 20
```

To the right of the code editor is the 'Chlorine REPL' panel, which displays the output of the last command: '20'. A large red arrow points upwards from the bottom of the slide towards this REPL panel.

Clear console

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



The screenshot shows the Atom code editor interface. On the left, the Project sidebar lists files: Clojure_01.clj, Clojure_02.clj, Clojure_03.clj, Clojure_04.clj, and Clojure_05.clj (which is currently selected). The main editor area displays Clojure code:

```
1 ( + (* (+ 2 3) 4) 5 )  
2 20
```

A large red arrow points upwards from the crossed-out value '20' towards the code. To the right of the editor is the 'Chlorine REPL' panel, which shows the output '20'. The status bar at the bottom indicates the file is Clojure_05.clj, the time is 1:20, and the line number is (1, 13).

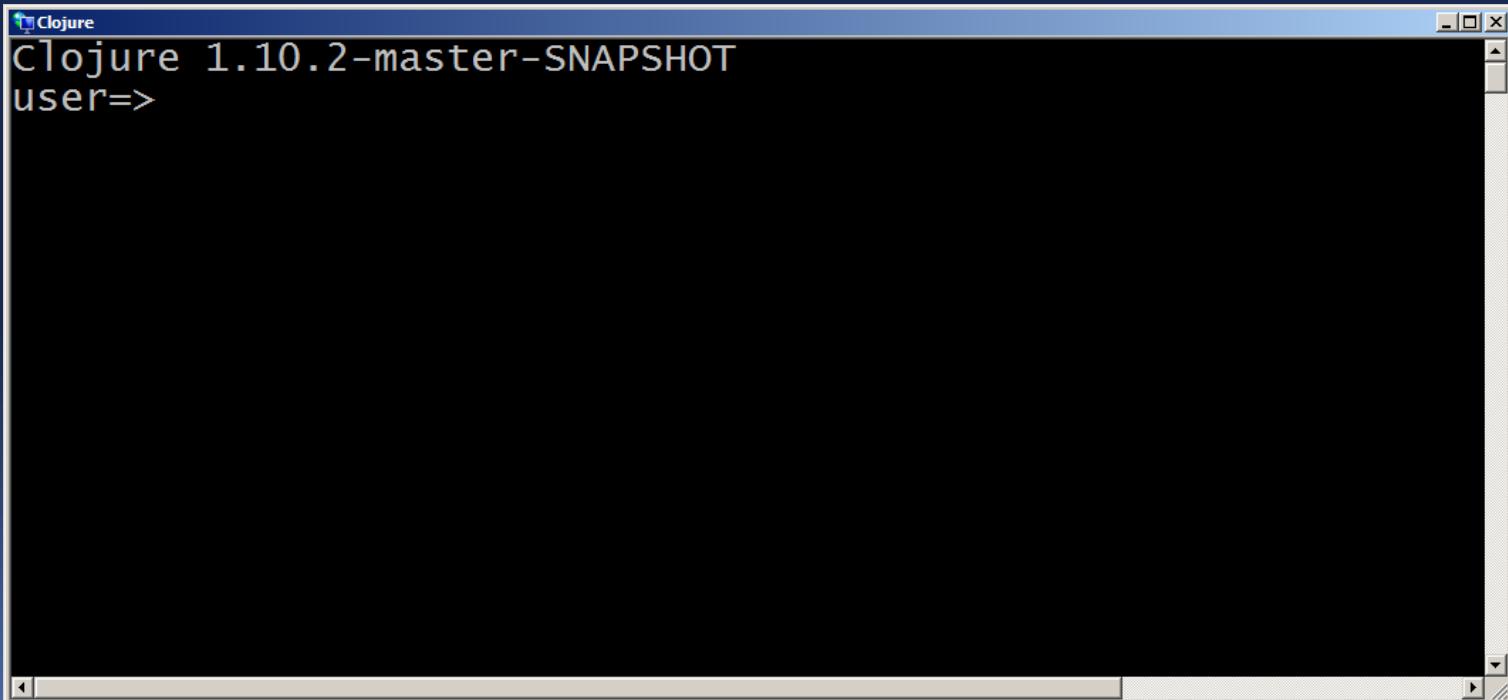


Desenvolvimento com REPL



Usando REPL

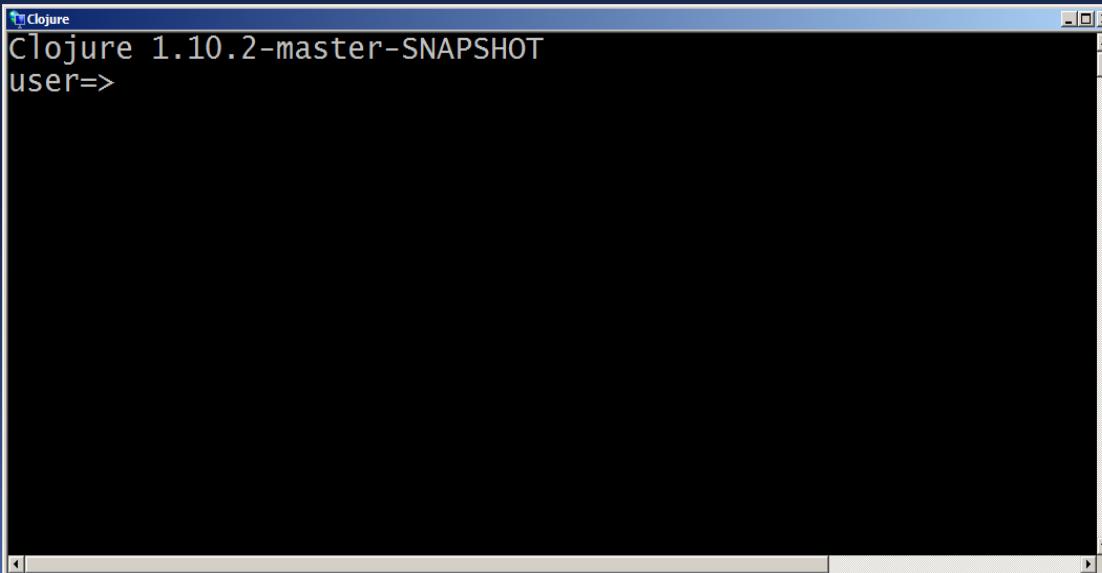
- ✓ REPL significa **READ EVAL PRINT LOOP** e representa um bom recurso para iniciar o estudo de **Clojure**;
- ✓ É uma **interface** de **comandos** que permite a avaliação direta de código **Clojure**;
- ✓ No prompt do REPL, a primeira linha representa a **versão** do Clojure, o qual em nosso caso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



A screenshot of a Windows-style application window titled "Clojure". The title bar also displays the text "Clojure 1.10.2-master-SNAPSHOT". The main window is black and contains white text. It shows the Clojure version information and a user prompt: "user=>". There is no further input or output displayed in the window.

Usando REPL

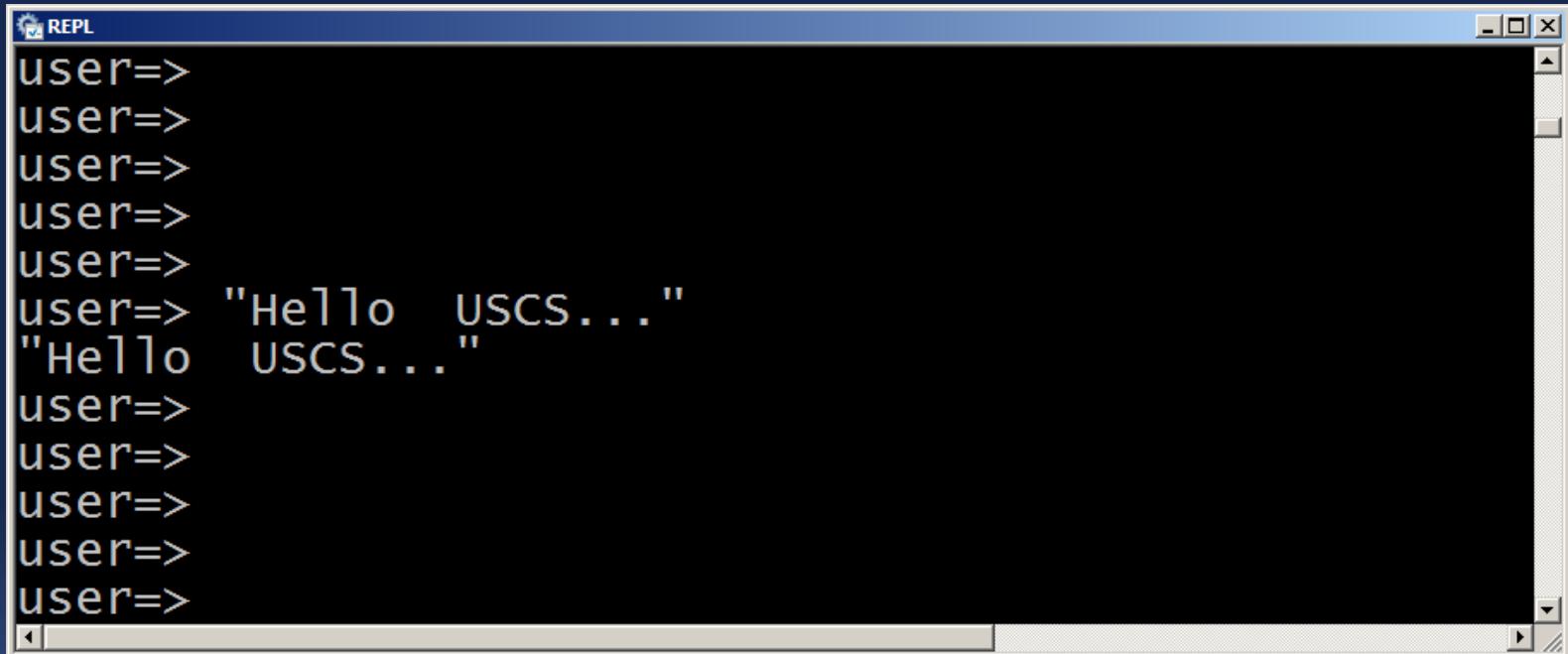
- ✓ A segunda linha exibe o **namespace** corrente (**user**) e solicita input do usuário;
- ✓ Um **namespace** é um grupo de coisas (tais como funções) que estão agrupadas em um espaço;
- ✓ Aqui, nesse caso, tudo que for criado estará no **namespace user** por default;
- ✓ **REPL** está agora pronto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



A screenshot of a Windows-style application window titled "Clojure 1.10.2-master-SNAPSHOT". The window has a dark background and contains the text "user=>" in white, indicating the current namespace and the REPL prompt. There is a vertical scroll bar on the right side of the window.

Avaliando expressões

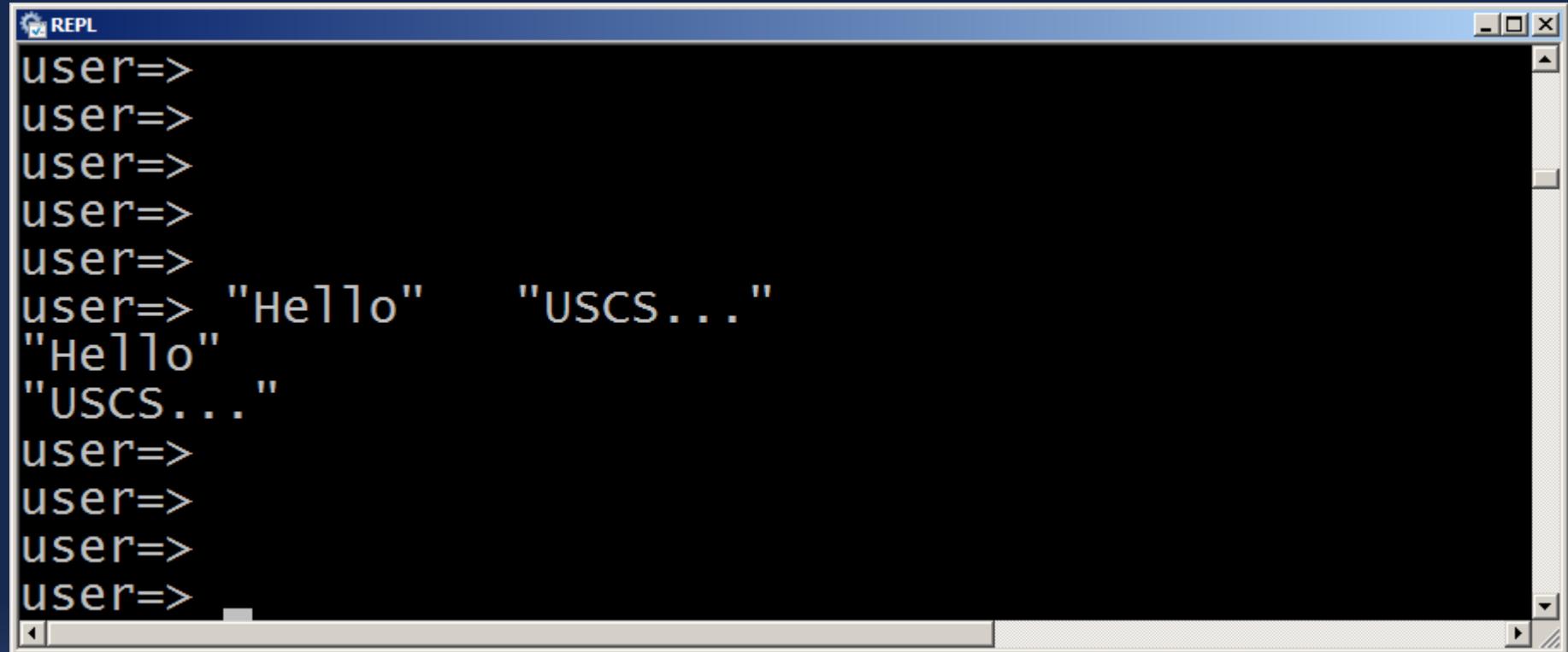
- ✓ Em **Clojure**, **literais string** são criados com aspas duplas, " ";
- ✓ Um **literal** é uma notação para representar valores **fixos** no código fonte.



The screenshot shows a Windows-style application window titled "REPL". Inside the window, the Clojure REPL is running. The history of input and output is as follows:

```
user=>
user=>
user=>
user=>
user=>
user=> "Hello USCS..."
"Hello USCS..."
user=>
user=>
user=>
user=>
user=>
```

Avaliando múltiplos strings

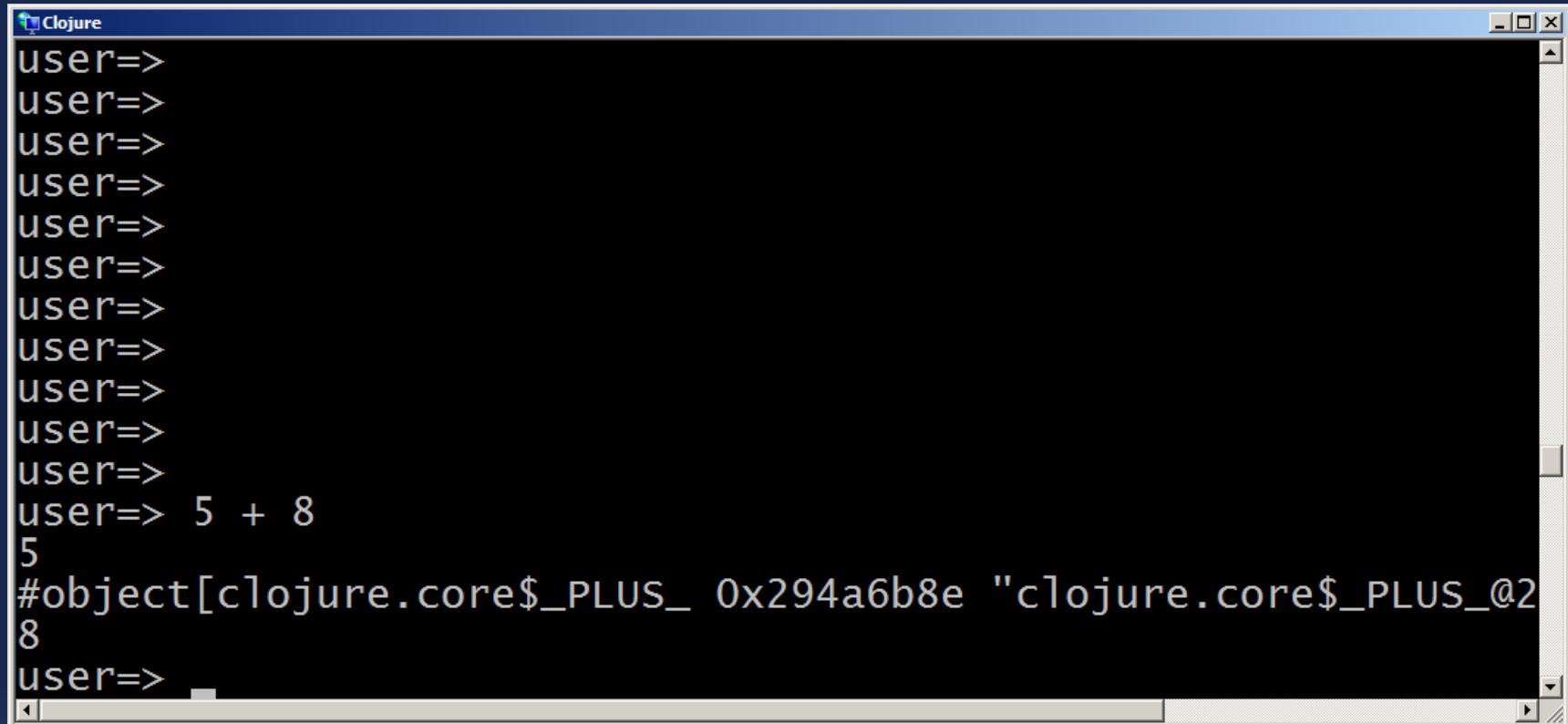
A screenshot of a REPL (Read-Evaluate-Print Loop) window. The title bar says "REPL". The window contains the following text:

```
user=>
user=>
user=>
user=>
user=>
user=> "Hello"    "USCS..."
"Hello"
"USCS..."
user=>
user=>
user=>
user=>
```

The text shows two separate string literals being evaluated sequentially. The first string "Hello" is followed by a space and the second string "USCS...". Both strings are then printed on separate lines, demonstrating that each expression was evaluated individually.

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

Avaliando expressões aritméticas

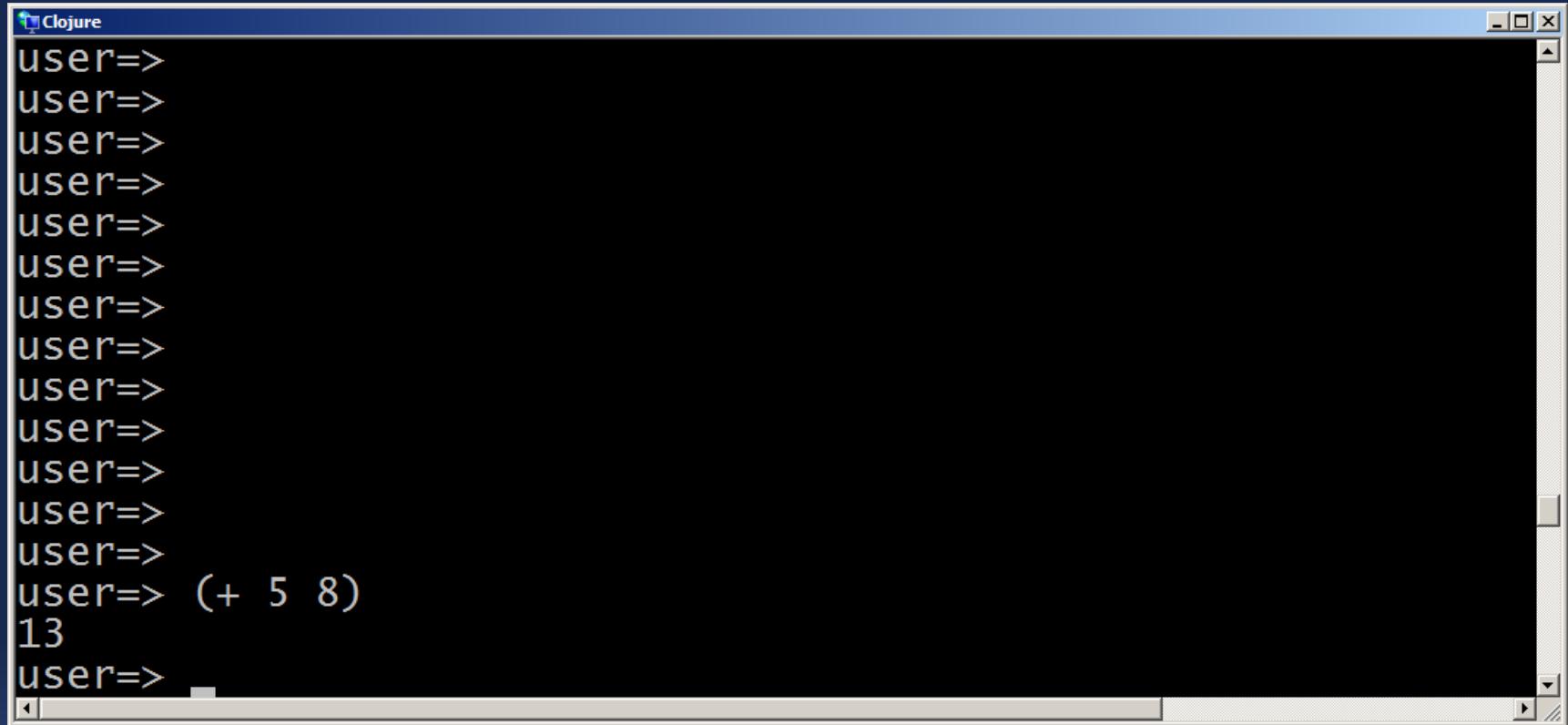


The screenshot shows a Windows-style application window titled "Clojure". Inside, the Clojure REPL is running. The user has typed the expression "5 + 8" and pressed Enter. The REPL has evaluated the first argument "5" and returned it. The second argument "8" is shown as a reference to a function object: "#object[clojure.core\$PLUS_ 0x294a6b8e "clojure.core\$PLUS_@28"]". The user prompt "user=>" is visible at the bottom.

```
Clojure
user=>
user=> 5 + 8
5
#object[clojure.core$PLUS_ 0x294a6b8e "clojure.core$PLUS_@28"]
user=>
```

- ✓ 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**).

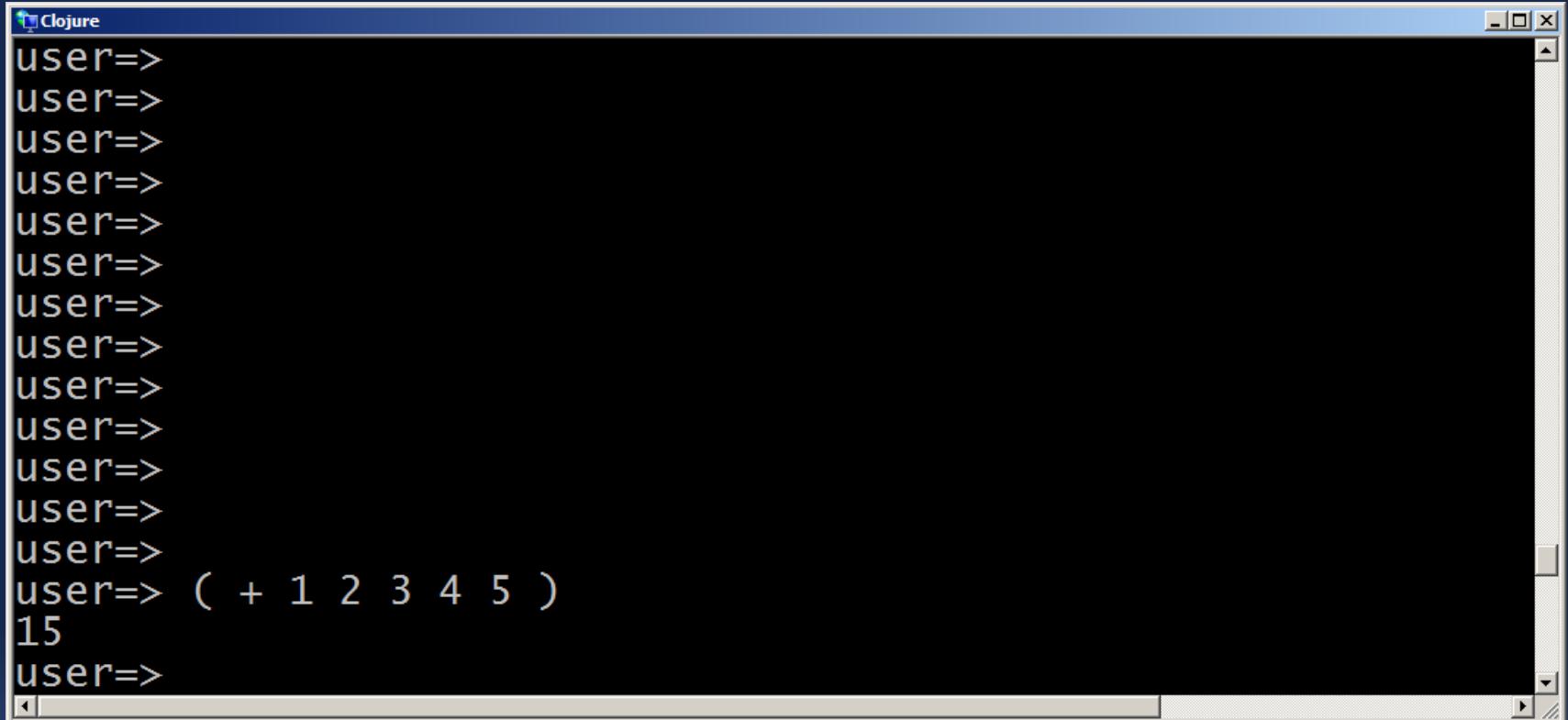
Avaliando expressões aritméticas



The screenshot shows a terminal window titled "Clojure". The user has entered the expression `(+ 5 8)`, which is evaluated to the result `13`. The window has a standard Windows-style title bar and scroll bars.

```
Clojure
user=>
user=> (+ 5 8)
13
user=>
```

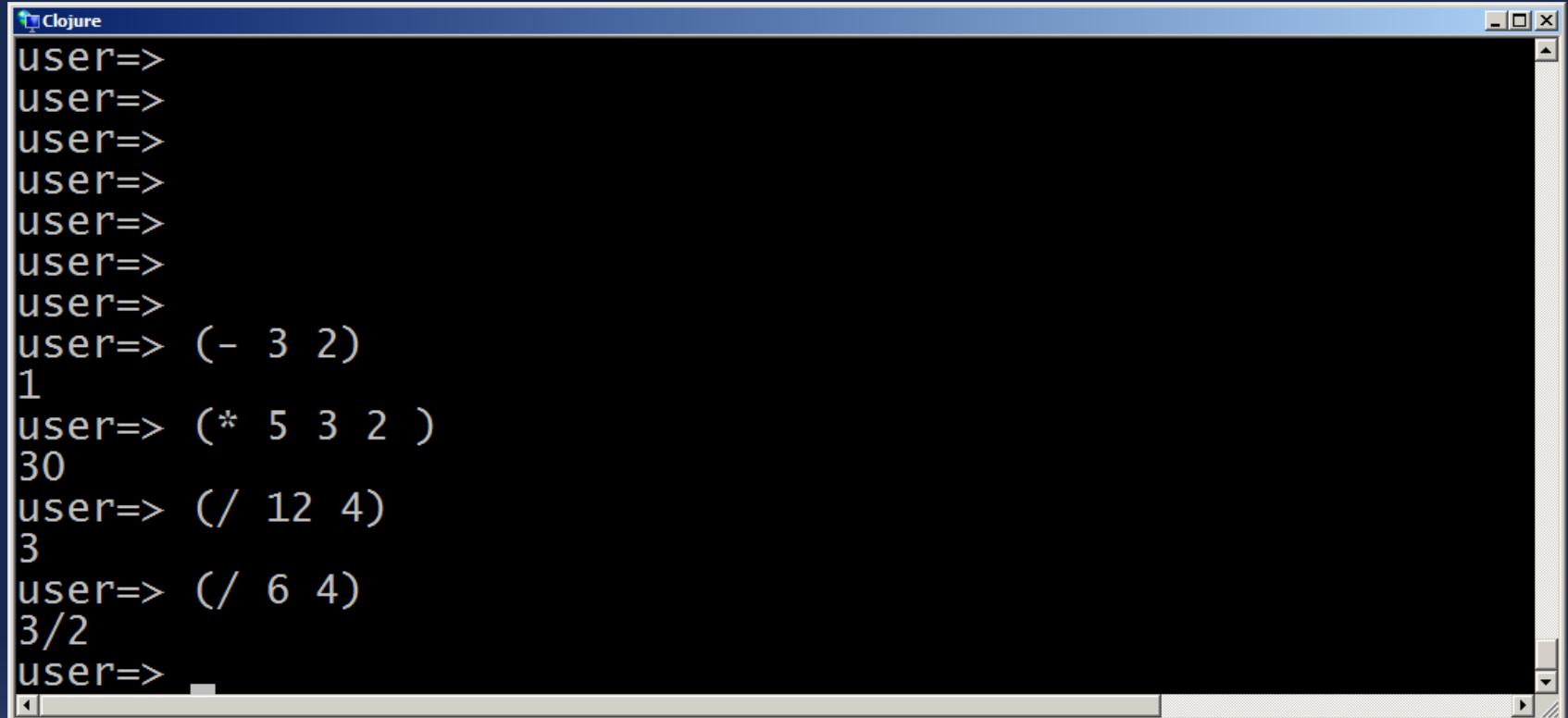
Avaliando expressões aritméticas



The screenshot shows a Windows-style window titled "Clojure". Inside, the Clojure REPL is running. The user has typed the expression "(+ 1 2 3 4 5)" and the system has evaluated it to the result "15". The REPL prompt "user=>" appears at the end of each line of input.

```
Clojure
user=>
user=> ( + 1 2 3 4 5 )
15
user=>
```

Avaliando expressões aritméticas



The screenshot shows a Windows-style application window titled "Clojure". Inside, a Clojure REPL session is running, displaying the following interactions:

```
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> (- 3 2)
1
user=> (* 5 3 2 )
30
user=> (/ 12 4)
3
user=> (/ 6 4)
3/2
user=>
```



Função println

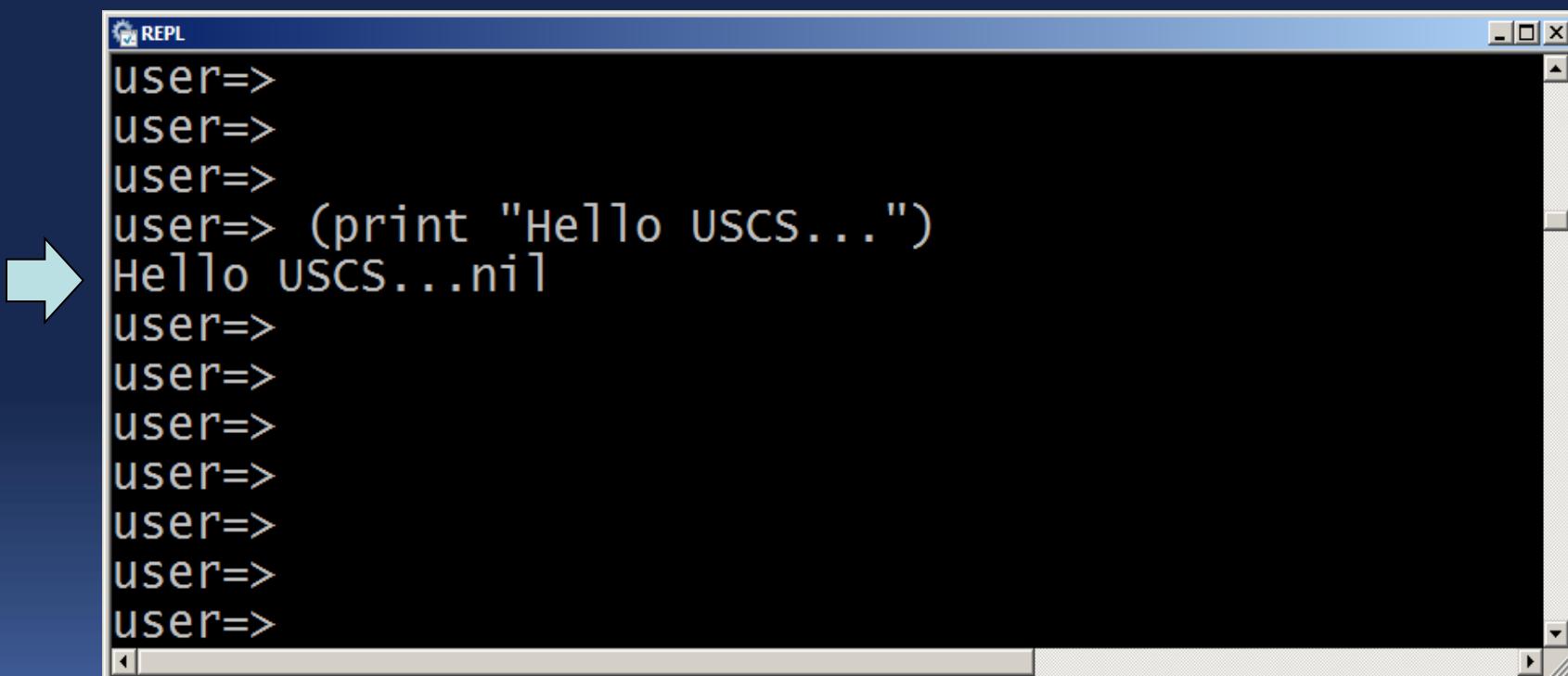


```
REPL
user=>
user=>
user=>
user=> (println "Hello USCS...")
Hello USCS...
nil
user=>
user=>
user=>
user=>
user=>
user=>
```

- ✓ 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;



The screenshot shows a Clojure REPL window with the title bar "REPL". The window contains the following text:

```
user=>
user=>
user=>
user=> (print "Hello USCS...")
Hello USCS...nil
user=>
user=>
user=>
user=>
user=>
user=>
user=>
```

A large green arrow points from the left towards the "Hello USCS..." line.



Funções nested

A screenshot of a Clojure REPL window titled "Clojure". The window shows the following interaction:

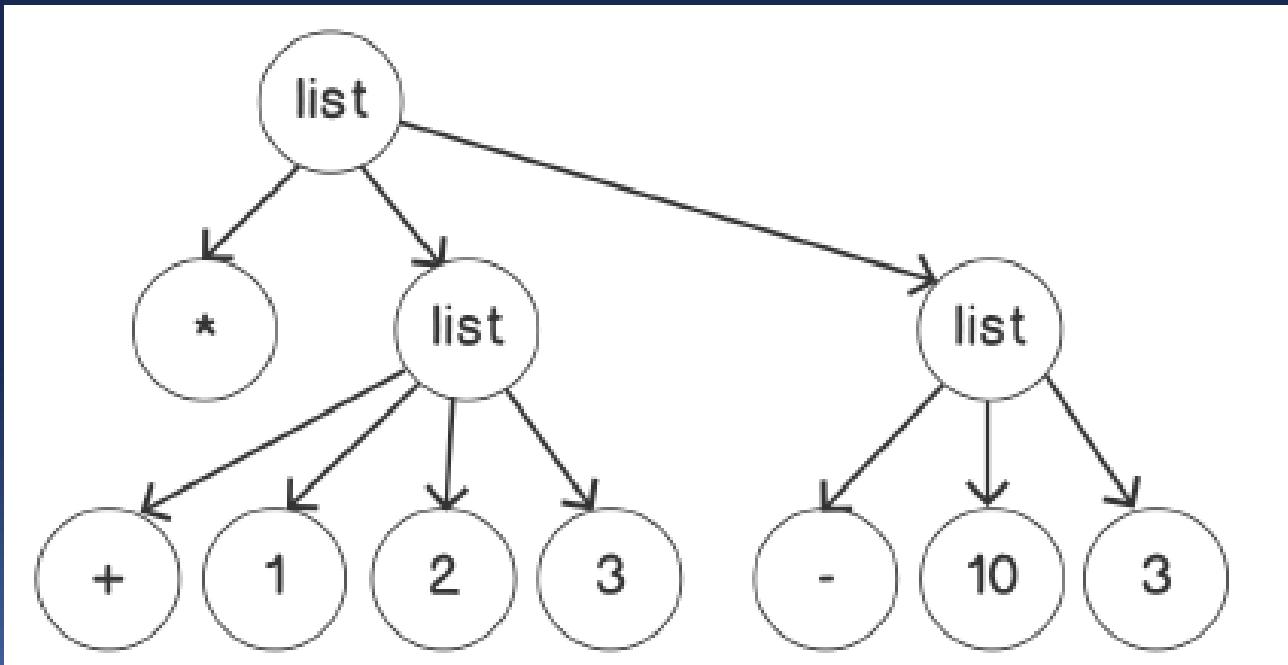
```
Clojure 1.10.2-master-SNAPSHOT
user=> (+ (* 3 2) (- 5 3))
8
user=>
```

The window has a dark background and a light blue header bar. The code is entered at the "user=>" prompt, and the result "8" is displayed. The window is set against a dark blue background of the presentation slide.

Expressões Simbólicas (s-expression)

- ✓ Podem ser visualizadas em uma árvore

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





Exit REPL (System/exit 0)

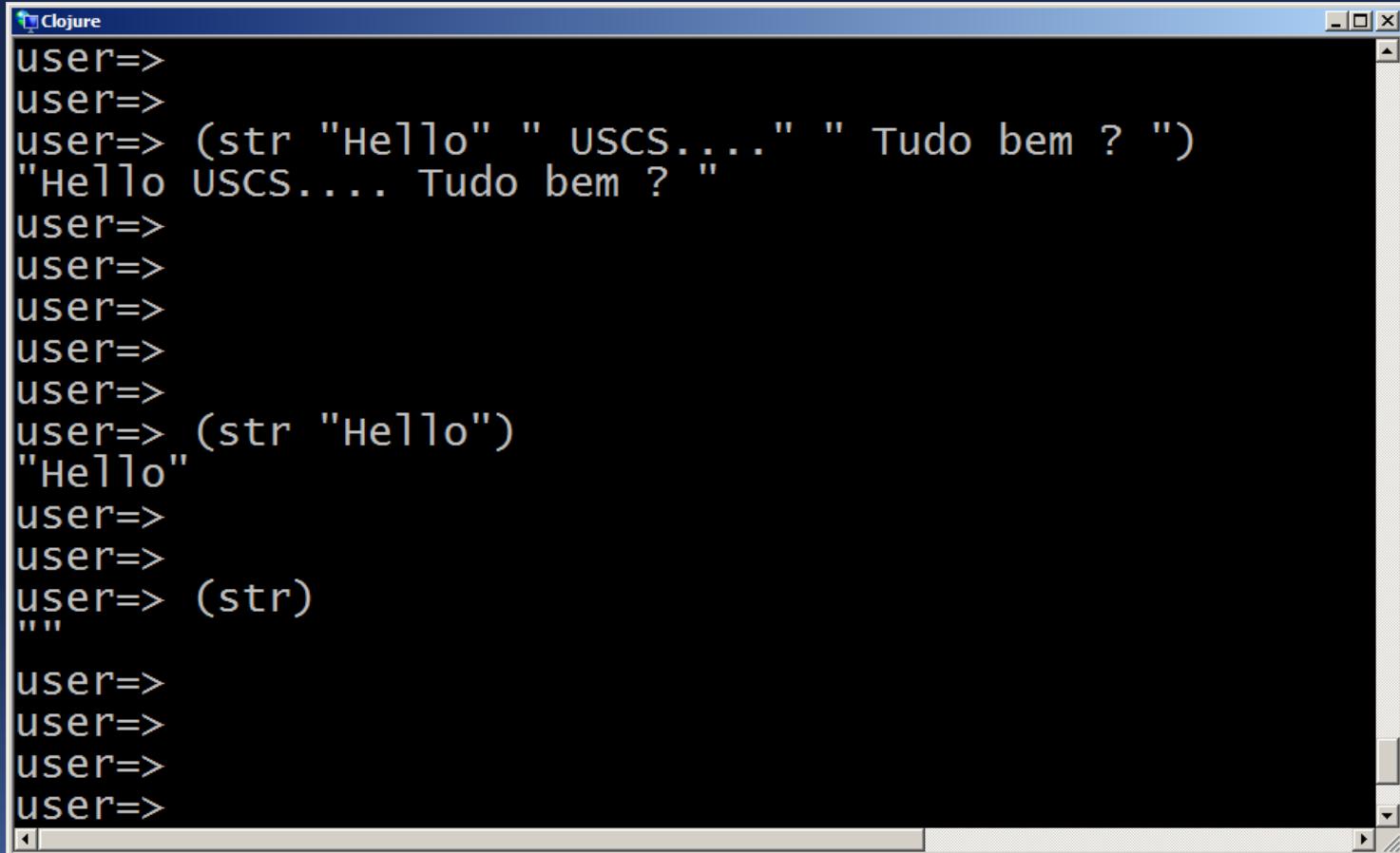
A screenshot of a terminal window titled "Clojure". The title bar also shows "user=>". The window contains the text: "Clojure 1.10.2-master-SNAPSHOT" and "user=> (System/exit 0) -". The rest of the window is blank black space, indicating the program has exited.

```
Clojure 1.10.2-master-SNAPSHOT
user=> (System/exit 0) -
```



Concatenando Strings – str

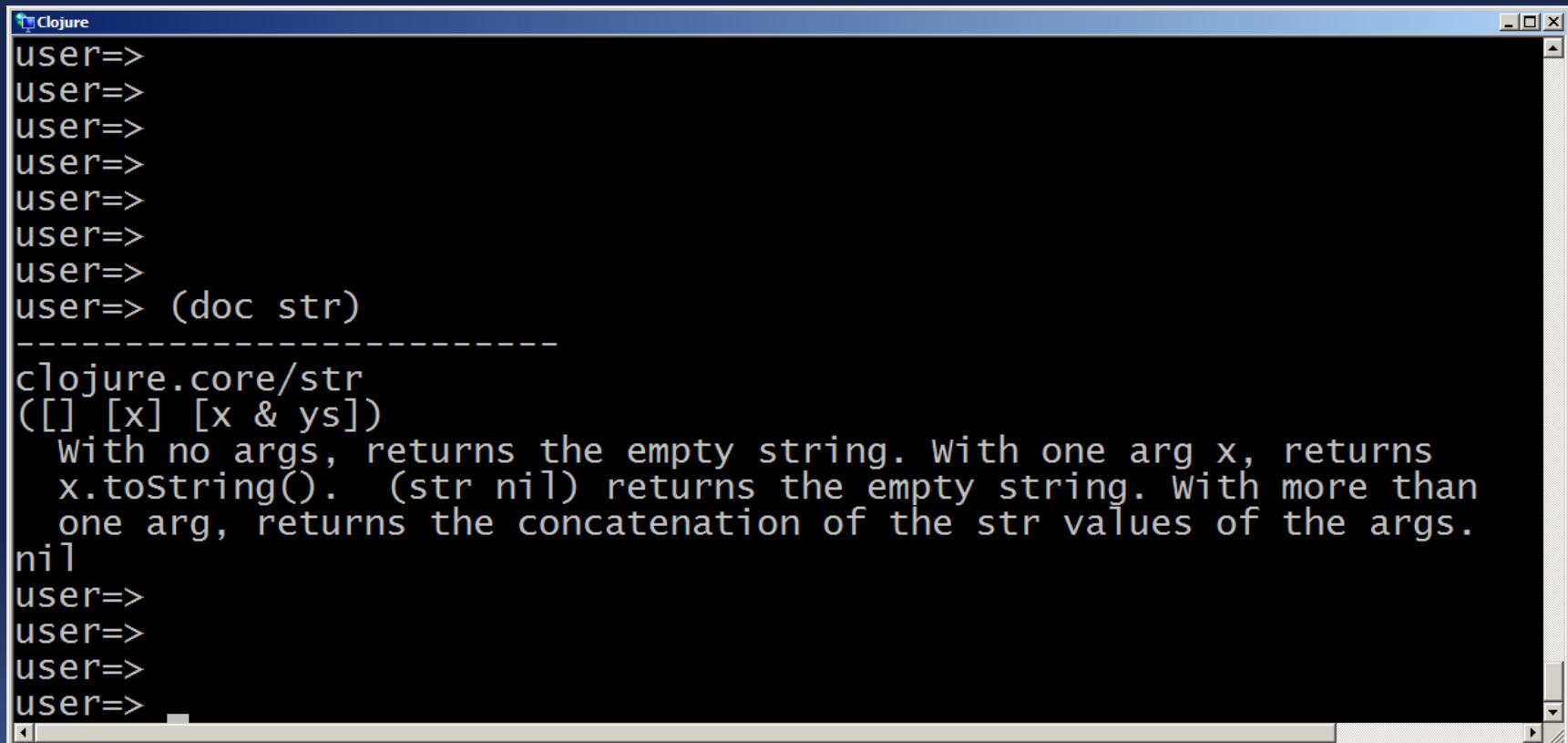
- ✓ A função **str** sem parâmetros retorna o String **nulo** ("")
- ✓ Com um argumento **x** retorna **x.toString()**:
- ✓ Com **mais de um argumento** **x...y** retorna a **concatenação** desses argumentos !



```
Clojure
user=>
user=>
user=> (str "Hello" " USCS...." " Tudo bem ? ")
"Hello USCS.... Tudo bem ? "
user=>
user=>
user=>
user=>
user=>
user=>
user=> (str "Hello")
"Hello"
user=>
user=>
user=> (str)
""
user=>
user=>
user=>
user=>
```

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**:



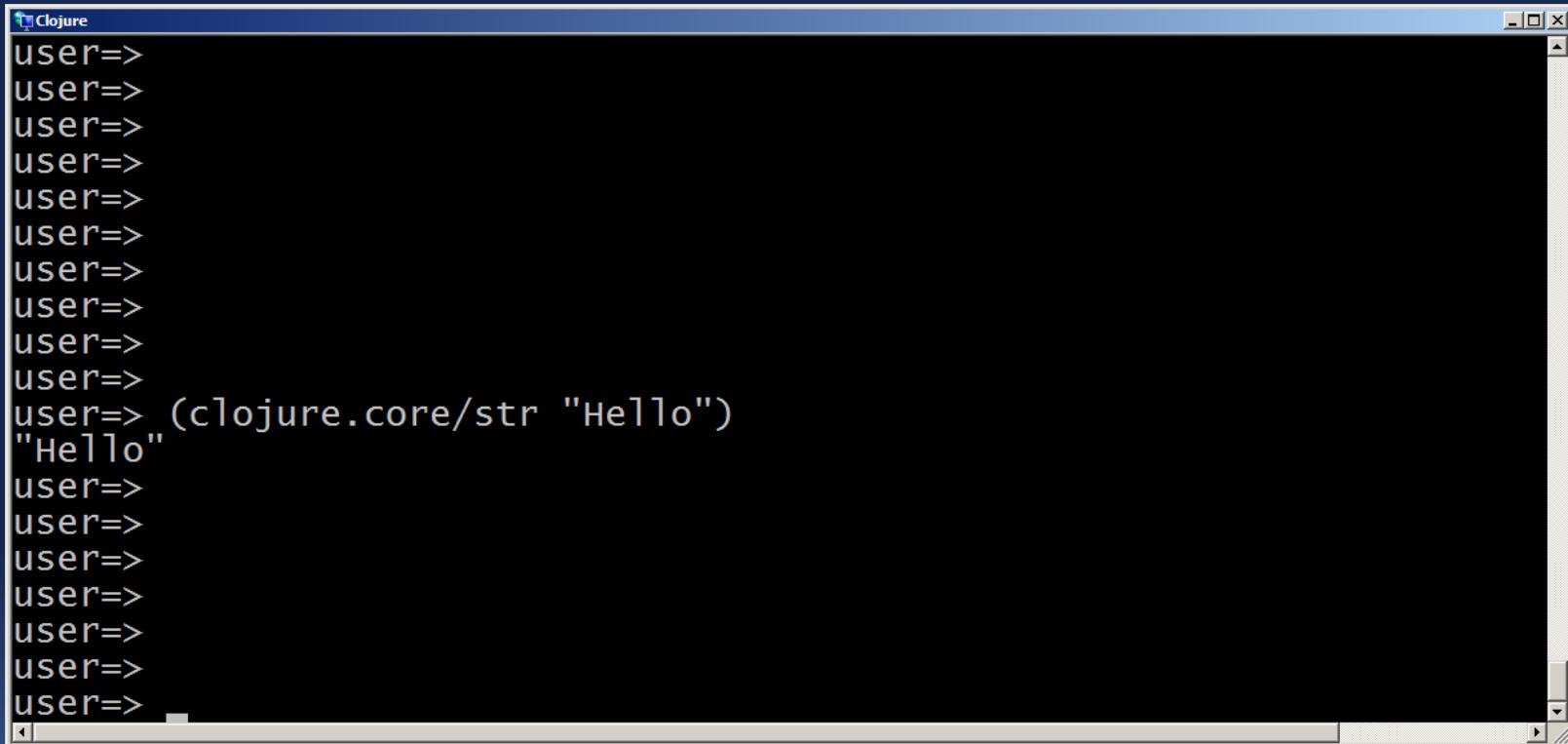
The screenshot shows a Clojure REPL window titled "Clojure". The user has entered the command `(doc str)`. The output shows the documentation for the `clojure.core/str` function, which states:

```
clojure.core/str
([] [x] [x & ys])
With no args, returns the empty string. With one arg x, returns
x.toString(). (str nil) returns the empty string. With more than
one arg, returns the concatenation of the str values of the args.
```

The user then types `nil` and the REPL returns `nil`.

clojure.core

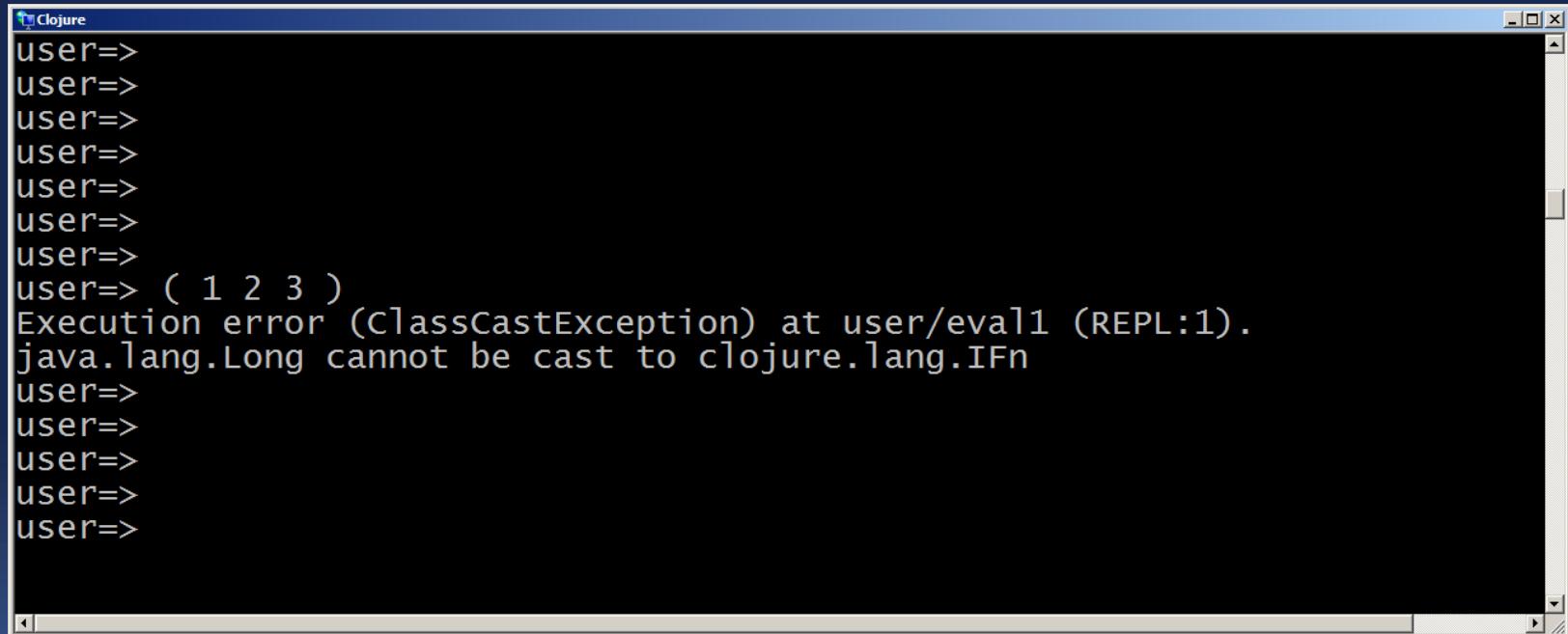
- ✓ Tudo o que está definido em `clojure.core` estará disponível ao seu **namespace** corrente por **default**, dispensando-se assim de se explicitamente requerê-la;
- ✓ Por exemplo: **(str "hello")** ao invés de **(clojure.core/str "hello")**

A screenshot of a Windows-style application window titled "Clojure". The window contains a black terminal-like interface. It shows several lines of text starting with "user=>" followed by the expression "(clojure.core/str "Hello")" and its result "Hello". This demonstrates that the core functions are available by default in the user namespace.

```
user=>
user=> (clojure.core/str "Hello")
>Hello
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
```

Listas

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



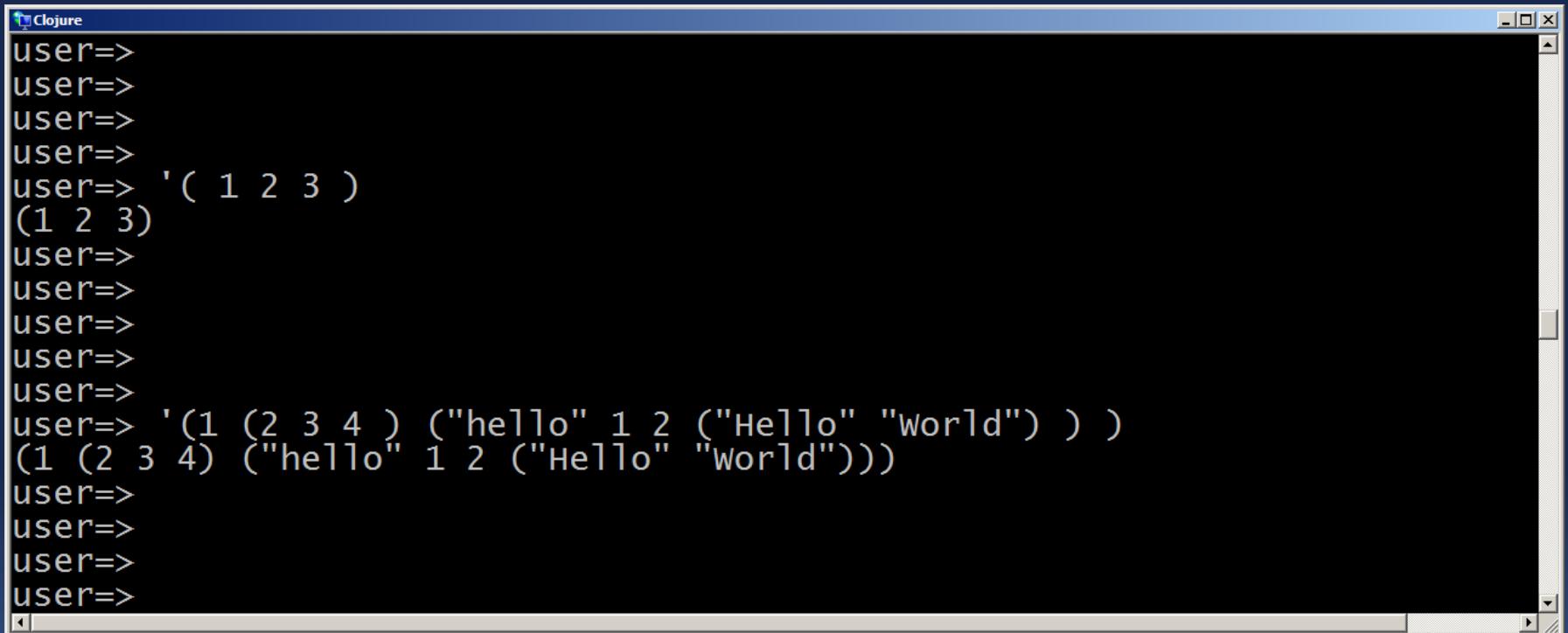
The screenshot shows a Windows-style window titled "Clojure". Inside, the Clojure REPL is running. The user has typed a list `'(1 2 3)` and received an error message: "Execution error (ClassCastException) at user/eval1 (REPL:1). java.lang.Long cannot be cast to clojure.lang.IFn". This demonstrates that Clojure does not allow lists to be used as simple data structures for numbers.

```
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> ( 1 2 3 )
Execution error (ClassCastException) at user/eval1 (REPL:1).
java.lang.Long cannot be cast to clojure.lang.IFn
user=>
user=>
user=>
user=>
user=>
```



Listas como dados

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

A screenshot of a Clojure REPL window titled "Clojure". The window shows several lines of Clojure code being entered and evaluated. The user has entered several list literals, some of which are wrapped in single quotes to prevent them from being interpreted as functions. The output shows the evaluated forms, such as '(1 2 3) and (1 (2 3 4) ("hello" 1 2 ("Hello" "world"))).

```
Clojure
user=>
user=>
user=>
user=>
user=> '( 1 2 3 )
(1 2 3)
user=>
user=>
user=>
user=>
user=>
user=> '(1 (2 3 4) ("hello" 1 2 ("Hello" "world") ) )
(1 (2 3 4) ("hello" 1 2 ("Hello" "world")))
user=>
user=>
user=>
user=>
```

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`).

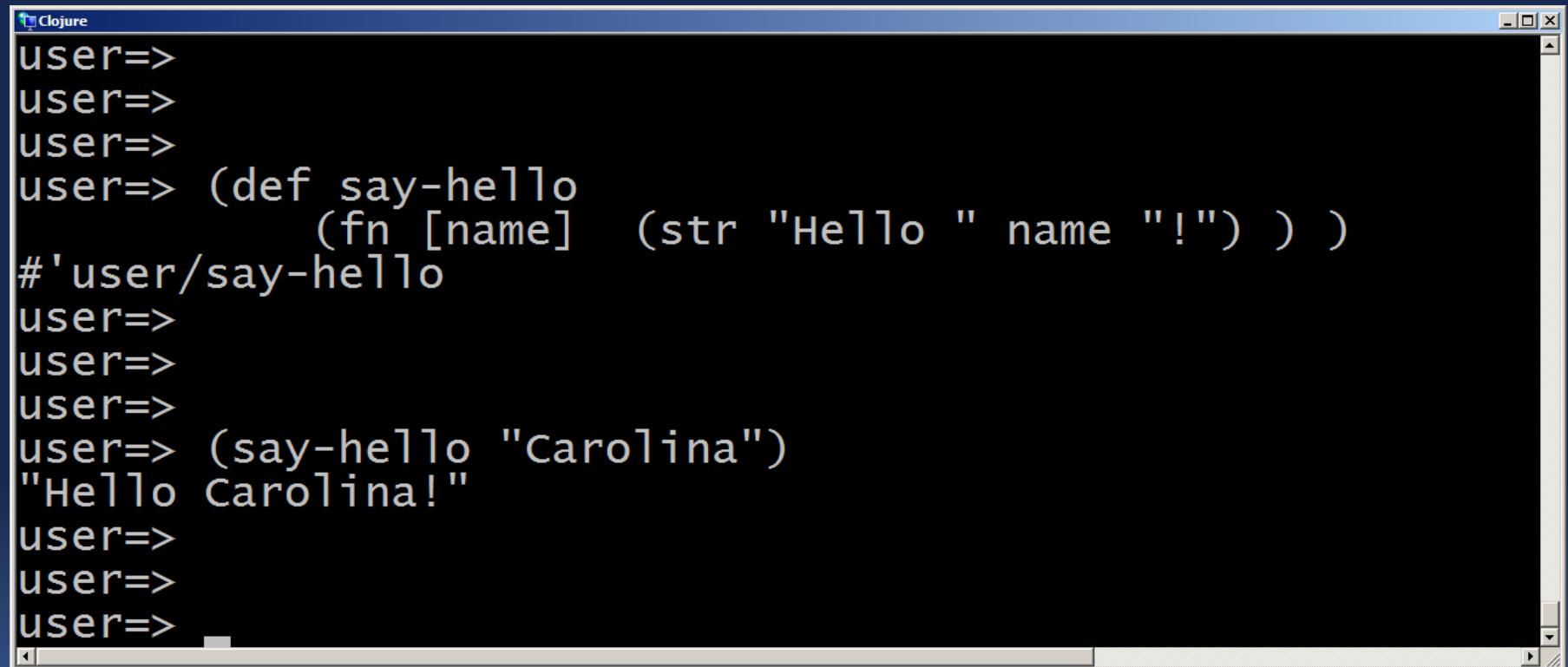
Macro - def

```
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:



The screenshot shows a Windows-style window titled "Clojure". Inside, a Clojure REPL session is running. The user has defined a macro named "say-hello" which takes a single argument "name" and returns a string concatenating "Hello " and the name followed by an exclamation mark. The user then calls this macro with the argument "Carolina", resulting in the output "Hello Carolina!".

```
Clojure
user=>
user=>
user=>
user=> (def say-hello
          (fn [name] (str "Hello " name "!") ) )
#'user/say-hello
user=>
user=>
user=>
user=> (say-hello "Carolina")
"Hello Carolina!"
user=>
user=>
user=>
```

Macro - fn

```
Clojure
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=>
user=> (ola-mundo)
Hello World
nil
user=>
```

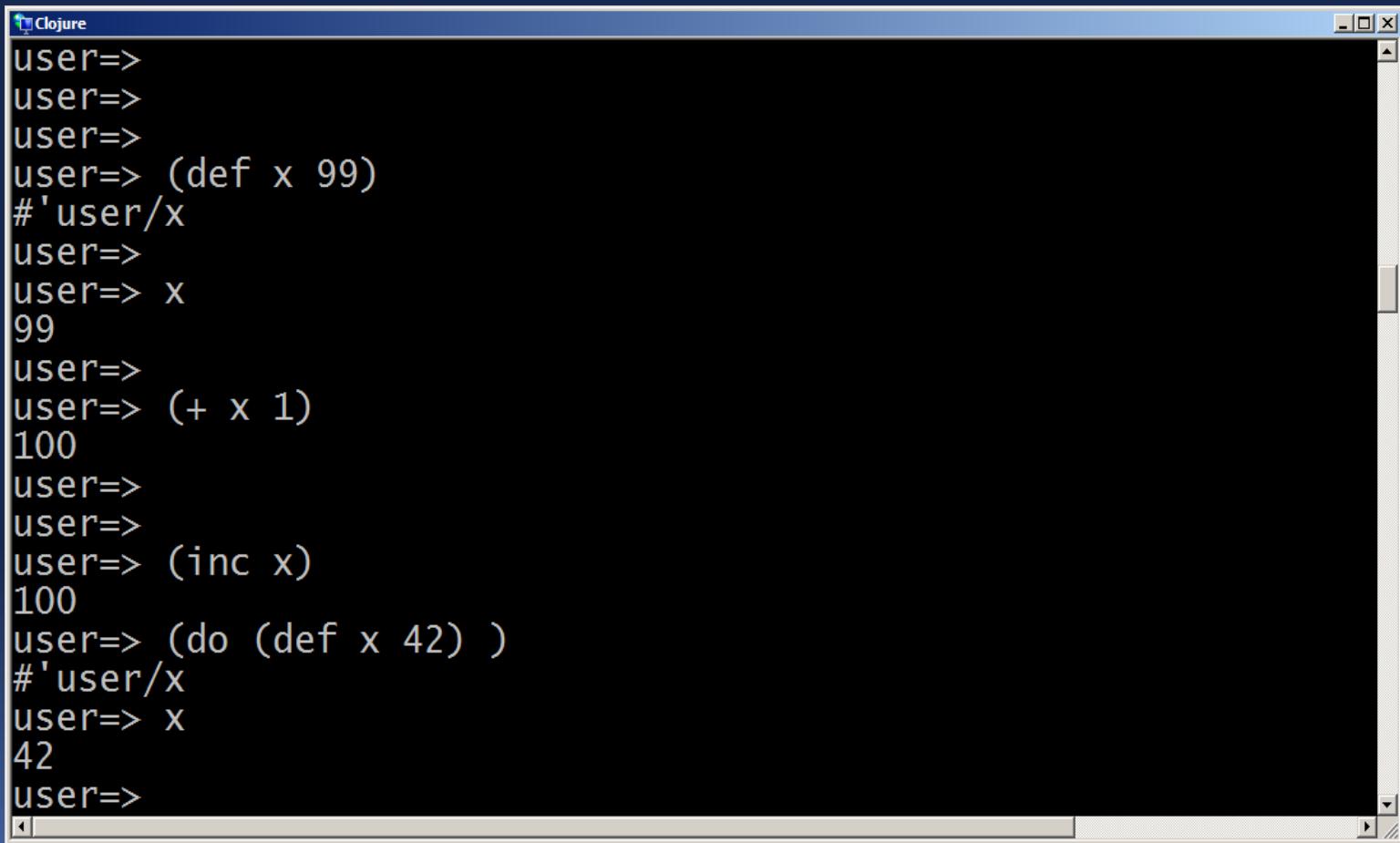


Binding

- ✓ 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**;

A screenshot of a Windows-style application window titled "Clojure". The window contains a text area showing a Clojure REPL session. The session starts with several "user=>" prompts, followed by the definition of a symbol "x" with the value 99. Subsequent prompts show the value of "x" being used in expressions like "(+ x 1)" and "(inc x)", both resulting in 100. Finally, a new binding is created for "x" with the value 42, which is then used in another expression. The text area has a black background and white text, with scroll bars on the right side.

```
user=>
user=>
user=>
user=> (def x 99)
#'user/x
user=>
user=> x
99
user=>
user=> (+ x 1)
100
user=>
user=>
user=> (inc x)
100
user=> (do (def x 42) )
#'user/x
user=> x
42
user=>
```

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

```
Clojure
user=>
user=> (defn quadrado [x] (* x x) )
#'user/quadrado
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> (quadrado 5)
25
user=>
user=>
```





Estruturas de Dados



Estruturas de Dados

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



Coleções

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



Interoperabilidade com Java

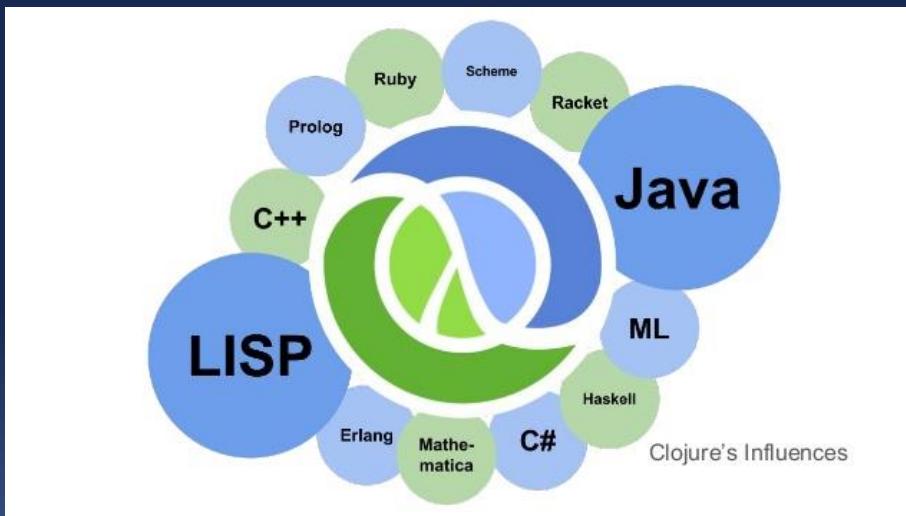


Interoperabilidade com Java

- ✓ 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 Garbage Collection, threading, **concorrência**, operações de I/O que são largamente já testadas e consolidadas, são incorporadas ao **Clojure**.

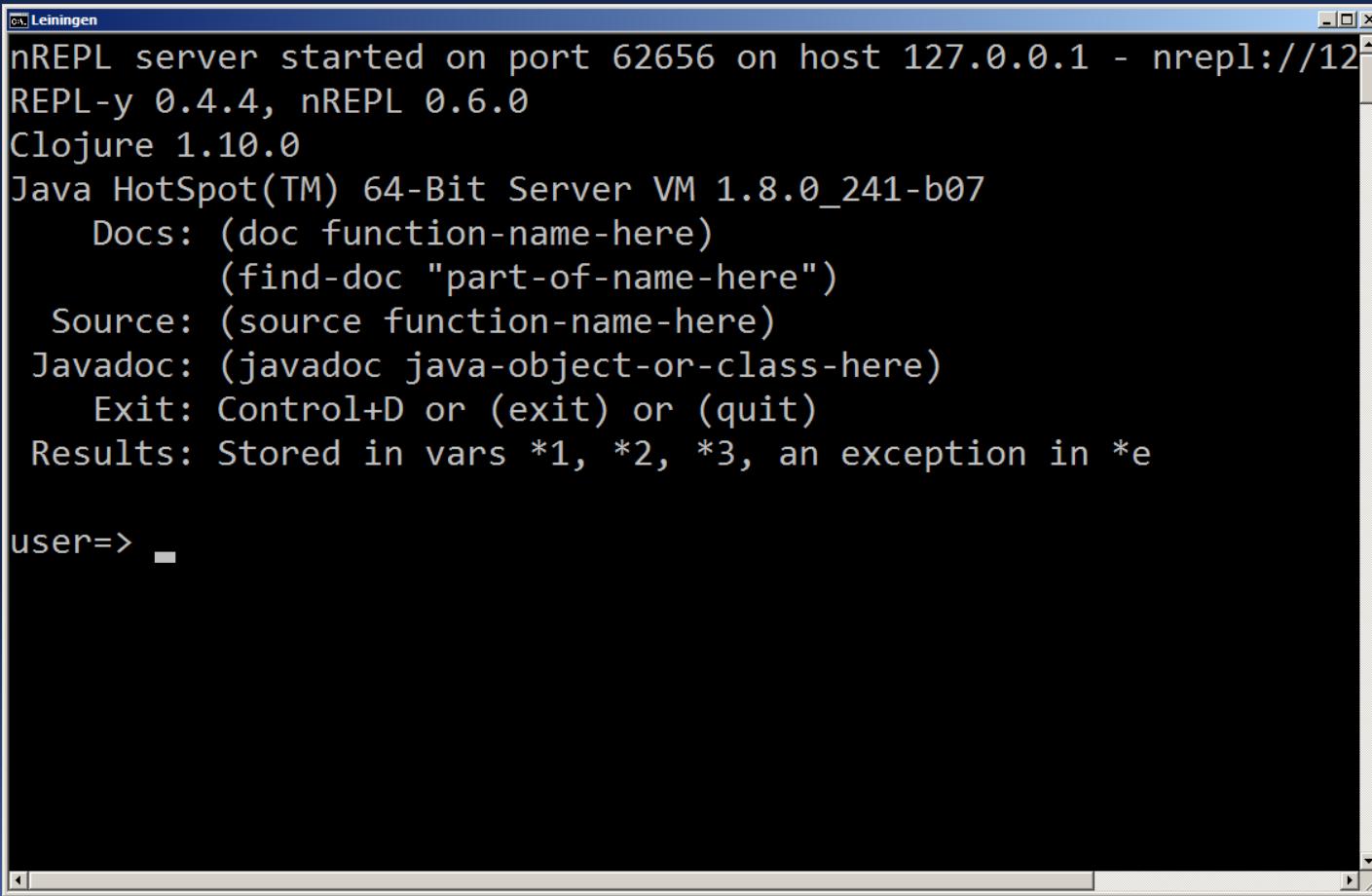


Usando Java em Clojure

- ✓ 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



A screenshot of a terminal window titled "Leiningen". The window displays the startup message for an nREPL server, including the host (127.0.0.1), port (62656), and version information (REPL-y 0.4.4, nREPL 0.6.0, Clojure 1.10.0, Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07). Below this, several Java documentation commands are listed: Docs, Source, Javadoc, Exit, and Results. At the bottom, the prompt "user=>" is visible.

```
nREPL server started on port 62656 on host 127.0.0.1 - nrepl://127.0.0.1:62656
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
Docs: (doc function-name-here)
      (find-doc "part-of-name-here")
Source: (source function-name-here)
Javadoc: (javadoc java-object-or-class-here)
Exit: Control+D or (exit) or (quit)
Results: Stored in vars *1, *2, *3, an exception in *e

user=> ■
```

Importando a classe Java BigDecimal (import ...)

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



```
Leiningen
nREPL server started on port 62909 on host 127.0.0.1 - nrepl
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
         (find-doc "part-of-name-here")
  Source: (source function-name-here)
  Javadoc: (javadoc java-object-or-class-here)
    Exit: Control+D or (exit) or (quit)
  Results: Stored in vars *1, *2, *3, an exception in *e

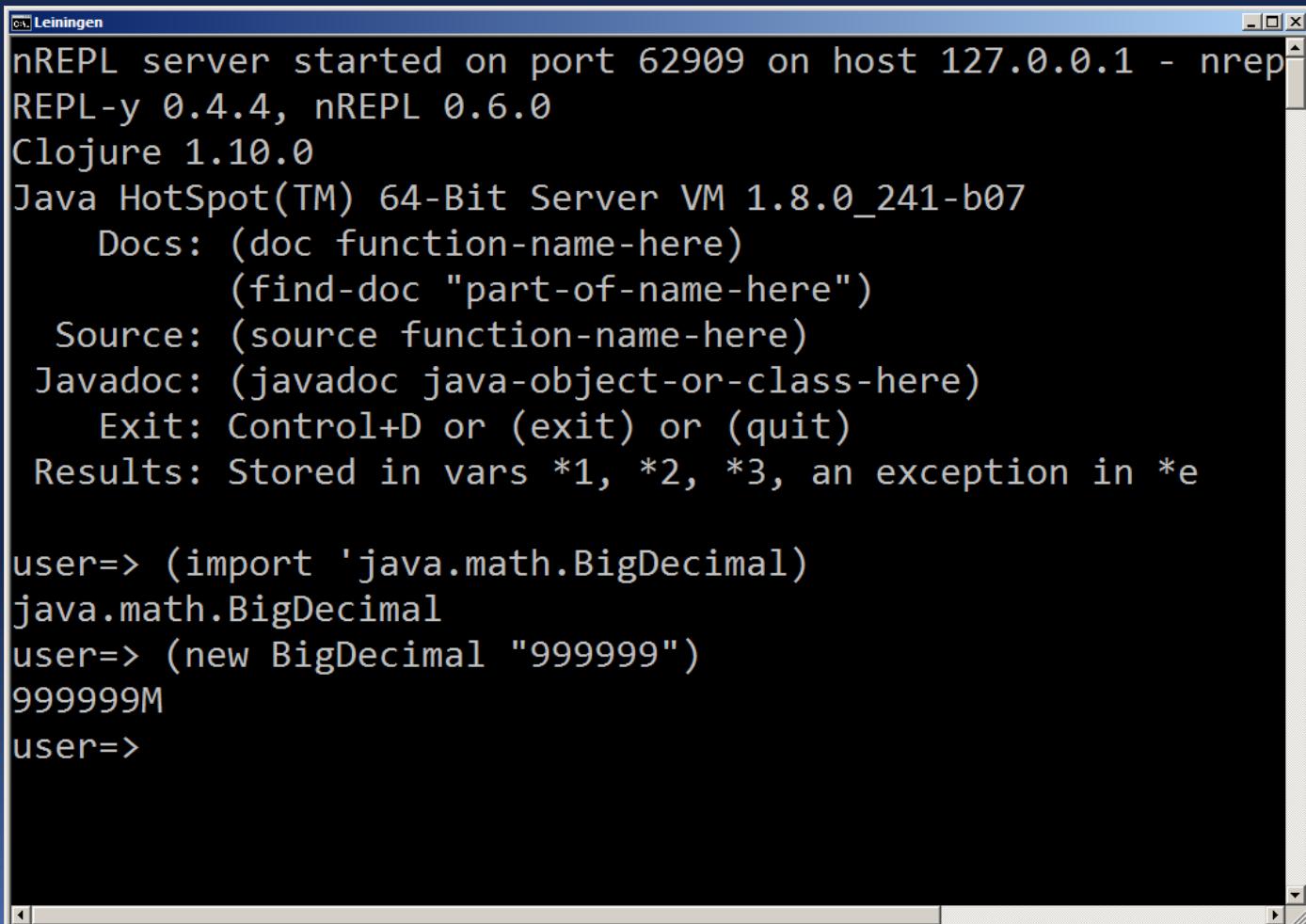
user=> (import 'java.math.BigDecimal)
java.math.BigDecimal
user=>
```

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")



The screenshot shows a terminal window titled 'C:\ Leiningen' displaying the output of a Clojure REPL session. The session starts with the nREPL server information and Clojure version. It then provides documentation and source code lookup instructions. The user enters a command to import the java.math.BigDecimal class, followed by a call to its constructor with the string "999999". The resulting value is printed as 999999M, indicating it is a BigInteger object.

```
nREPL server started on port 62909 on host 127.0.0.1 - nrepl
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
         (find-doc "part-of-name-here")
  Source: (source function-name-here)
Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
Results: Stored in vars *1, *2, *3, an exception in *e

user=> (import 'java.math.BigDecimal)
java.math.BigDecimal
user=> (new BigDecimal "999999")
999999M
user=>
```

Importando a classe Java BigDecimal

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

```
(def grande-numero (new BigDecimal "999999"))
```



```
Leiningen
user=> (import 'java.math.BigDecimal)
java.math.BigDecimal
user=> (new BigDecimal "999999")
999999M
user=> (def grande-numero (new BigDecimal "999999") )
#'user/grande-numero
user=> grande-numero
999999M
user=>
user=>
user=>
```

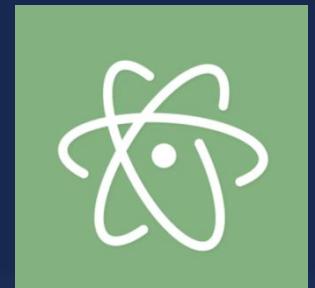


Desenvolvimento de API's em Clojure





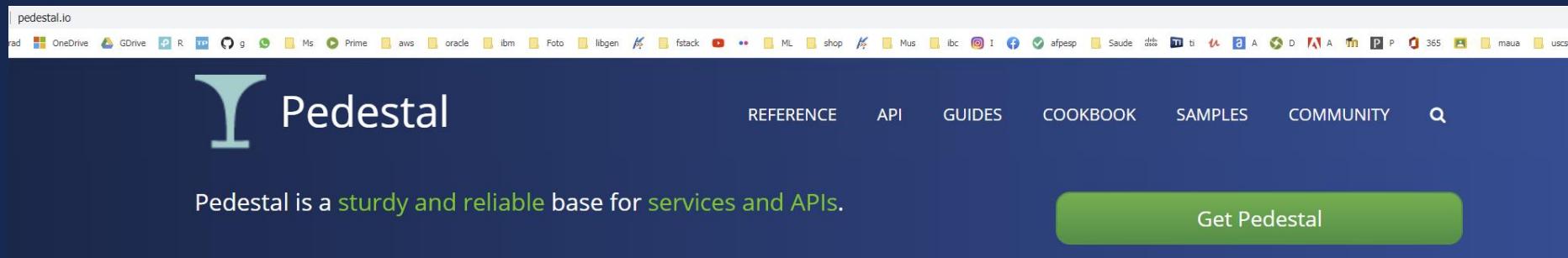
Criação de API's Rest em Clojure com o Framework Pedestal



Setup do Ambiente

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

<http://pedestal.io/>



The screenshot shows the official website for Pedestal. At the top, there's a navigation bar with links to REFERENCE, API, GUIDES, COOKBOOK, SAMPLES, and COMMUNITY, along with a search icon. Below the navigation, a large green button says "Get Pedestal". The main content area features a large "Pedestal" logo and the text: "Pedestal is a sturdy and reliable base for services and APIs." There's also a "Get Pedestal" button.

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 ?

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

 Pedestal

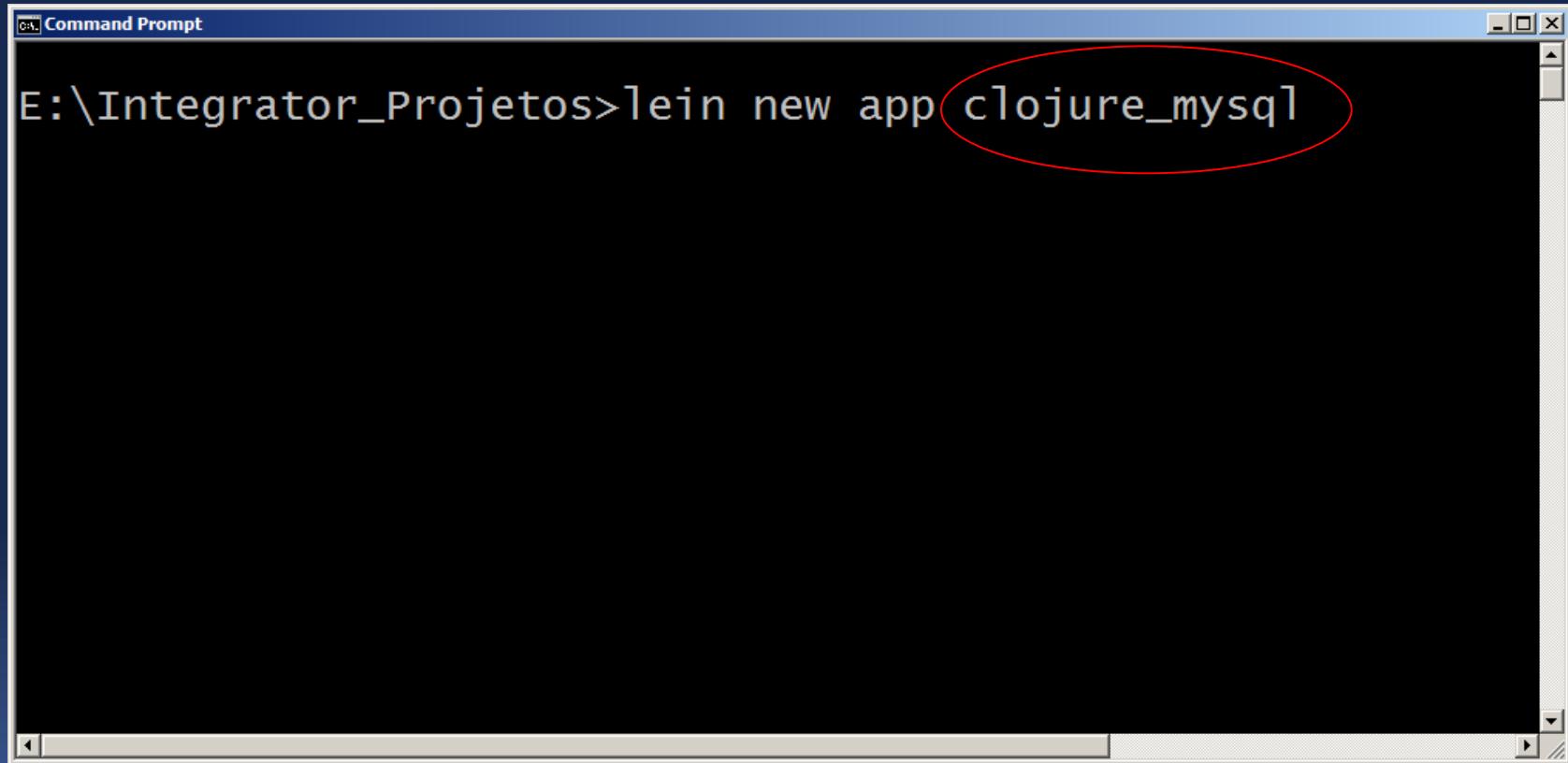


Acessando MySQL com projeto Leiningen



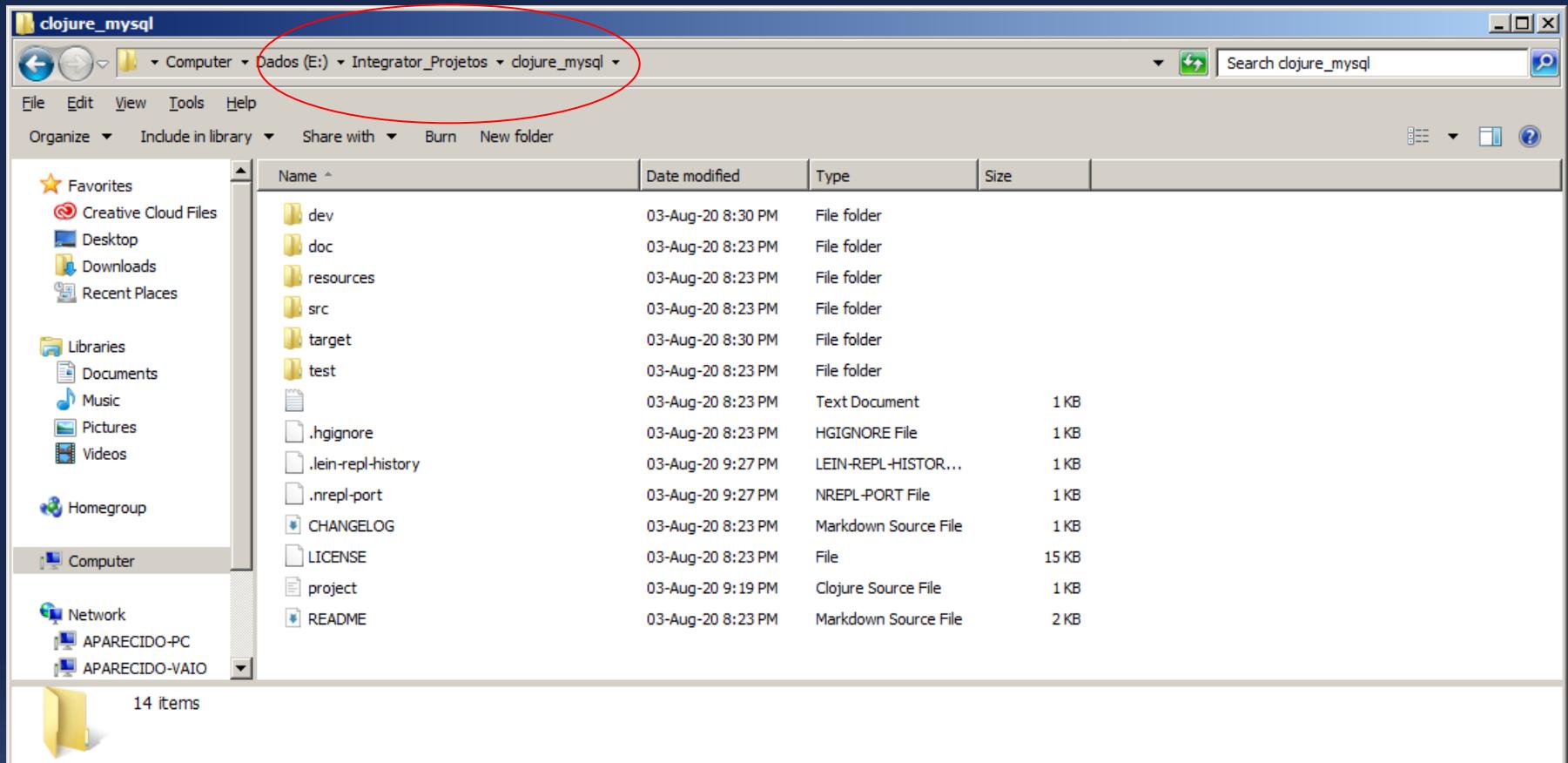
Recursos

④ Criando o projeto com Lein



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the command line path "E:\Integrator_Projetos>" followed by the command "lein new app clojure_mysql". The text "clojure_mysql" is highlighted with a red oval. The window has a standard Windows title bar and scroll bars.

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"] ] ←

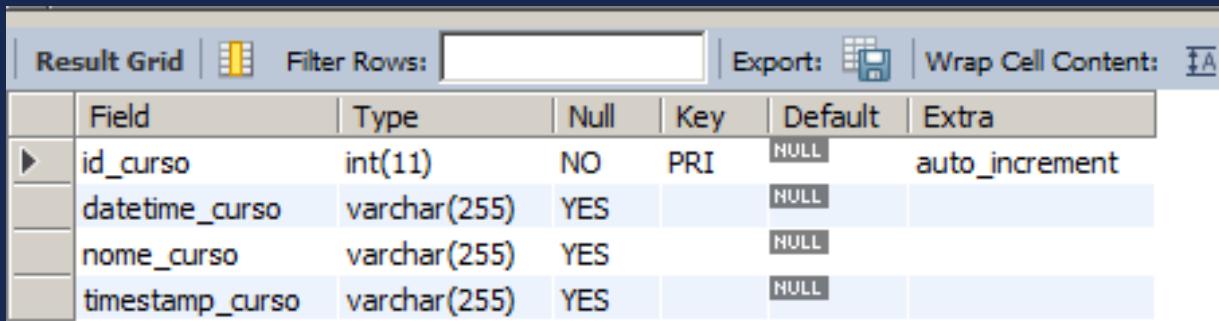
  :jvm-opts ["-Dclojure.server.myrepl={:port,5555,:accept,clojure.core.server/repl}"]

  :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

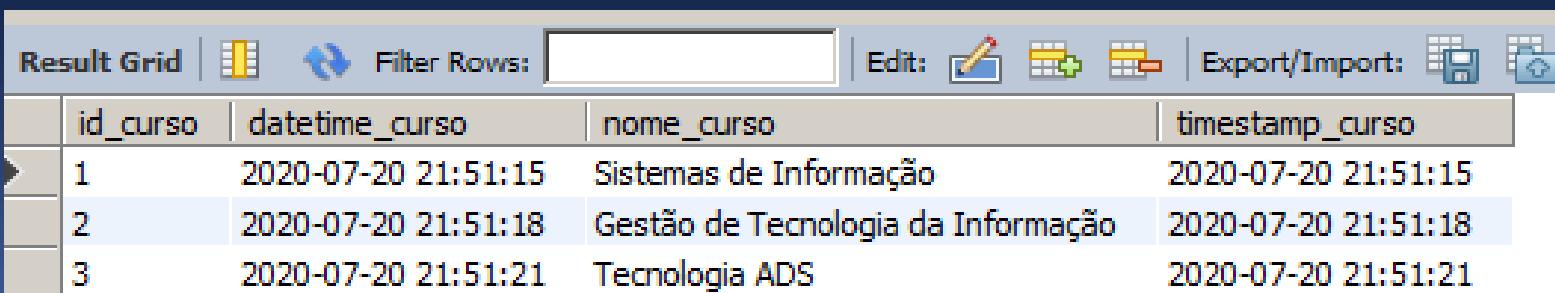
- Banco de Dados: scpe
- Porta: 3307
- Host: localhost
- Tabela: cursos

```
use scpe;  
  
describe curso;
```



	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;
```



	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"}))

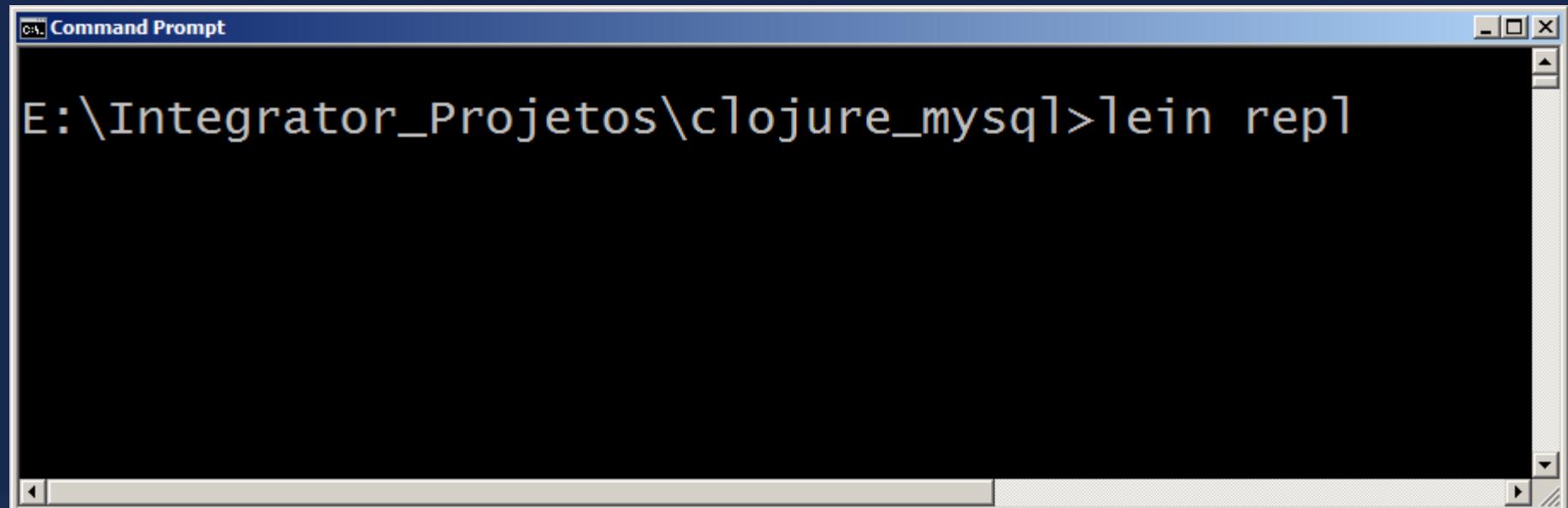
(def resp (jdbc/query db ["SELECT * FROM curso WHERE id_curso = 1"]))

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





Executando sob REPL - Chlorine



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window has a dark background and light-colored text. At the top, it shows the path "E:\Integrator_Projetos\clojure_mysql>". Below that, the command "lein repl" is typed and executed. The window includes standard Windows controls like minimize, maximize, and close buttons in the title bar, and scroll bars on the right side.

```
E:\Integrator_Projetos\clojure_mysql>lein repl
```





Servidor REPL ativo

```
Command Prompt - lein repl

E:\Integrator_Projetos\clojure_mysql>lein repl
nREPL server started on port 52158 on host 127.0.0.1 - nrepl://127.0.0.1:52158
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.1
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
         (find-doc "part-of-name-here")
  Source: (source function-name-here)
Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
Results: Stored in vars *1, *2, *3, an exception in *e

user=>
```





Conectando Atom Chlorine

The screenshot shows the Atom code editor with a Clojure file named `core.clj`. The code defines a namespace `(ns clojure-mysql.core (:gen-class))` and sets up a database connection with the following configuration:

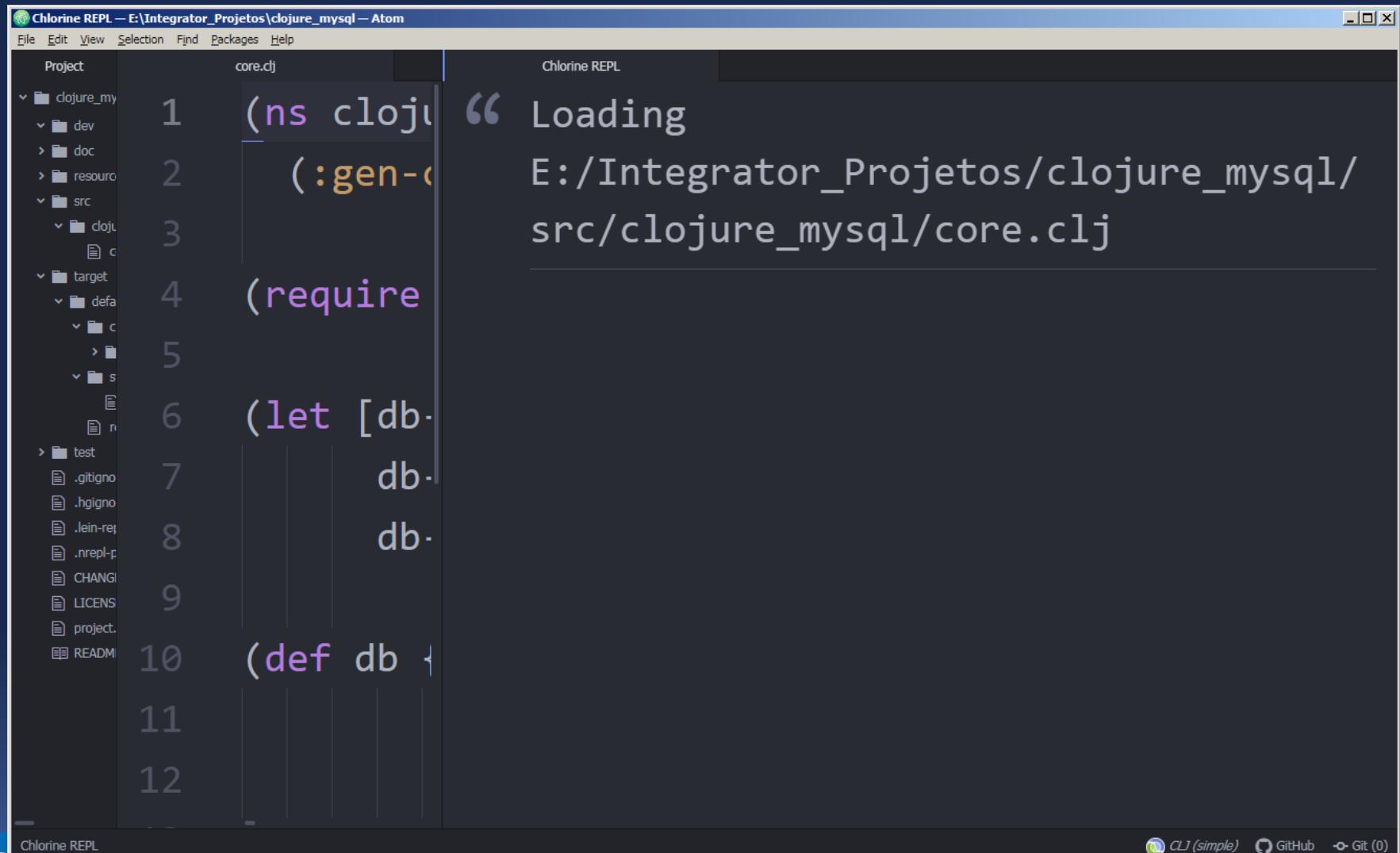
```
1 (ns clojure-mysql.core
2   (:gen-class))
3
4 (require '[clojure.java.jdbc :as jdbc])
5
6 (let [db-host "localhost"
7       db-port 3307
8       db-name "scope"]
9
10  (def db {:classname "com.mysql.jdbc.Driver"
11    :subprotocol "mysql"
12    :subname (str "jdbc:mysql://" db-host ":" db-port)
13    :user "root"
14    :password "root"}))
```

In the top right corner, there is a "Chlorine REPL" panel with a message: "Clojure socket REPL Connected".

Atom status bar at the bottom: CRLF, CLJ (simple), Windows 1252, Clojure, GitHub, Git (0).



Carregando código no REPL (Load-file)



The screenshot shows the Chlorine REPL interface within the Atom code editor. The left pane displays the project structure for 'clojure_mysql' with files like 'core.clj', 'dev.cljs', 'doc.cljs', 'resource.cljs', 'src.cljs', 'target.cljs', and various test and configuration files. The right pane shows the REPL output:

```
Chlorine REPL — E:\Integrator_Projetos\clojure_mysql — Atom
File Edit View Selection Find Packages Help
Project core.clj Chlorine REPL
1 (ns clojus “ Loading
2 (:gen-class)
3
4 (require
5
6 (let [db-
7 db-
8 db-
9
10 (def db {
11
12
```

The REPL is currently loading the file 'core.clj' from the path 'E:/Integrator_Projetos/clojure_mysql/src/clojure_mysql/core.clj'. The code being loaded includes namespace declarations, class generation, require statements, let bindings for database connections, and a def statement for a 'db' variable.

Executando sob REPL (Chlorine)

core.clj – E:\Integrator_Projetos\clojure_mysql – Atom

File Edit View Selection Find Packages Help

Project core.clj

```
9
10  (def db {:classname "com.mysql.jdbc.Driver"
11    :subprotocol "tcp"
12    :subname (str "jdbc:mysql://localhost:3306/"
13      :user "root"
14      :password "maua")
15
16  (jdbc/query db ["SELECT * FROM cursos WHERE id_curso = ?"])
17
18  (defn -main
19    "Código clojure para a aplicação"
20    [& args])
```

Chlorine REPL

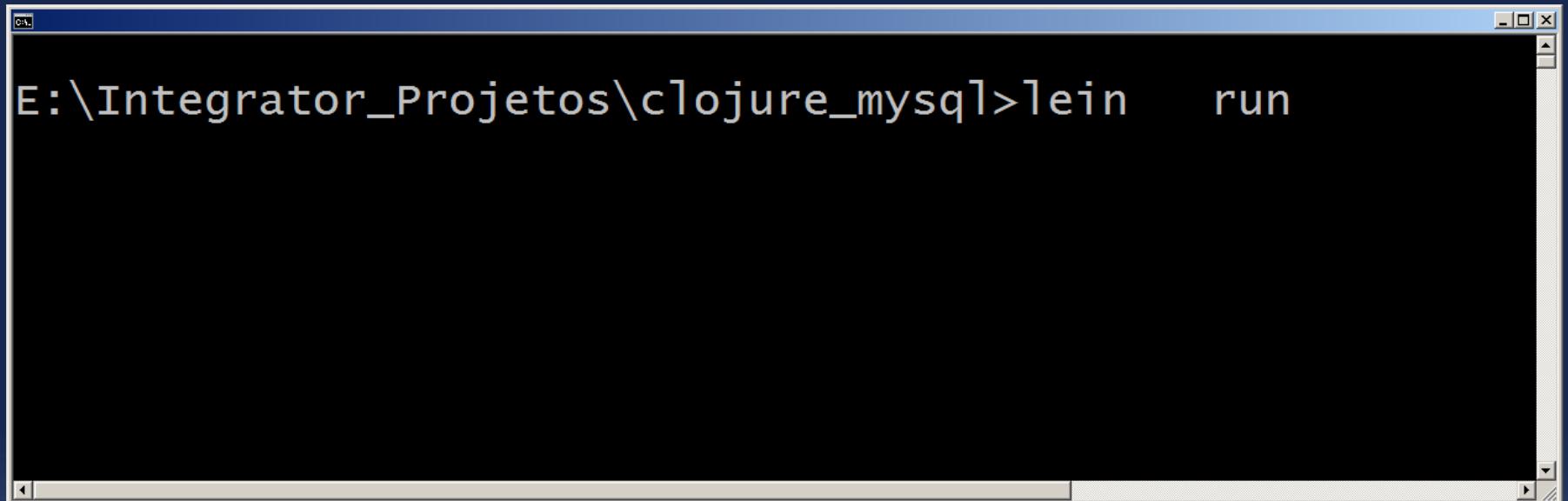
“ Loading
E:/Integrator_Projetos/clojure_mysql/src/clojure_mysql/core.clj
×{:id_curso 1, :datetim...”



CRLF CLJ (simple) Windows 1252 Clojure GitHub Git (0)

Executando – Linha de Comando REPL

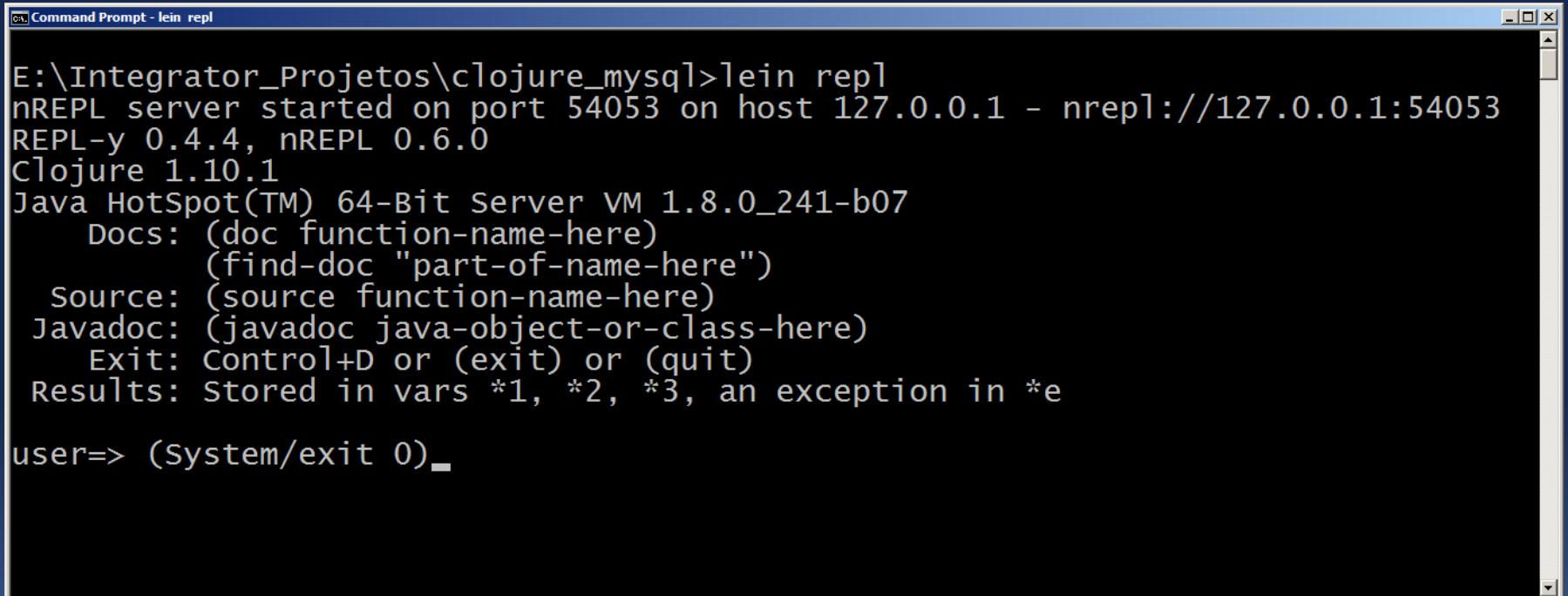
No diretório do projeto > lein run



A screenshot of a Windows command prompt window titled 'CMD'. The window shows the path 'E:\Integrator_Projetos\clojure_mysql>' followed by the command 'lein run'. The window has a standard Windows title bar and scroll bars on the right side.

Executando - Linha de Comando REPL

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



The screenshot shows a Windows Command Prompt window with the title "Command Prompt - lein repl". The window displays the following text:

```
E:\Integrator_Projetos\clojure_mysql>lein repl
nREPL server started on port 54053 on host 127.0.0.1 - nrepl://127.0.0.1:54053
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.1
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
        (find-doc "part-of-name-here")
  Source: (source function-name-here)
Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
Results: Stored in vars *1, *2, *3, an exception in *e

user=> (System/exit 0)
```

Executando – Linha de Comando REPL

```
C:\ 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  
rmação, :timestamp_curso 2020-07-20 21:51:15})  
E:\Integrator_Projetos\clojure_mysql>-
```



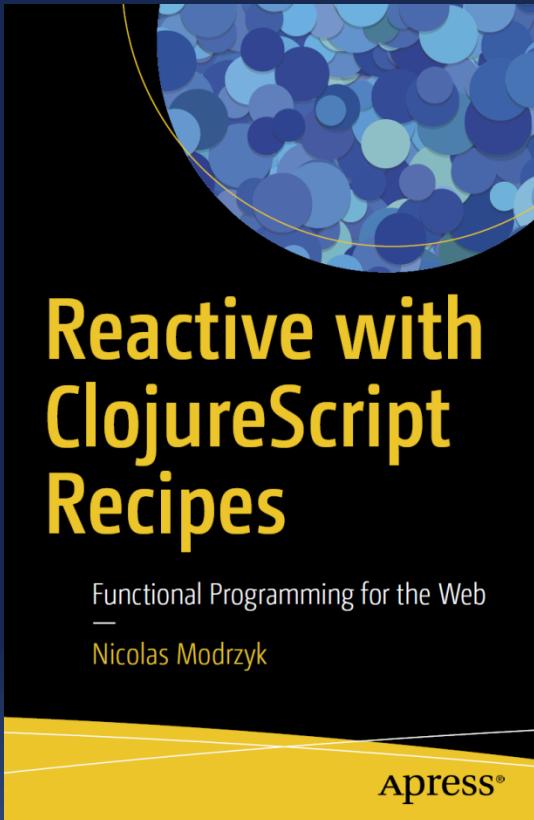


Interação com JavaScript





ClojureScript





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