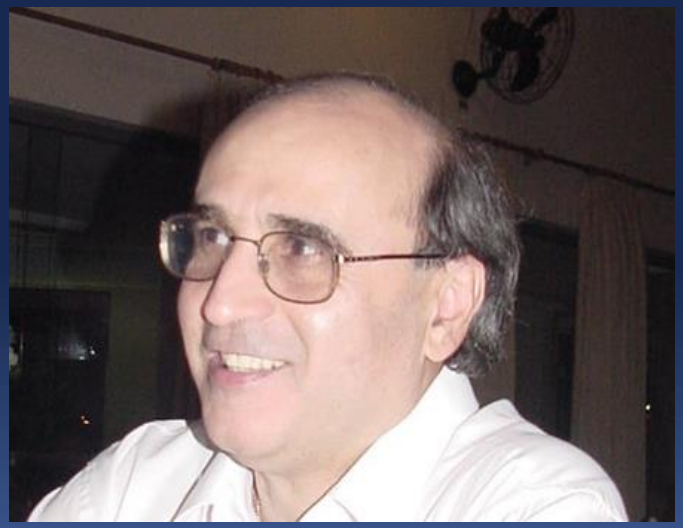
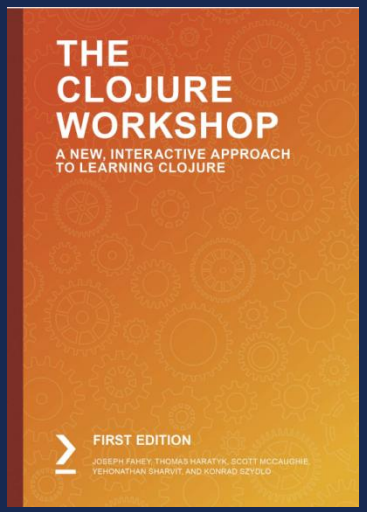
Programação Funcional

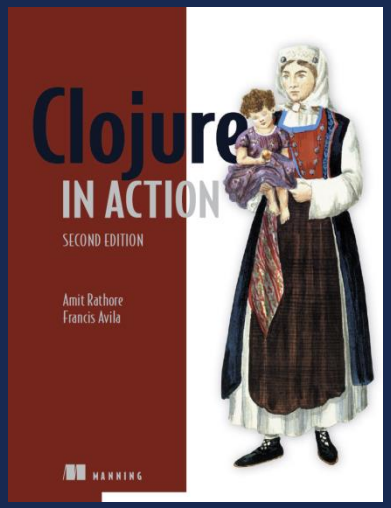
Unidade 4 – Visão geral da Linguagem Clojure

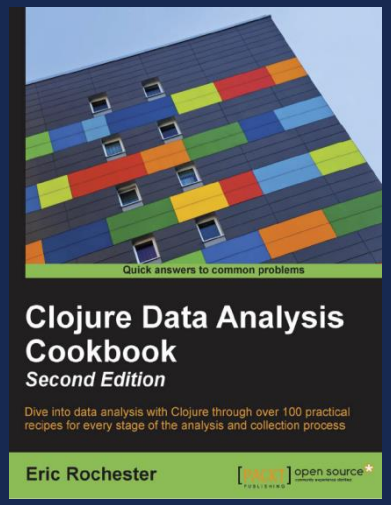


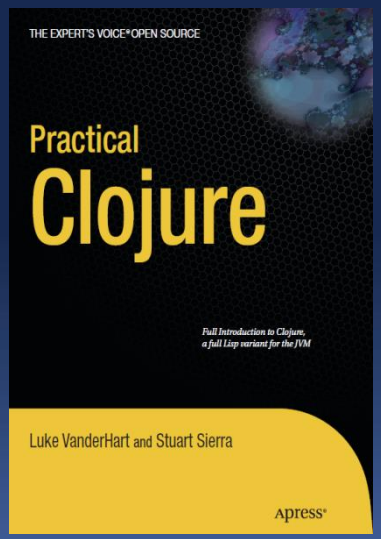
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@prof.uscs.edu.br
aparecidovfreitas@gmail.com

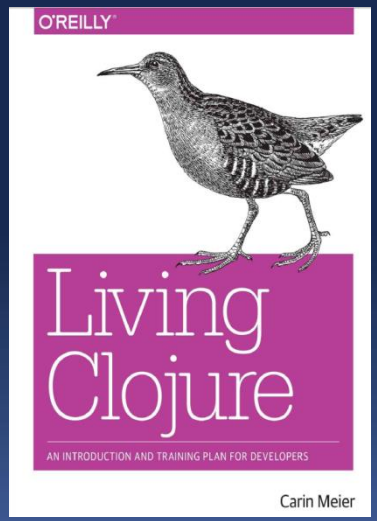
Bibliografia

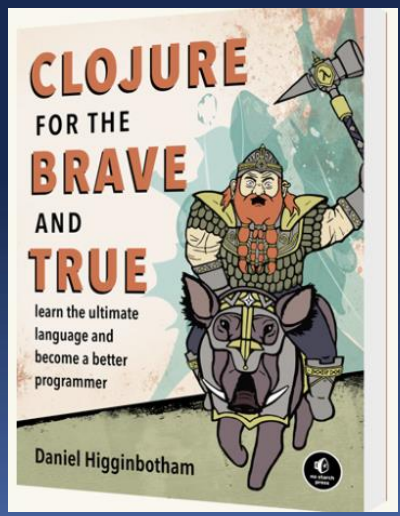








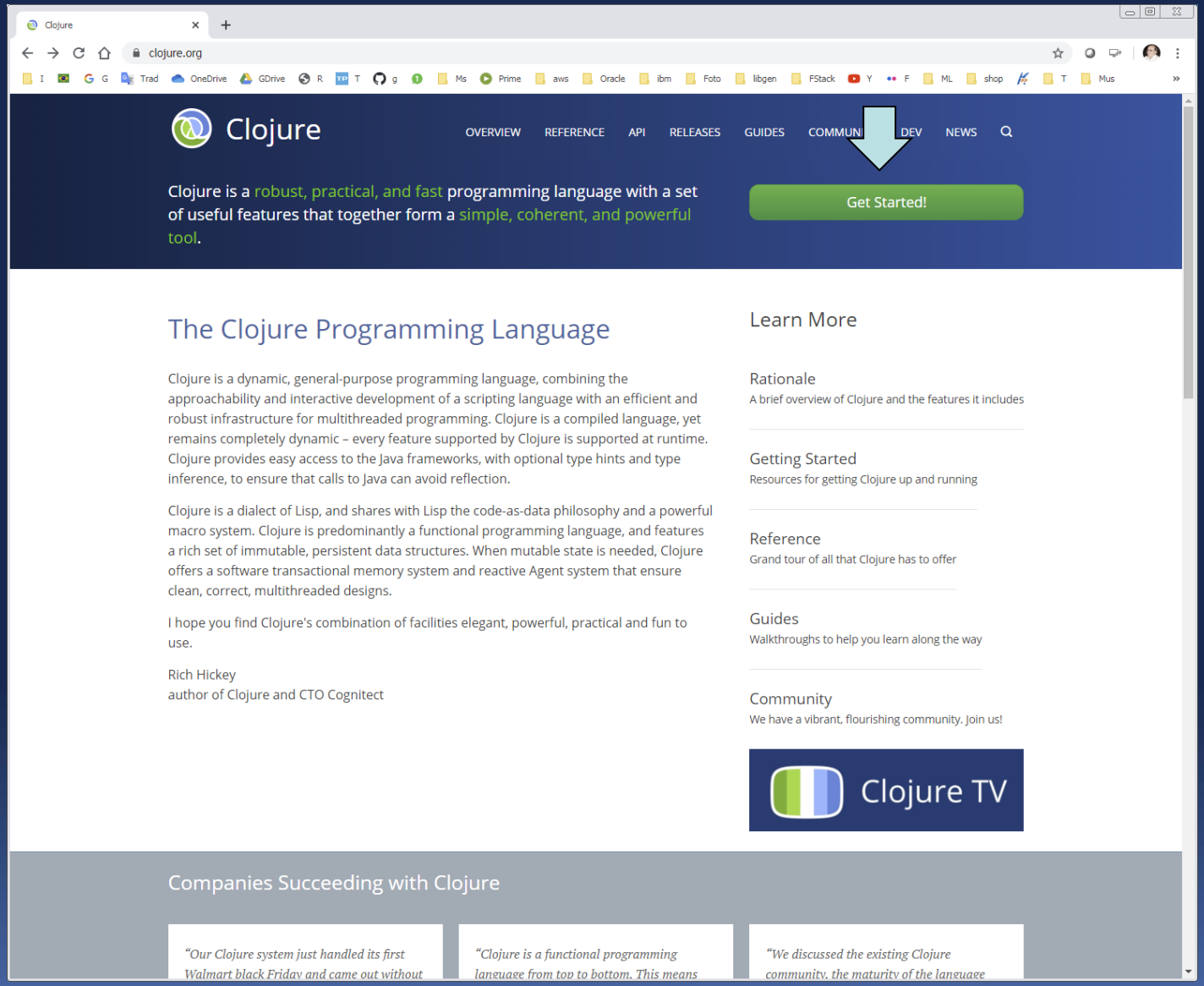




Antes de iniciar

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

https://clojure.org/



Instruções de Instalação

Local build

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

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

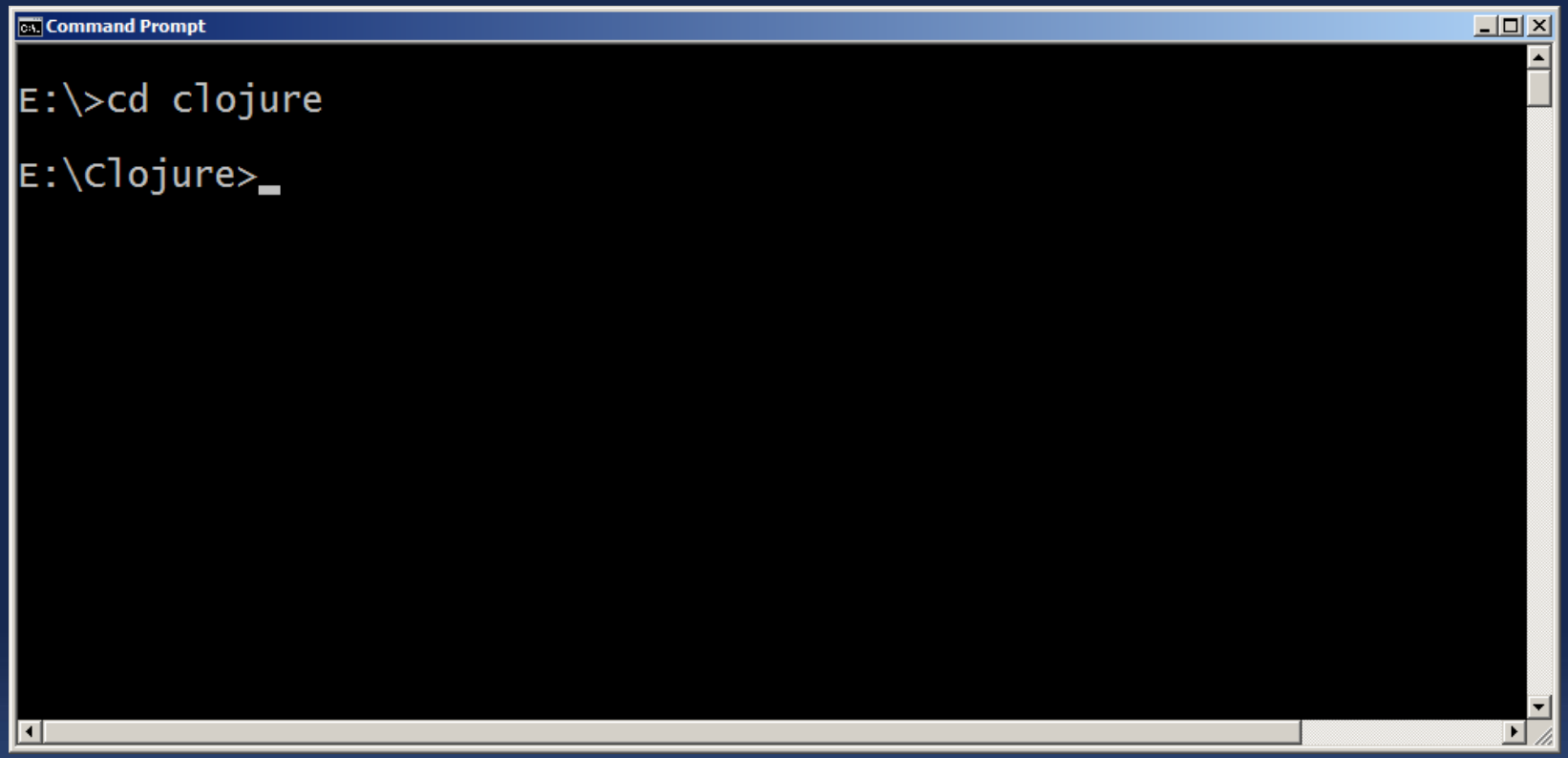
Then start the REPL with the local jar:

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

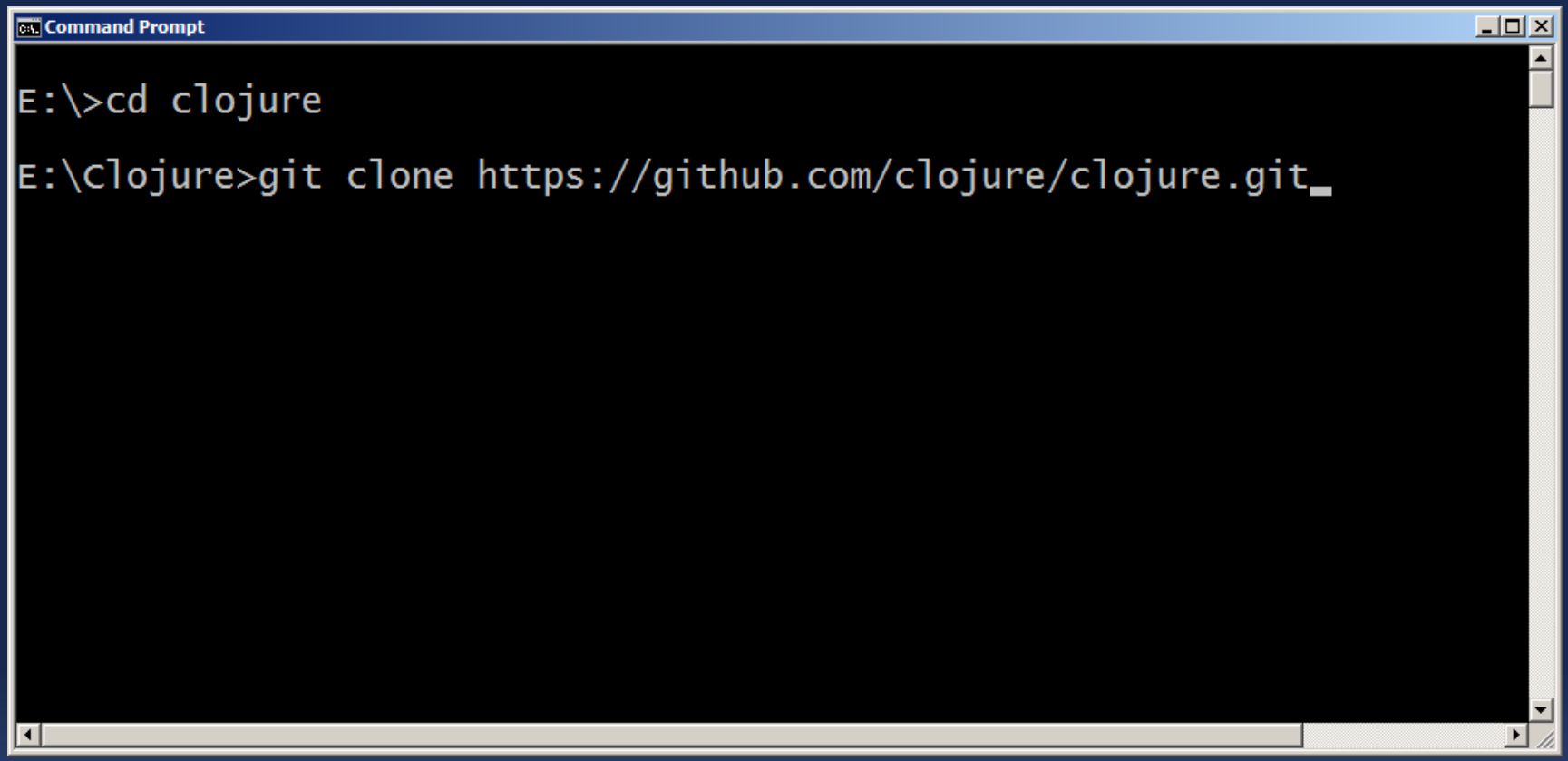
Try Clojure online

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

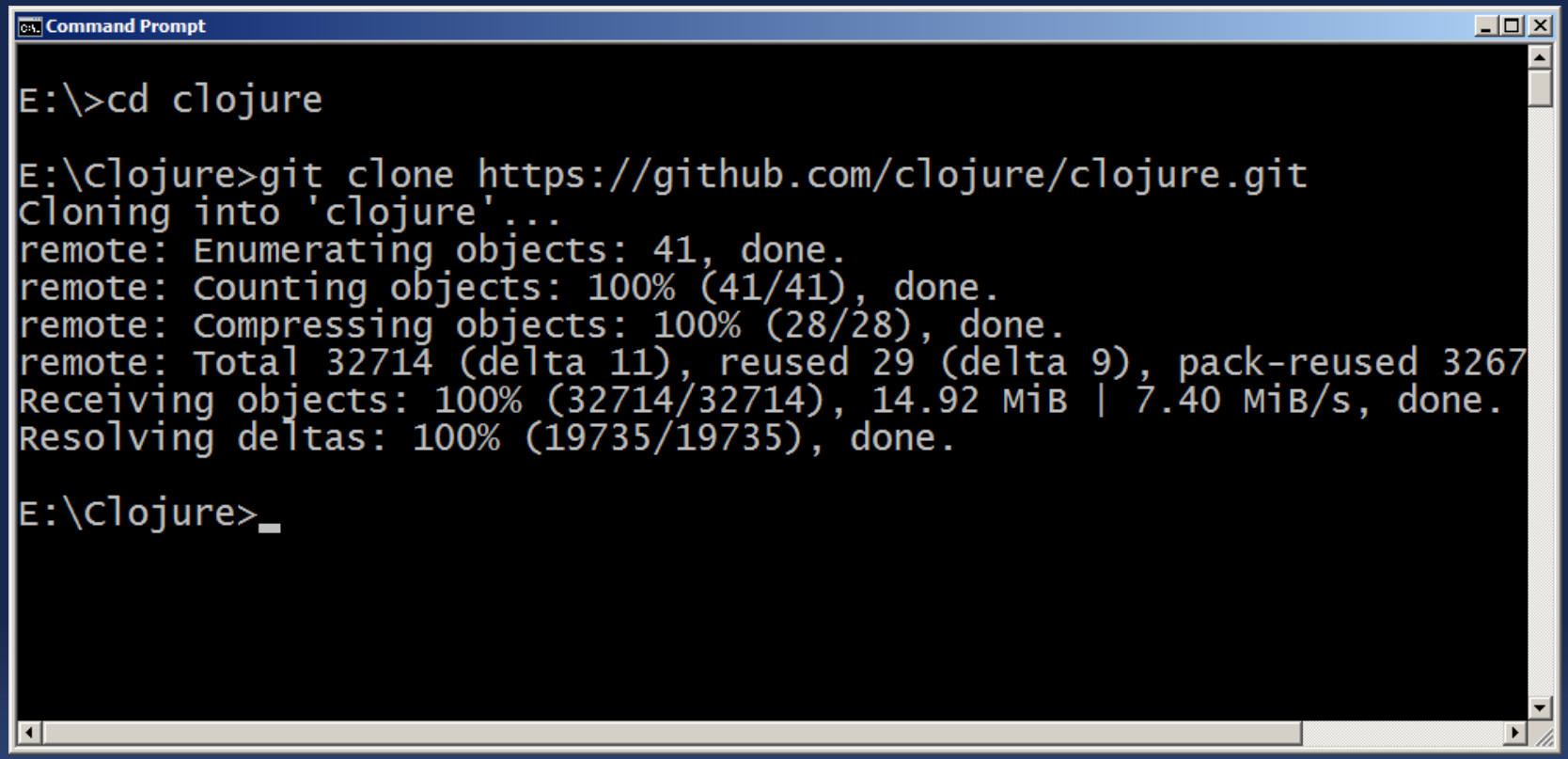
Criar pasta E:\clojure



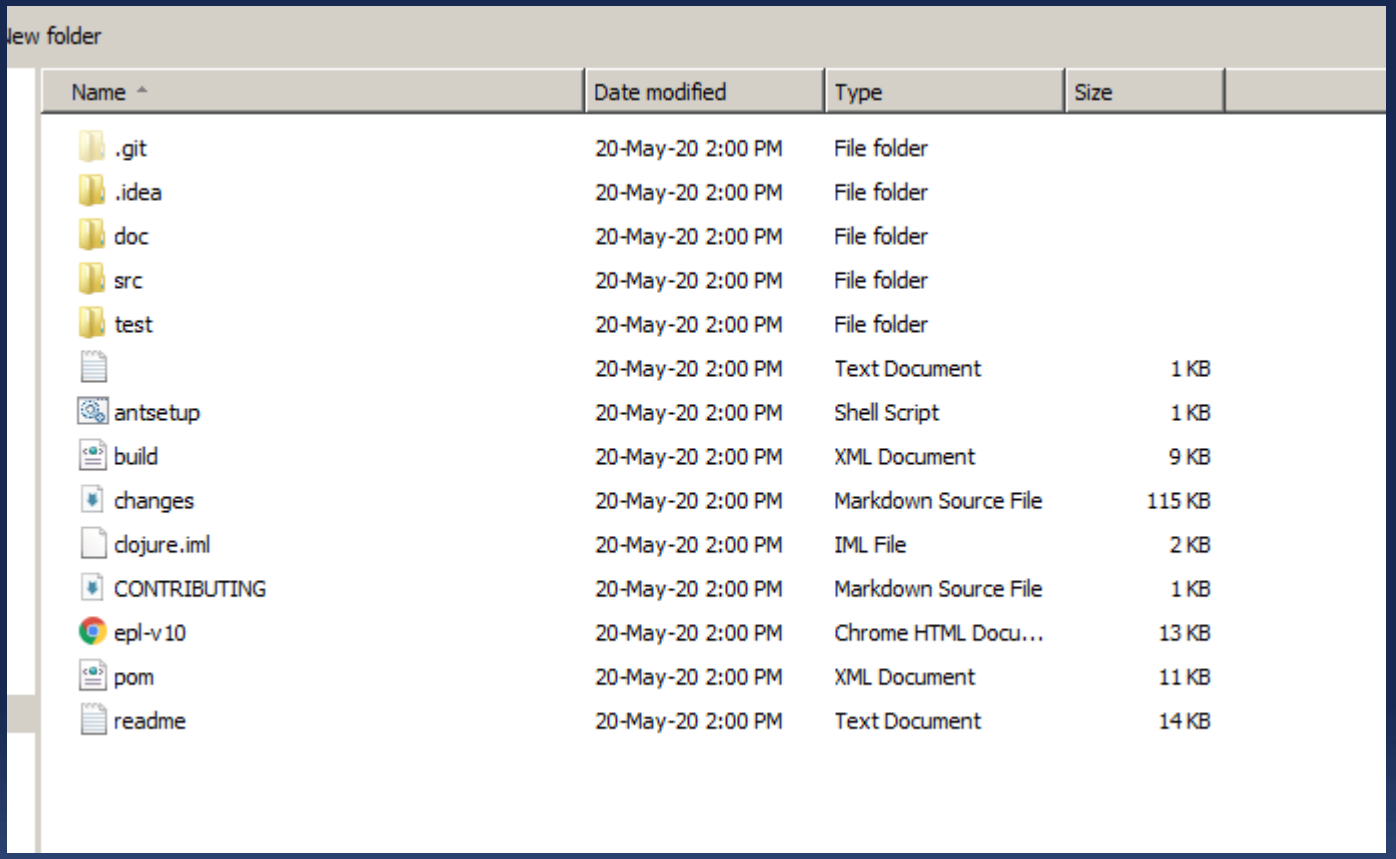
Baixando clojure



Baixando clojure

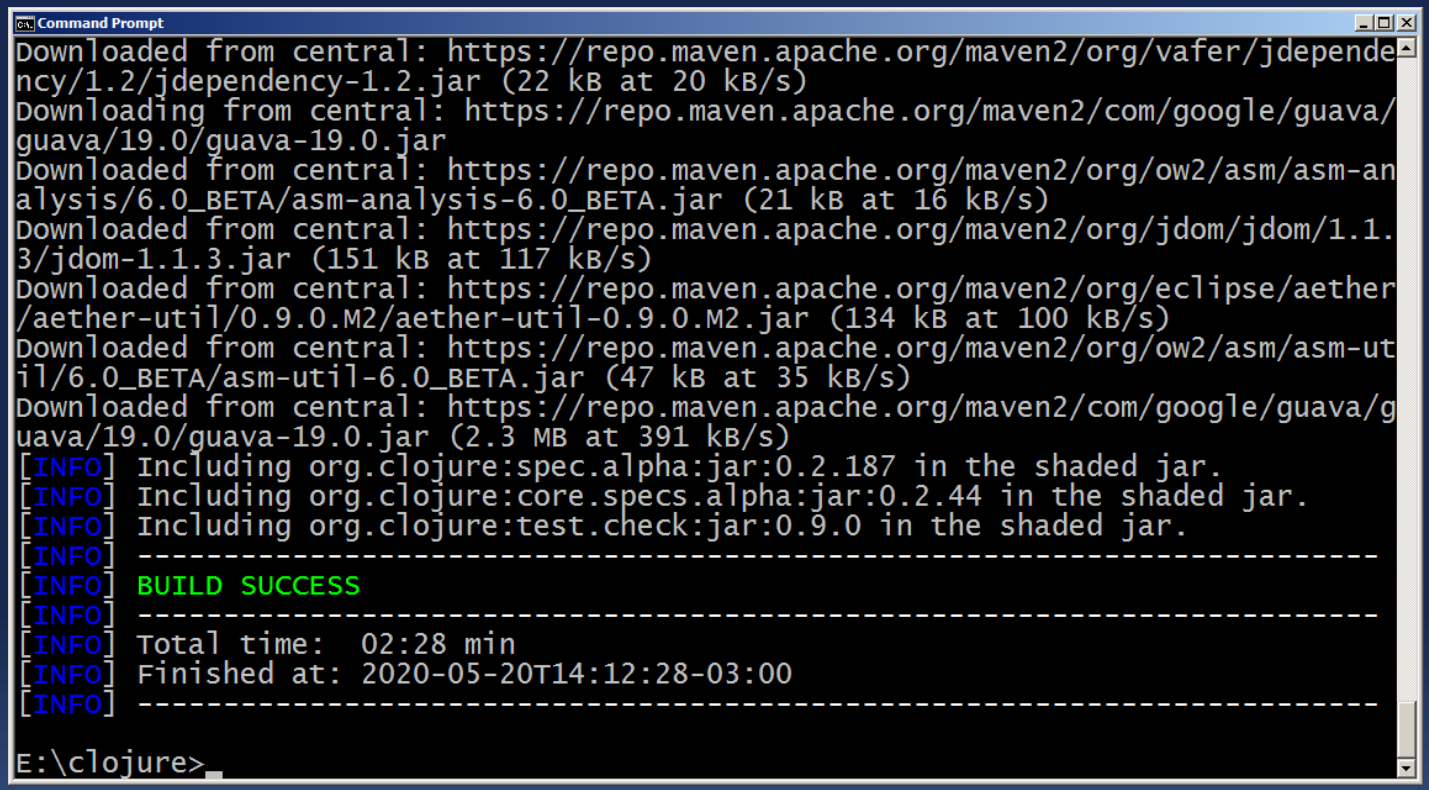


Pasta Clojure



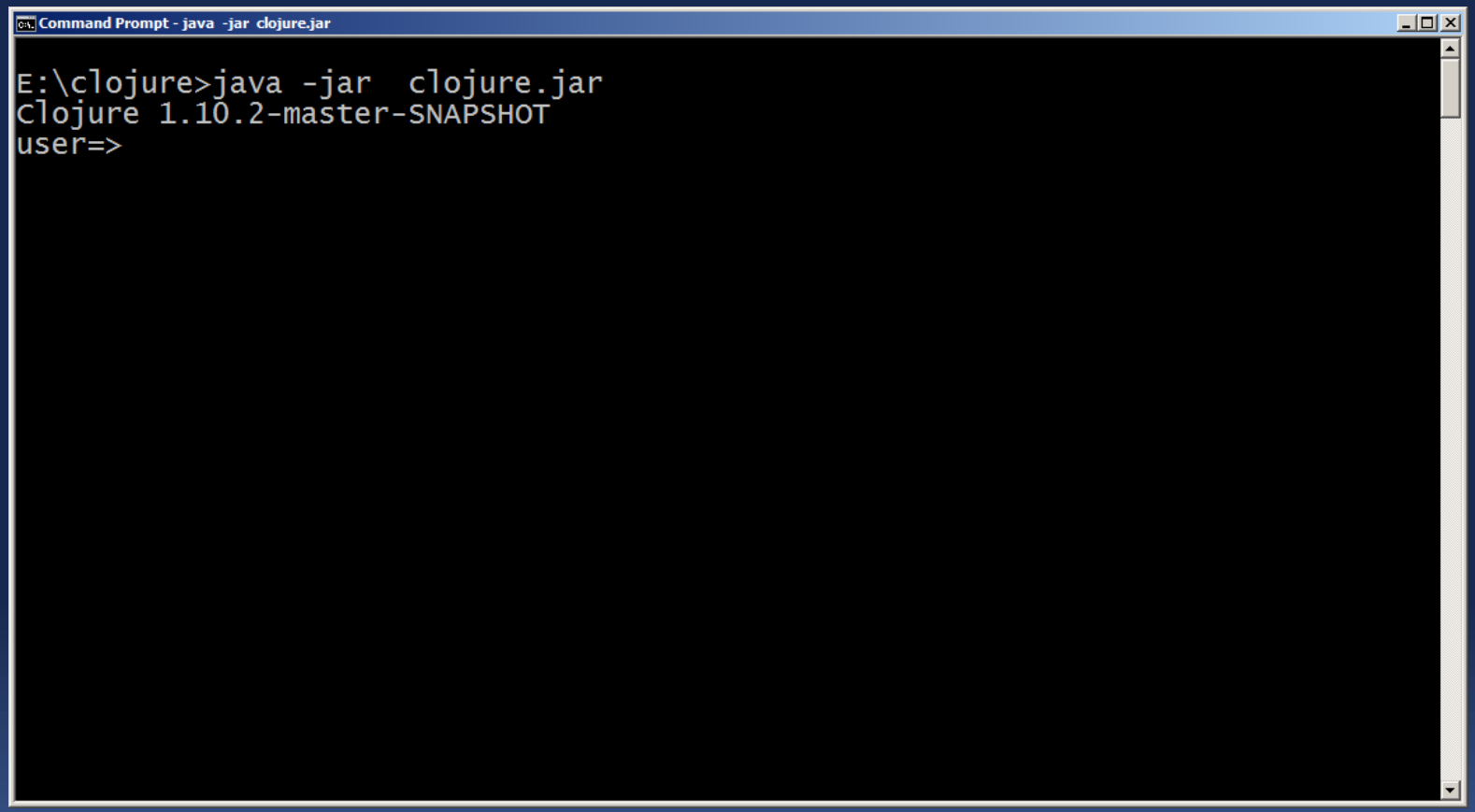
Building Clojure com Maven

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



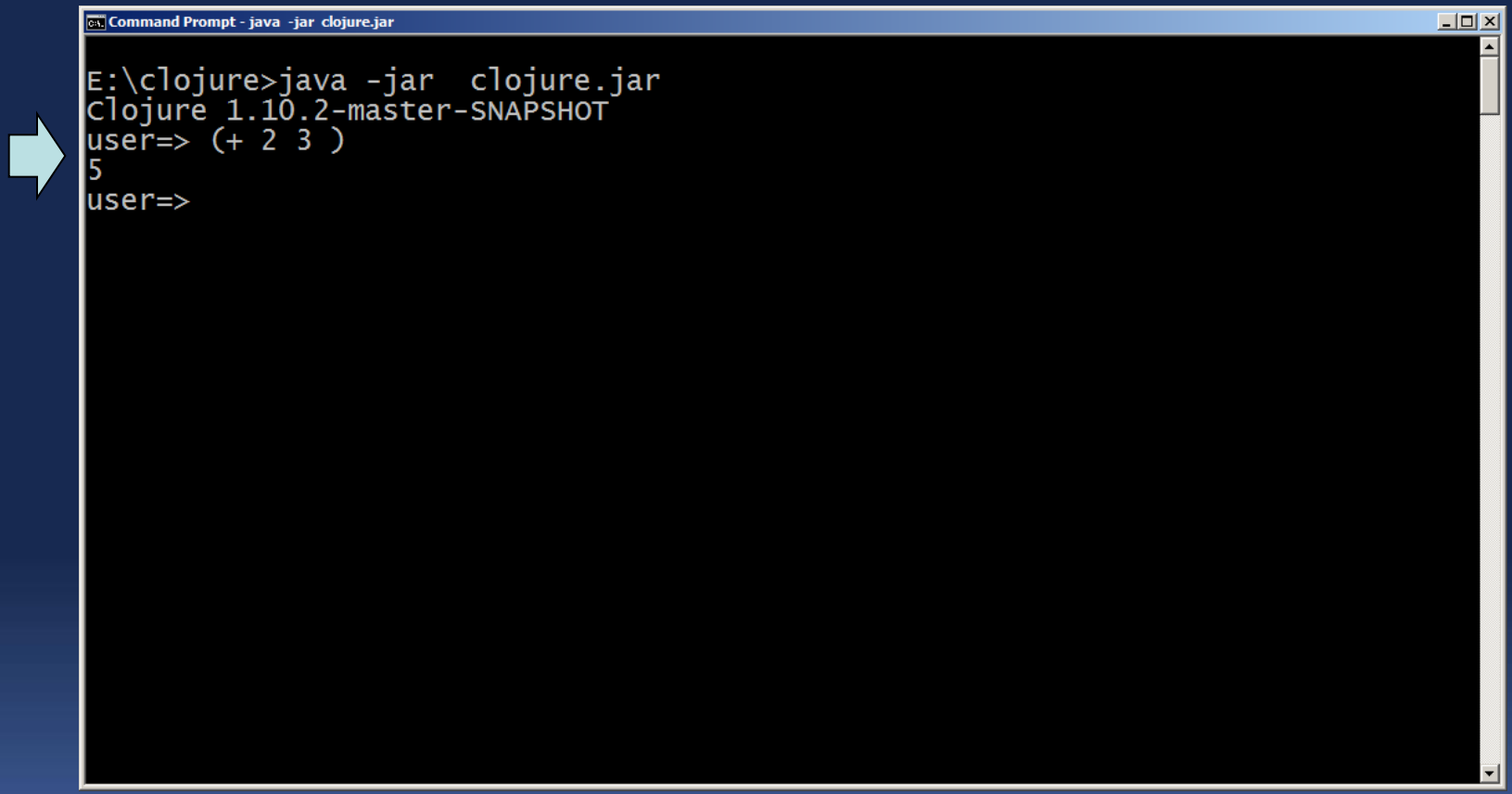
Iniciando REPL

java -jar clojure.jar



Testando REPL

java -jar clojure.jar



Clojure installer e CLI tools

Clojure installer e CLI tools

Clojure installer and CLI tools

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

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

Installation on Windows

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

clj on Windows

clj on Windows

Alex Miller edited this page 5 days ago · 37 revisions

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

Install

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

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

clj on Windows

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

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

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

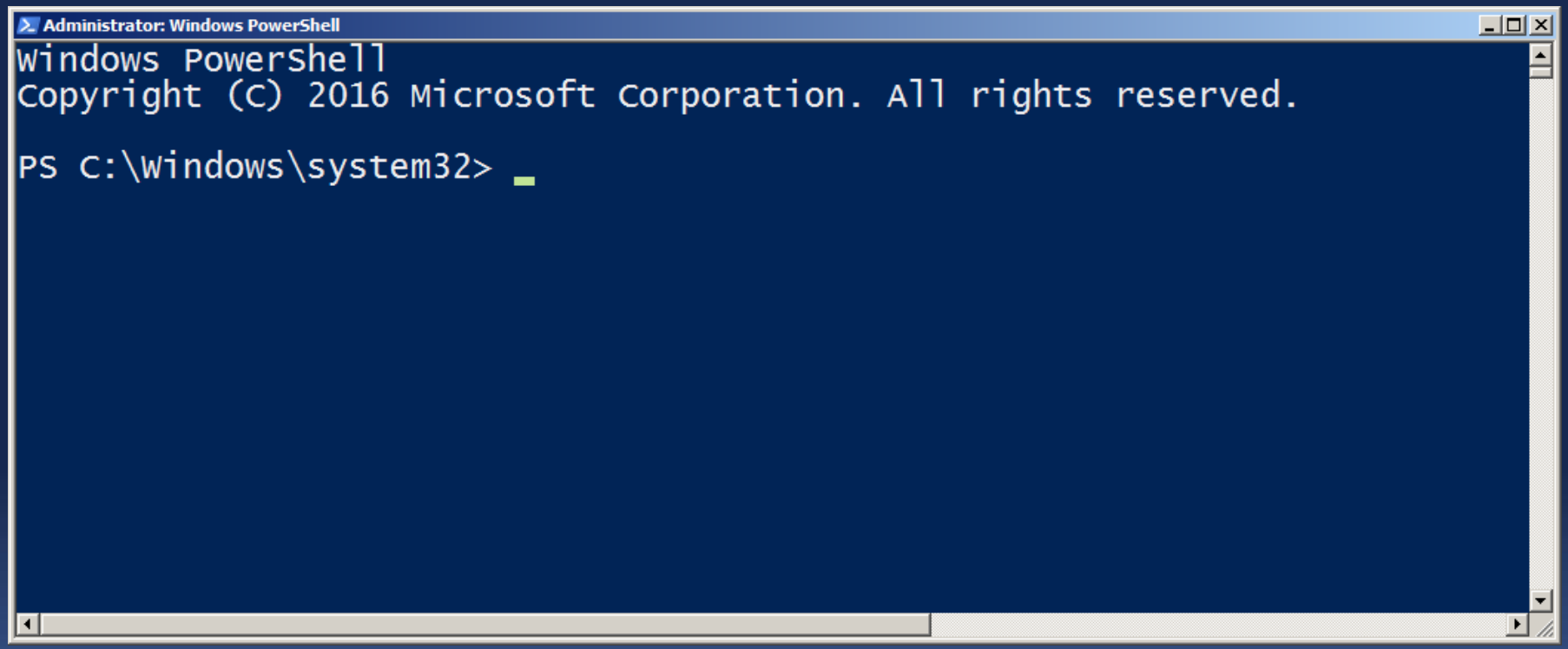
```
PS Y:\Downloads> .\win-install-1.10.1.561.ps1
Downloading Clojure tools
WARNING: Clojure will install as a module in your PowerShell module path.

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

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

When choosing which location to install consider these tradeoffs:

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

clj on Windows

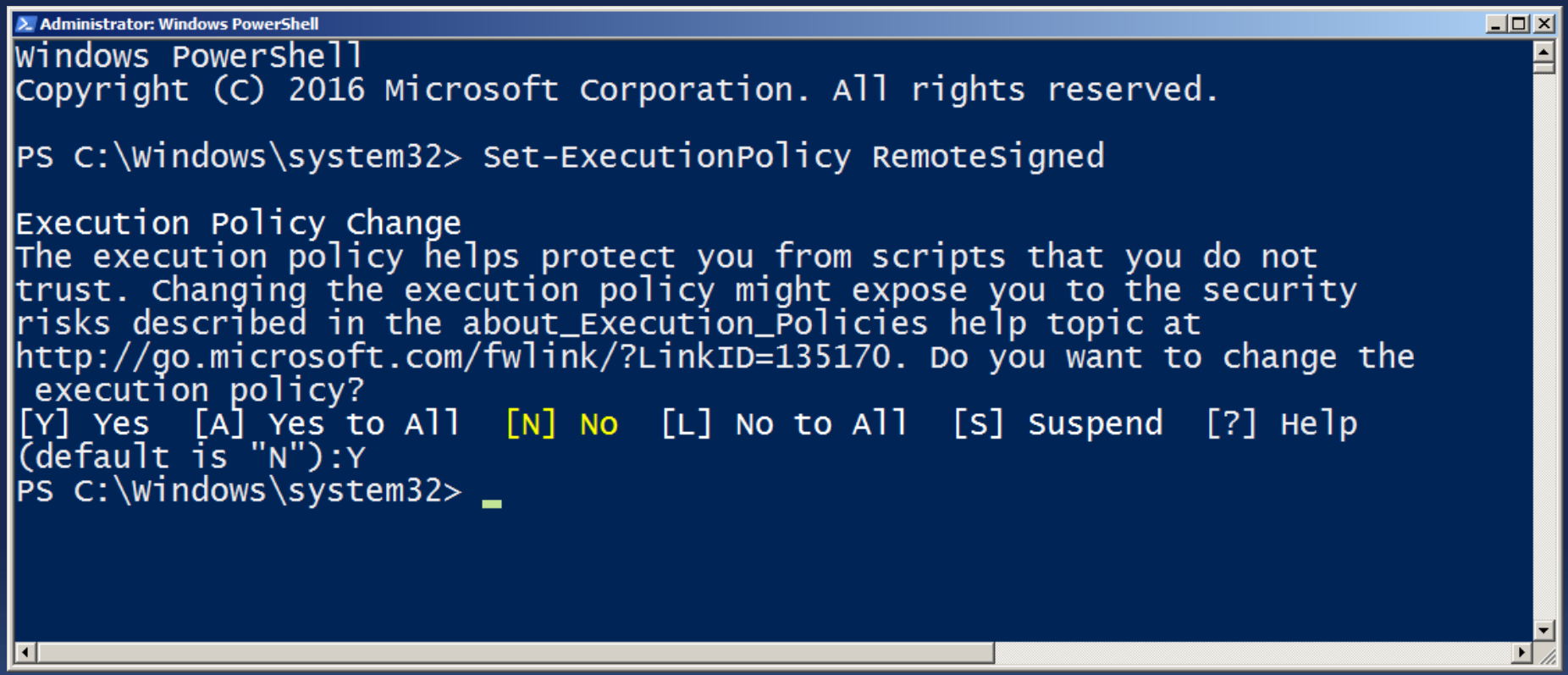
- ✓ A instalação será feita com Powershell versão 5
- ✓ Esteja certo de que **PowerShell 5** ou superior esteja instalado em sua máquina;



Configurações de segurança - Powershell

Habilitar PowerShell para execução

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



Habilitar PowerShell para execução

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

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

Fix for PowerShell Script Not Digitally Signed

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

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

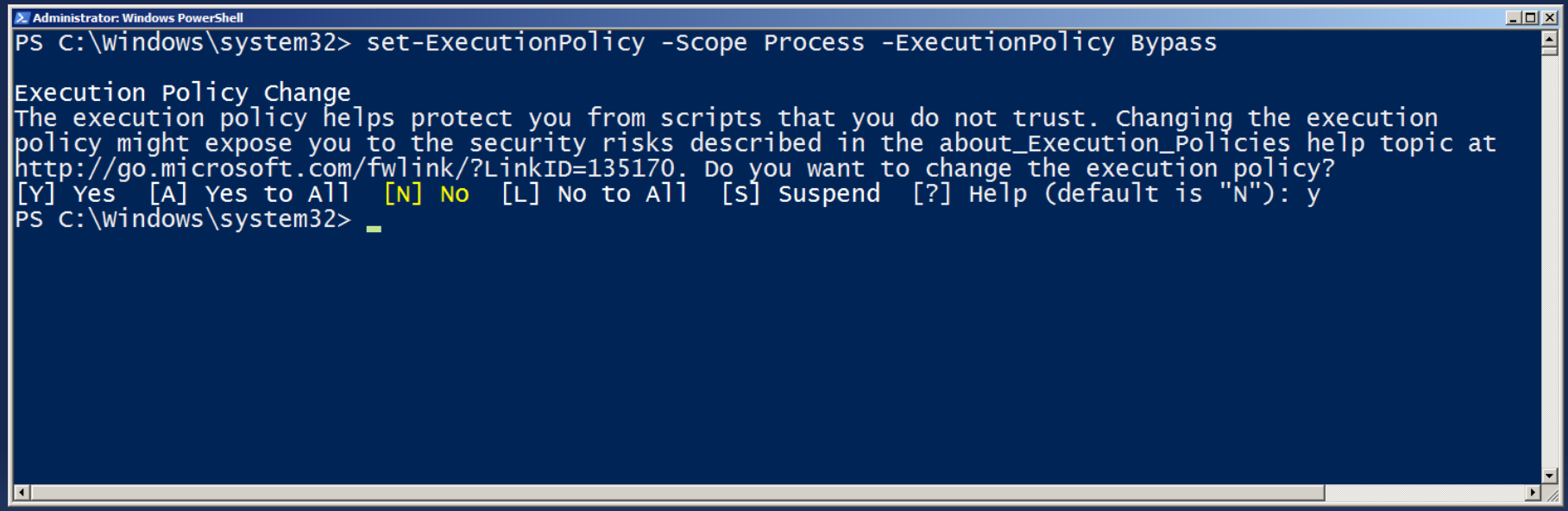
```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

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

Habilitar PowerShell para execução

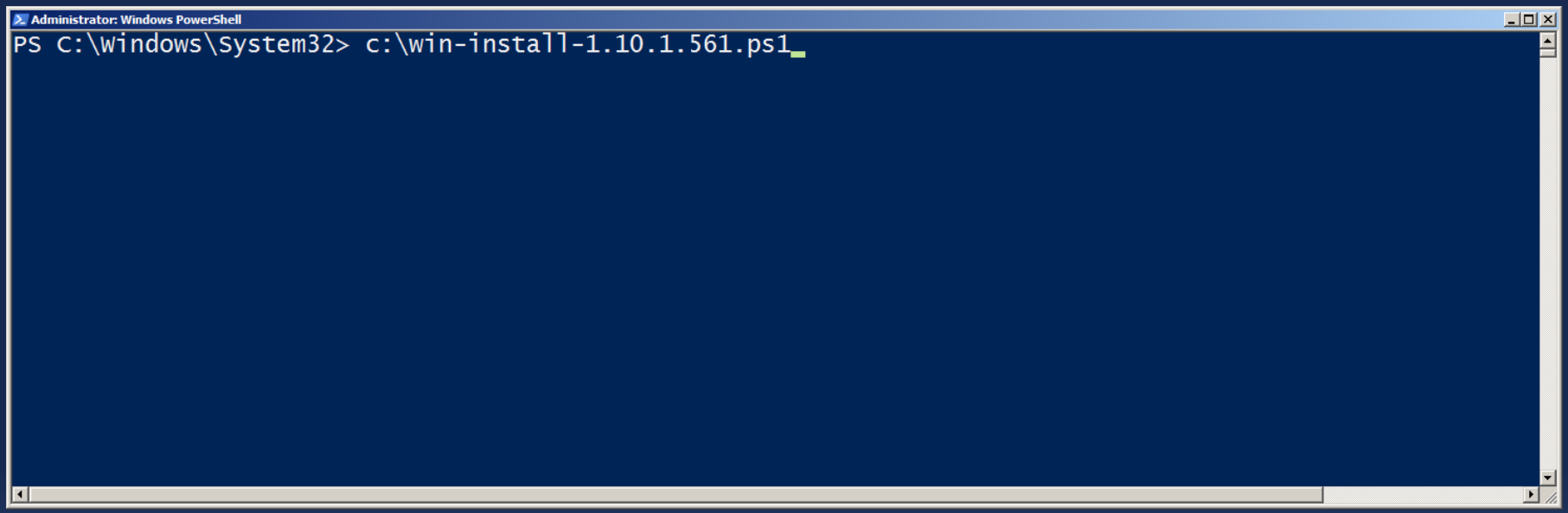
- ✓ No prompt do PowerShell: (em modo administrador)

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



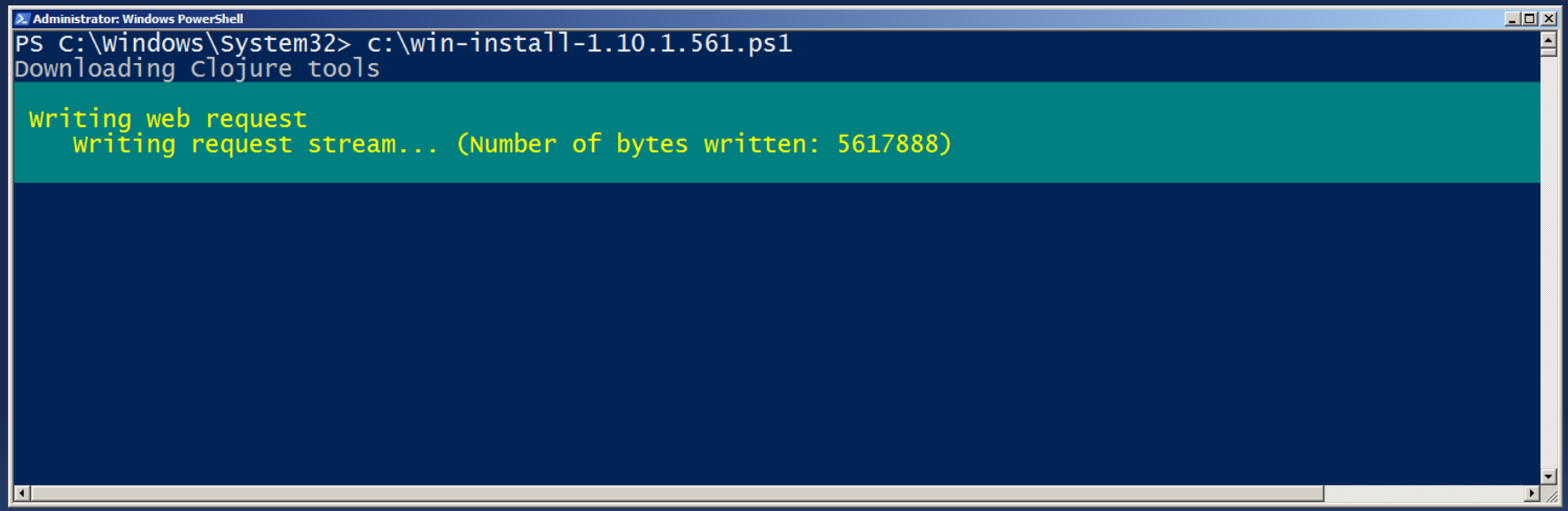
clj on Windows

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



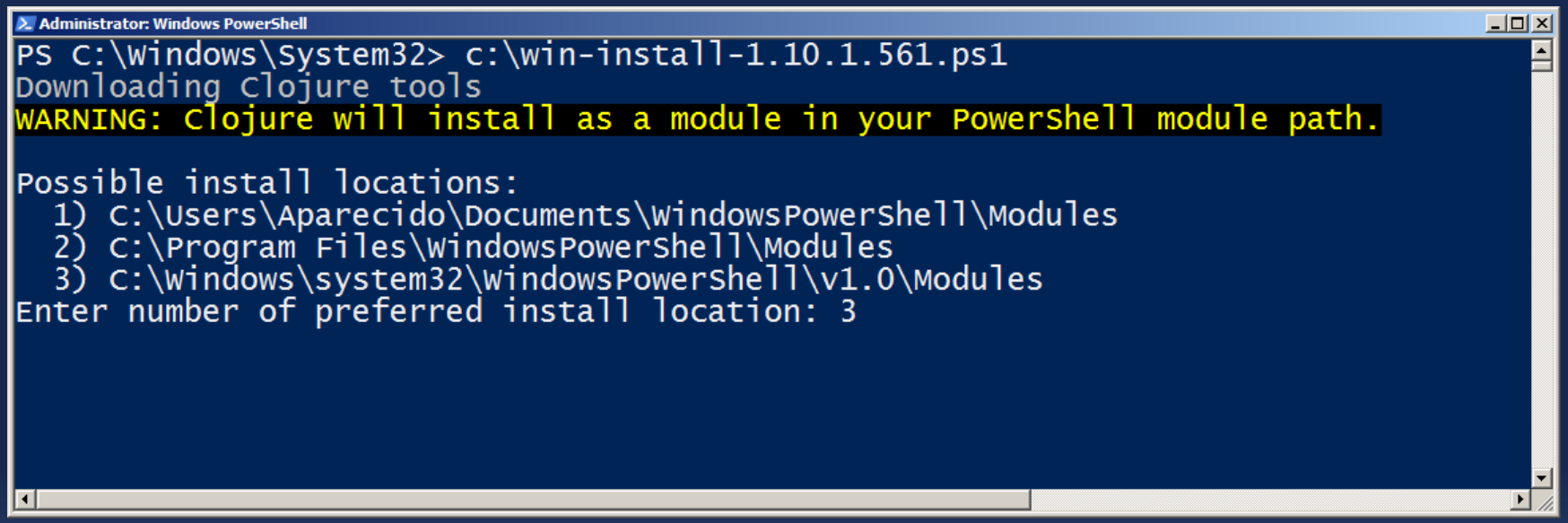
clj on Windows

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



clj on Windows

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

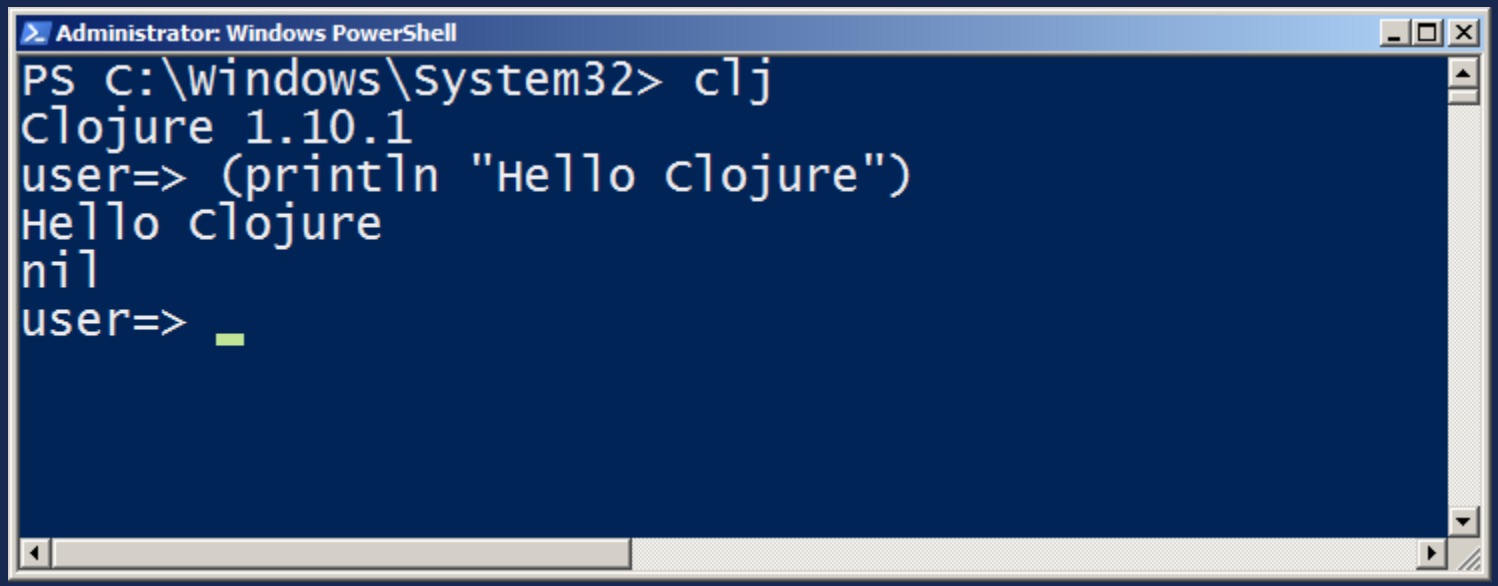


clj on Windows

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

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

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

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

clj on Windows



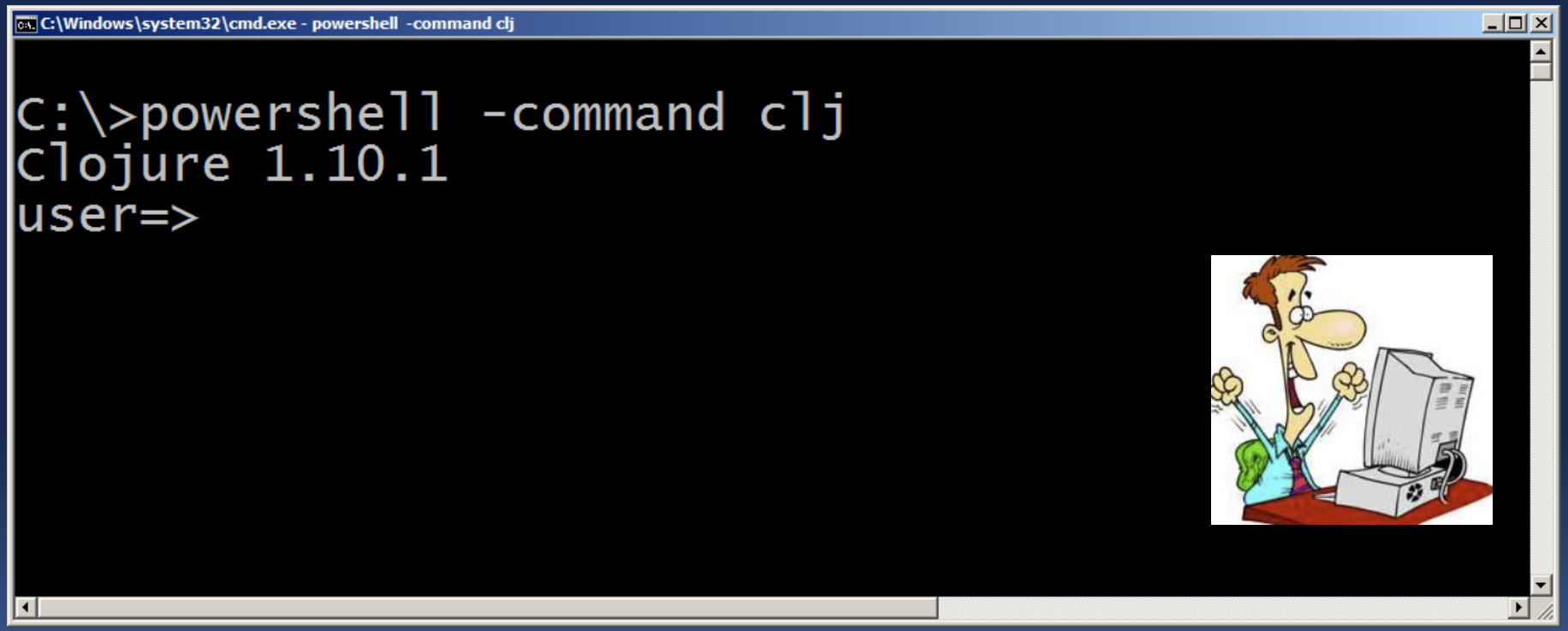
Executando clj

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

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

Ativando clj

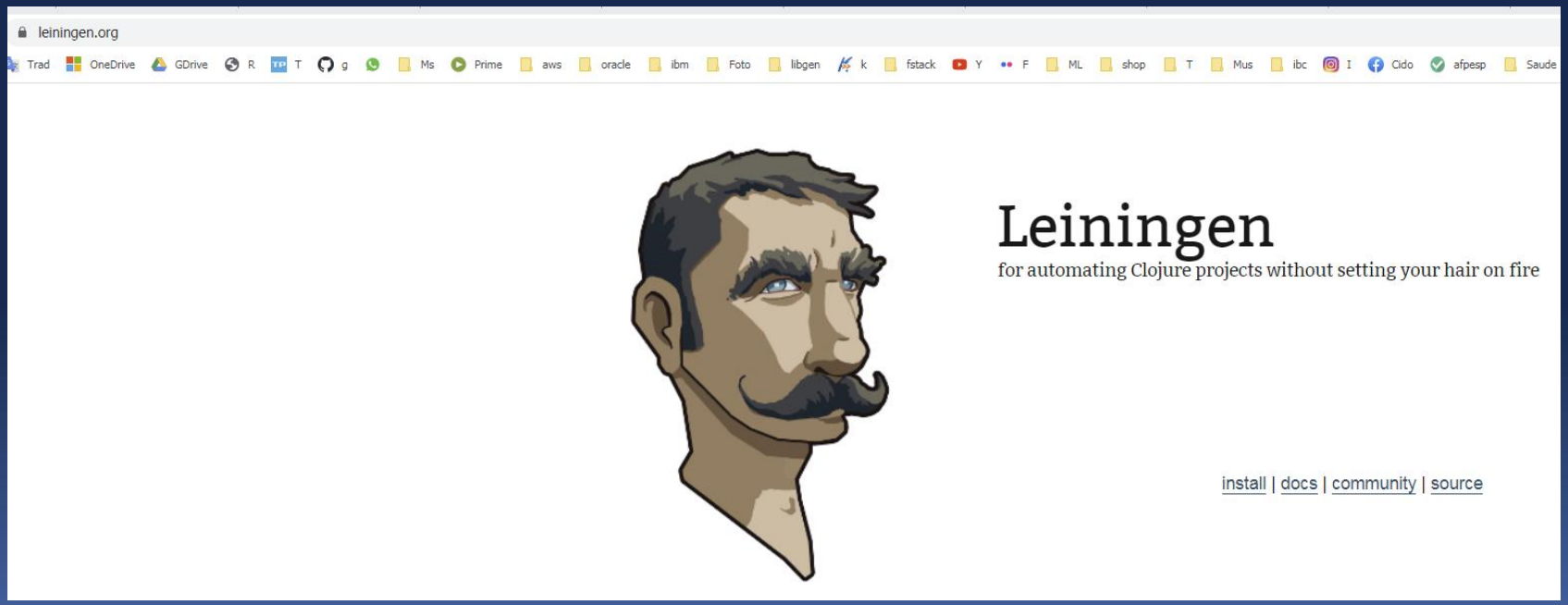
- ✓ Sob prompt do Windows: powershell -command clj



Instalação com Leiningen

Leiningen – www.leiningen.org

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



Leiningen - Install

Install

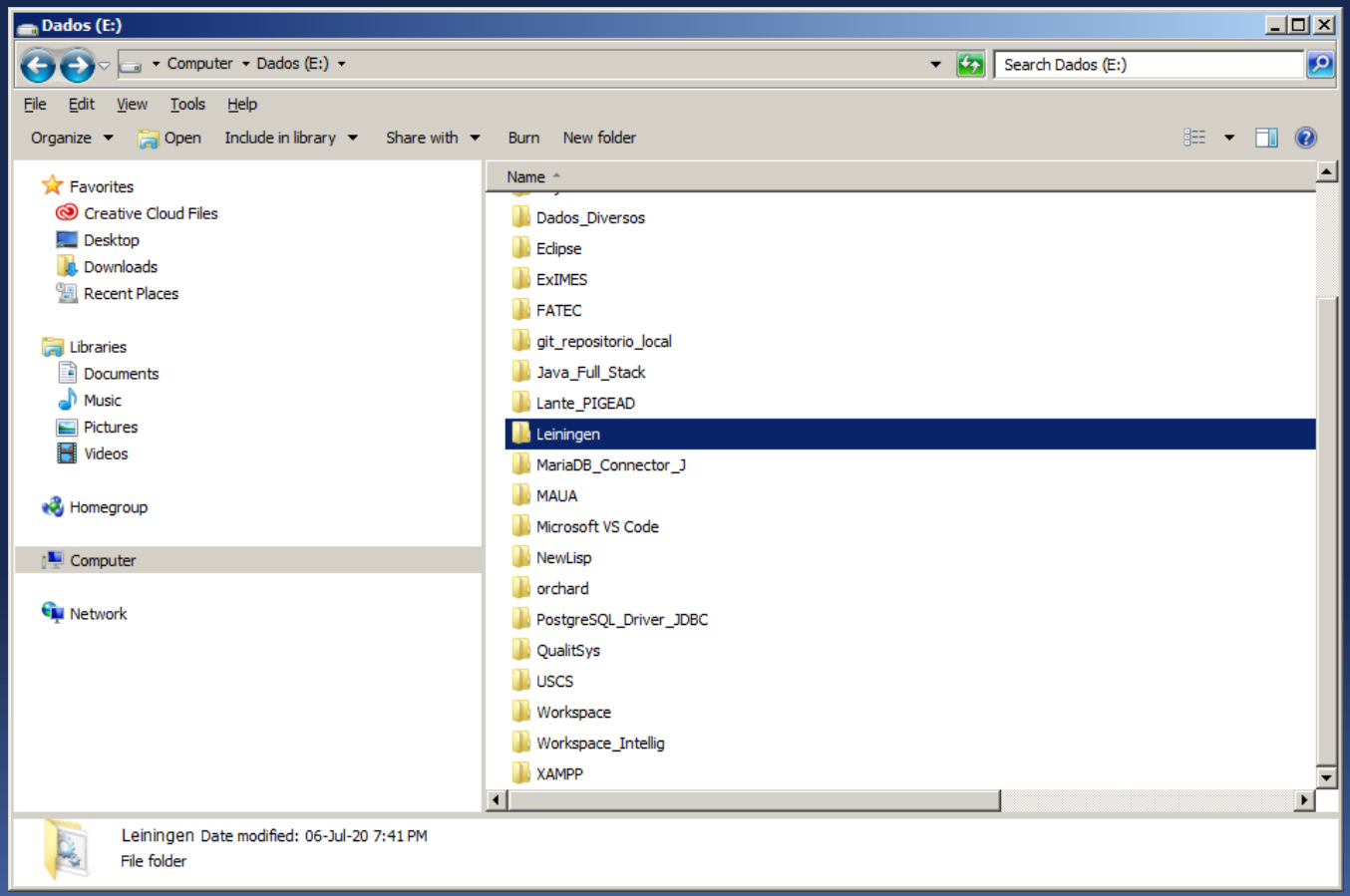
Leiningen and Clojure require Java. OpenJDK version 8 is recommended at this time.

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

You can check your package manager as well..

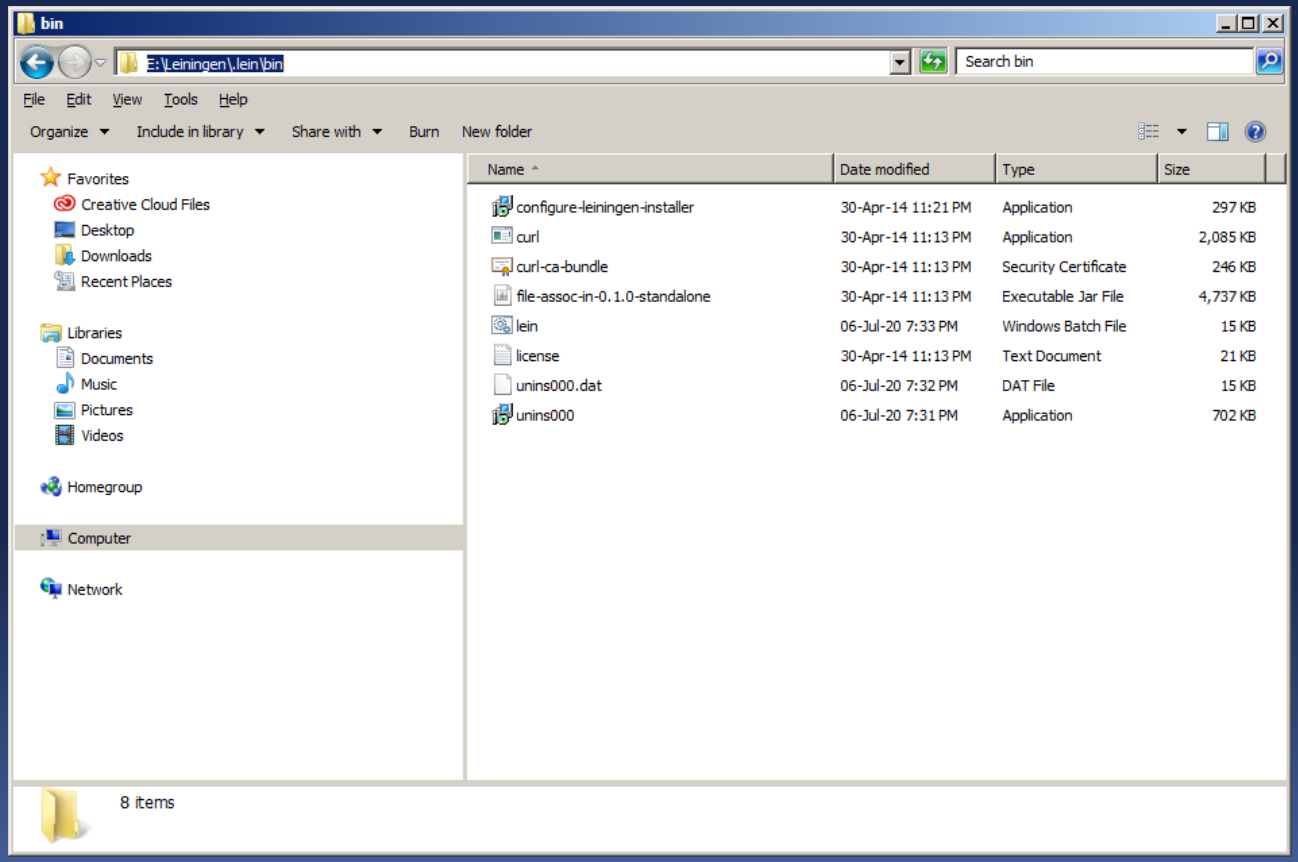
Leiningen - Install

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



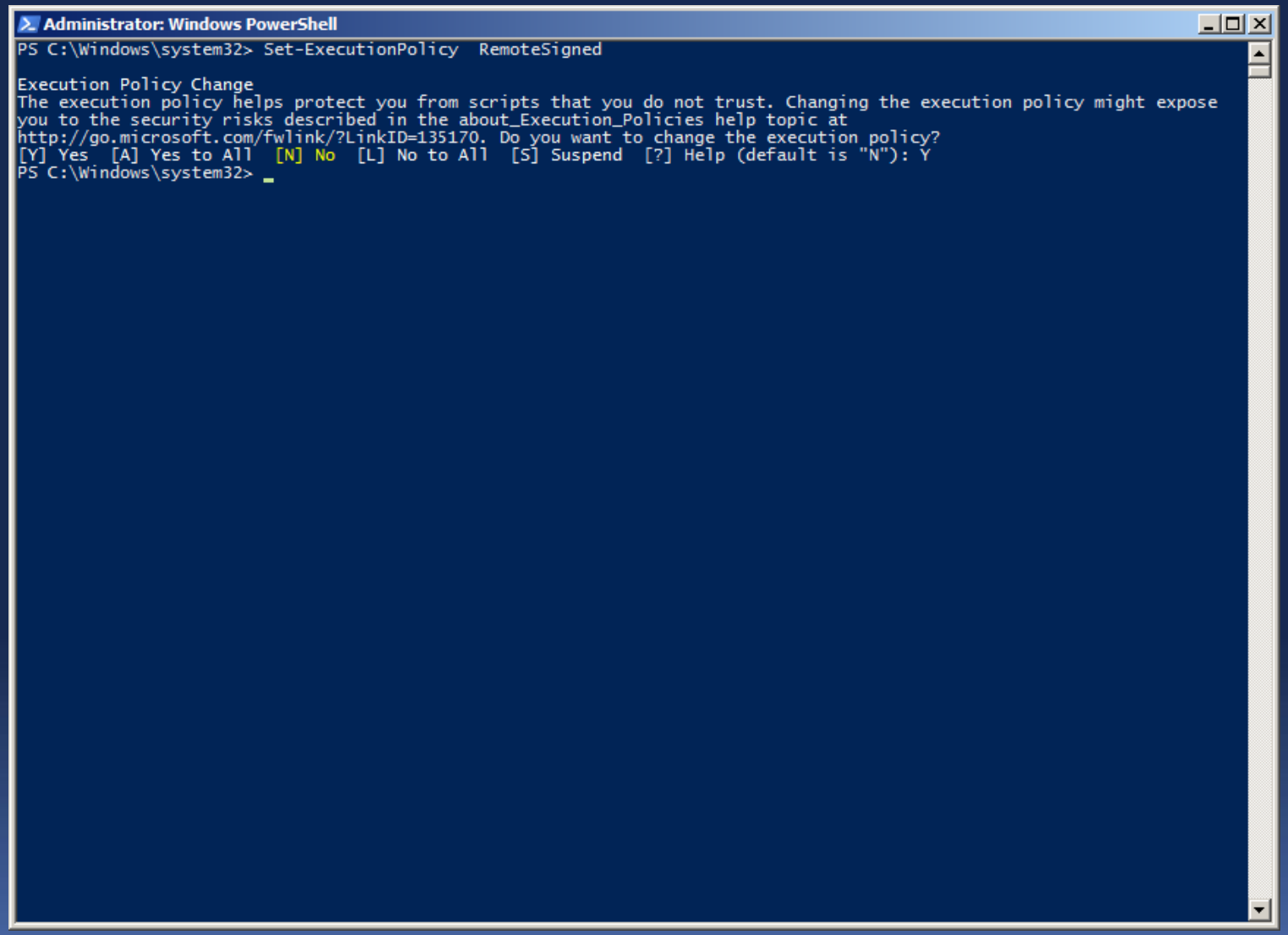
Leiningen - Install

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



Habilitar PowerShell para execução

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



Habilitar PowerShell para execução

- ✓ No prompt do PowerShell:

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

Fix for PowerShell Script Not Digitally Signed

Caio Moreno Follow

Jul 15, 2019 · 1 min read

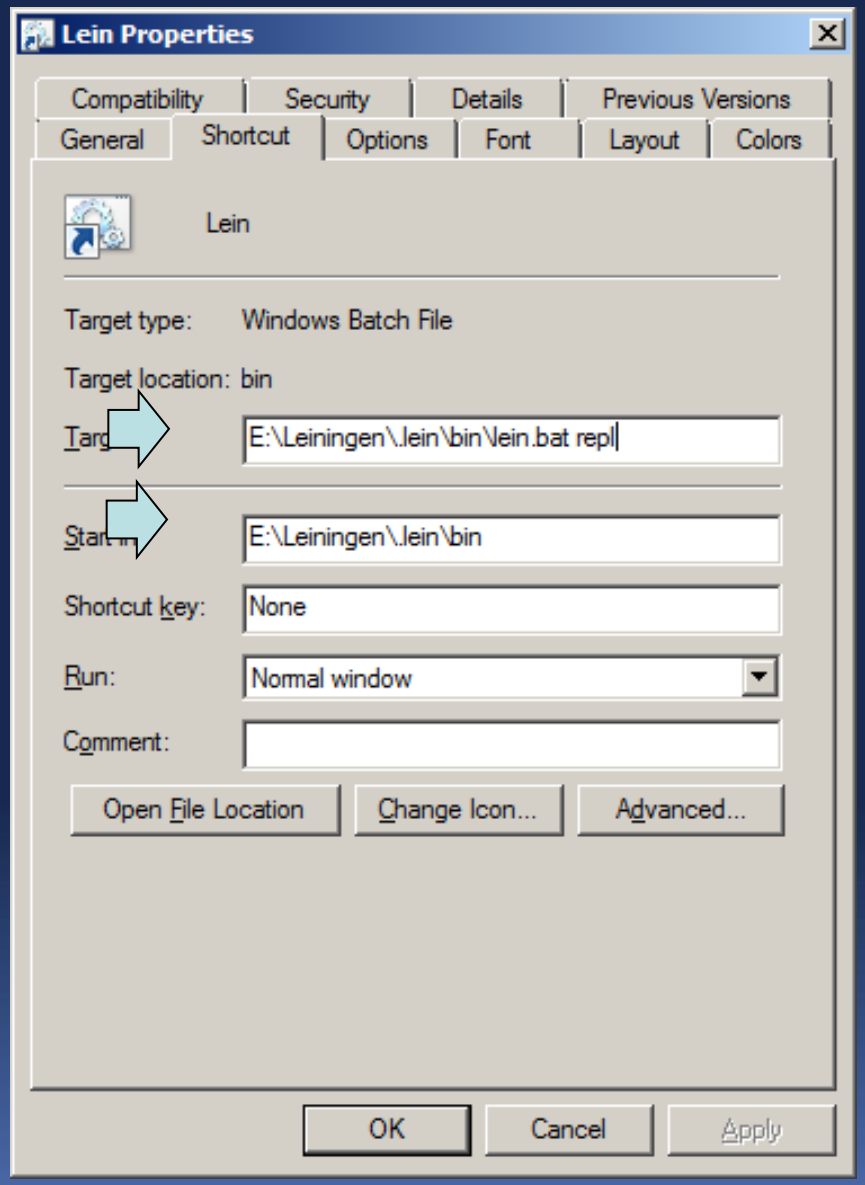
When you run a .ps1 PowerShell script you might get the message saying "**.ps1 is not digitally signed. The script will not execute on the system.**"

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

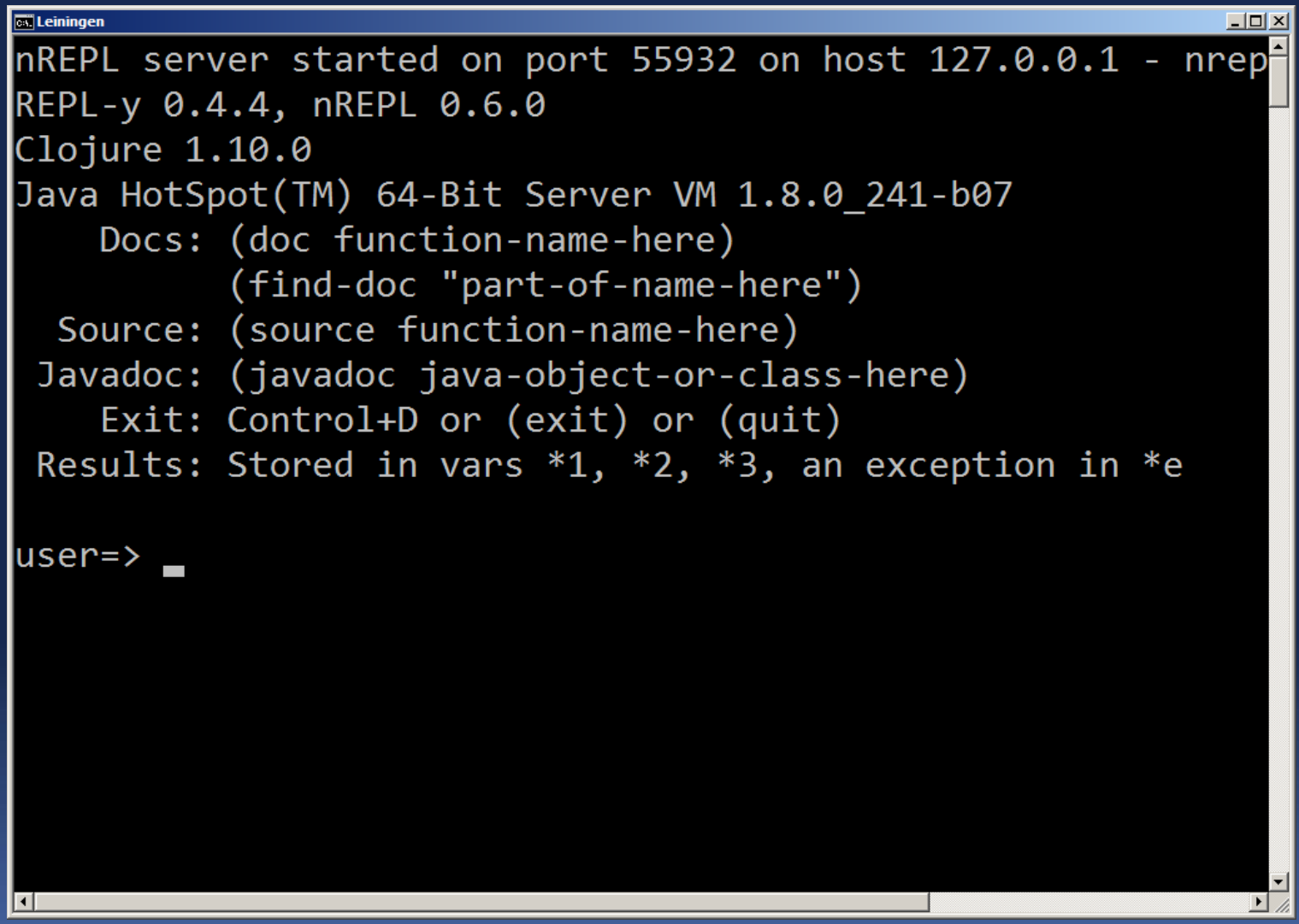
```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

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

Subindo nREPL

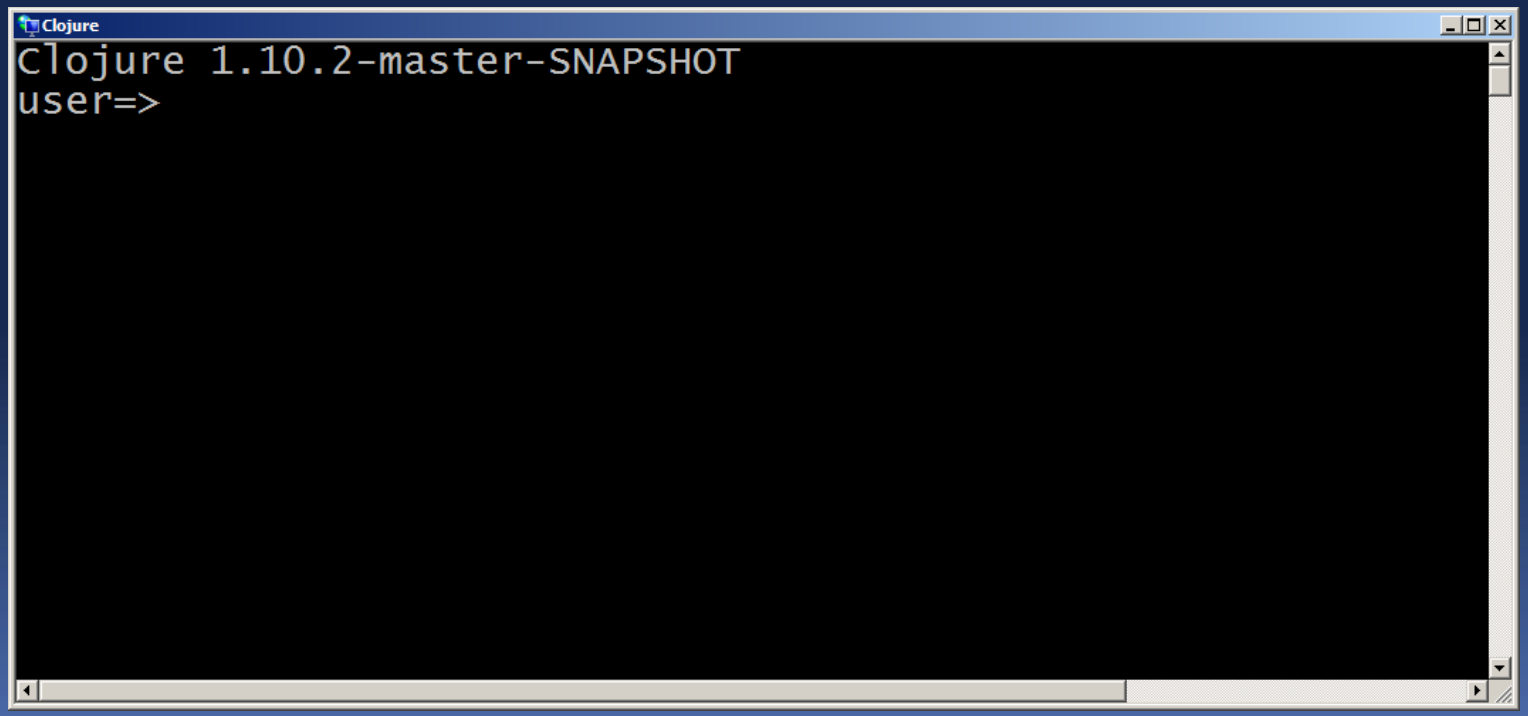


Leiningen - nREPL



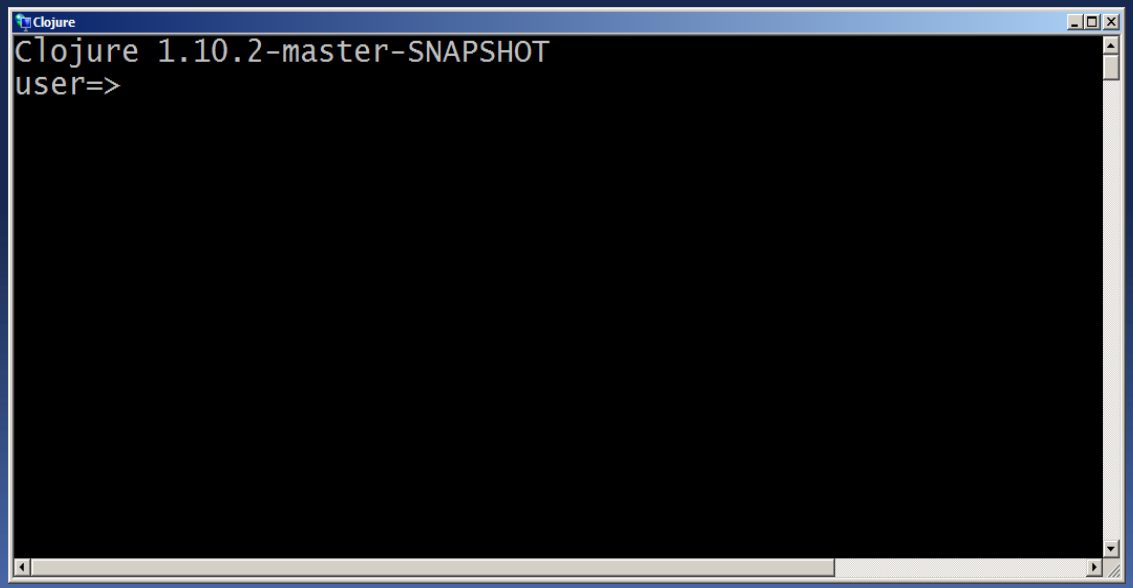
Usando REPL

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



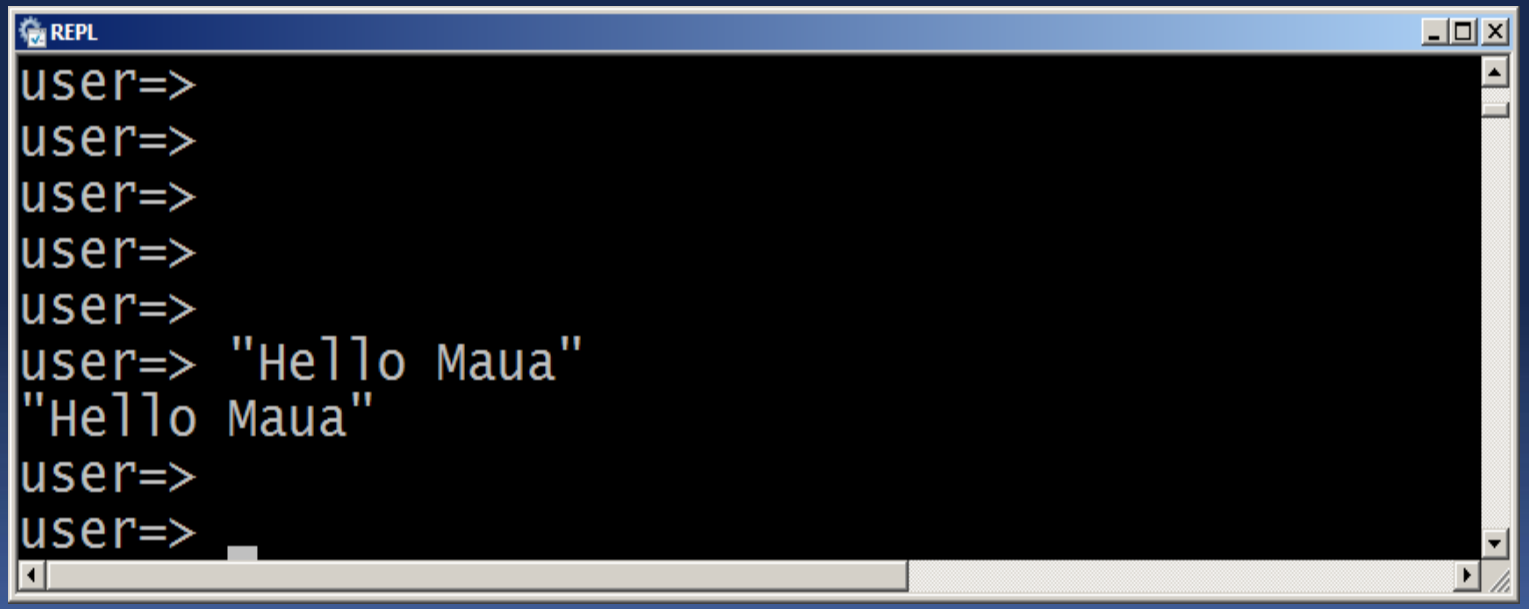
Usando REPL

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

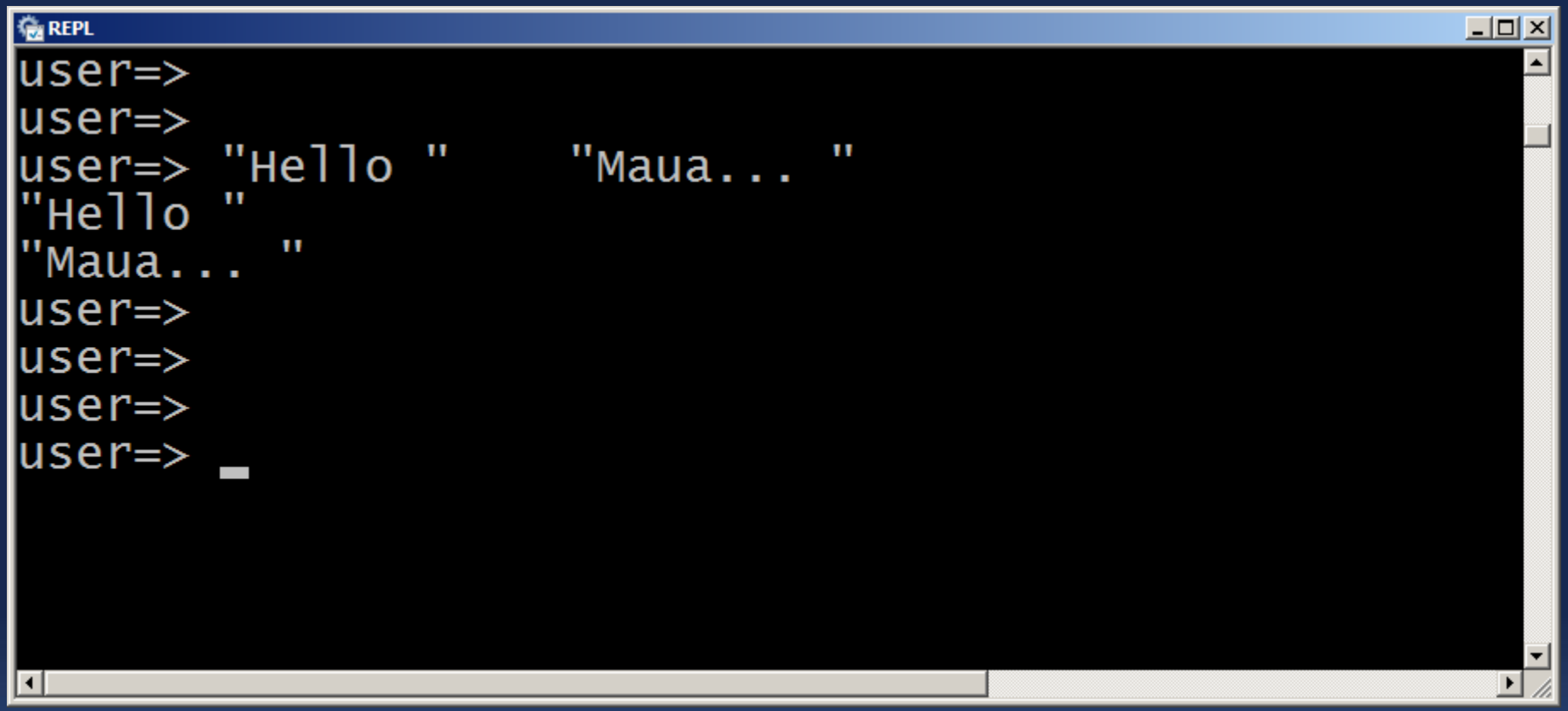


Avaliando expressões

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

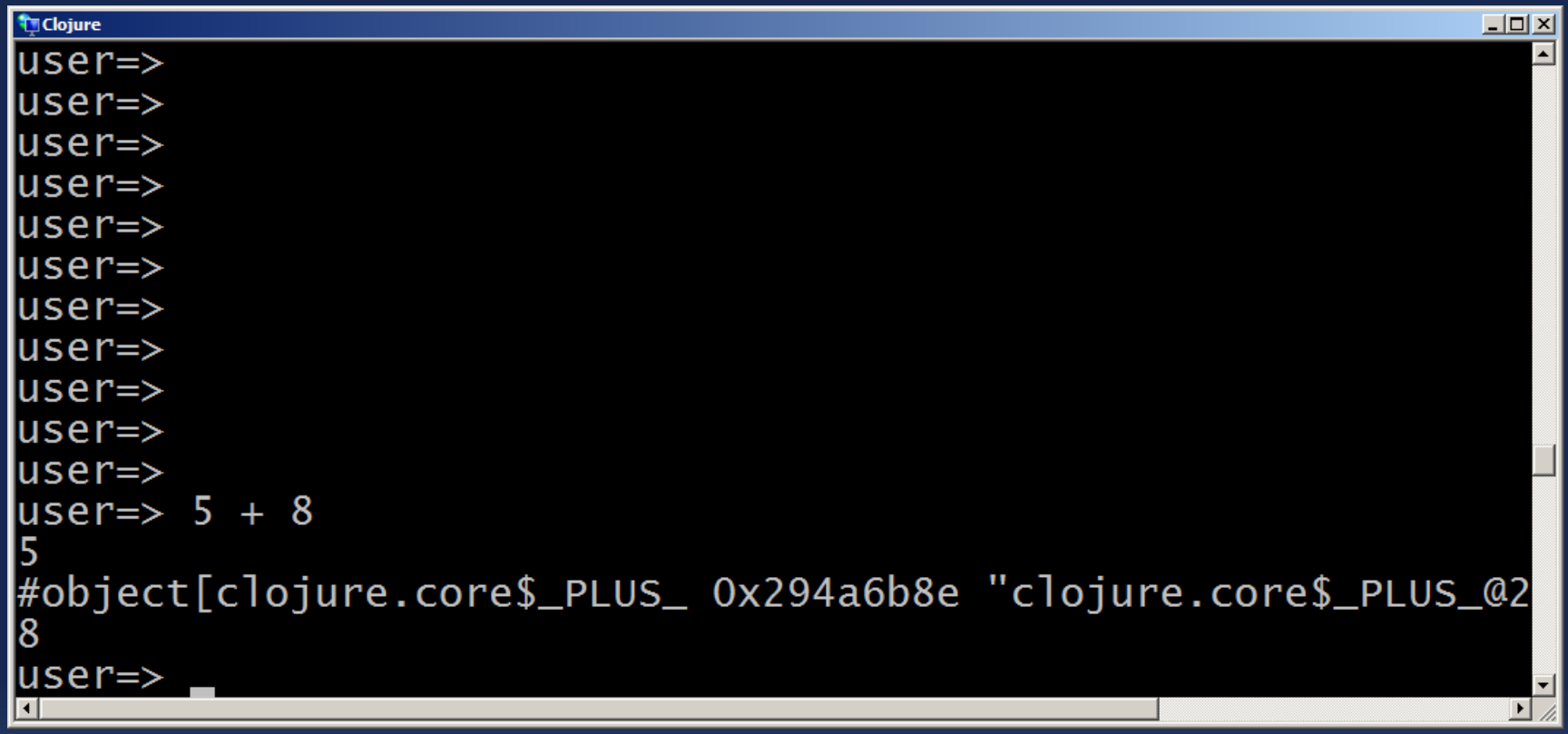


Avaliando múltiplos strings



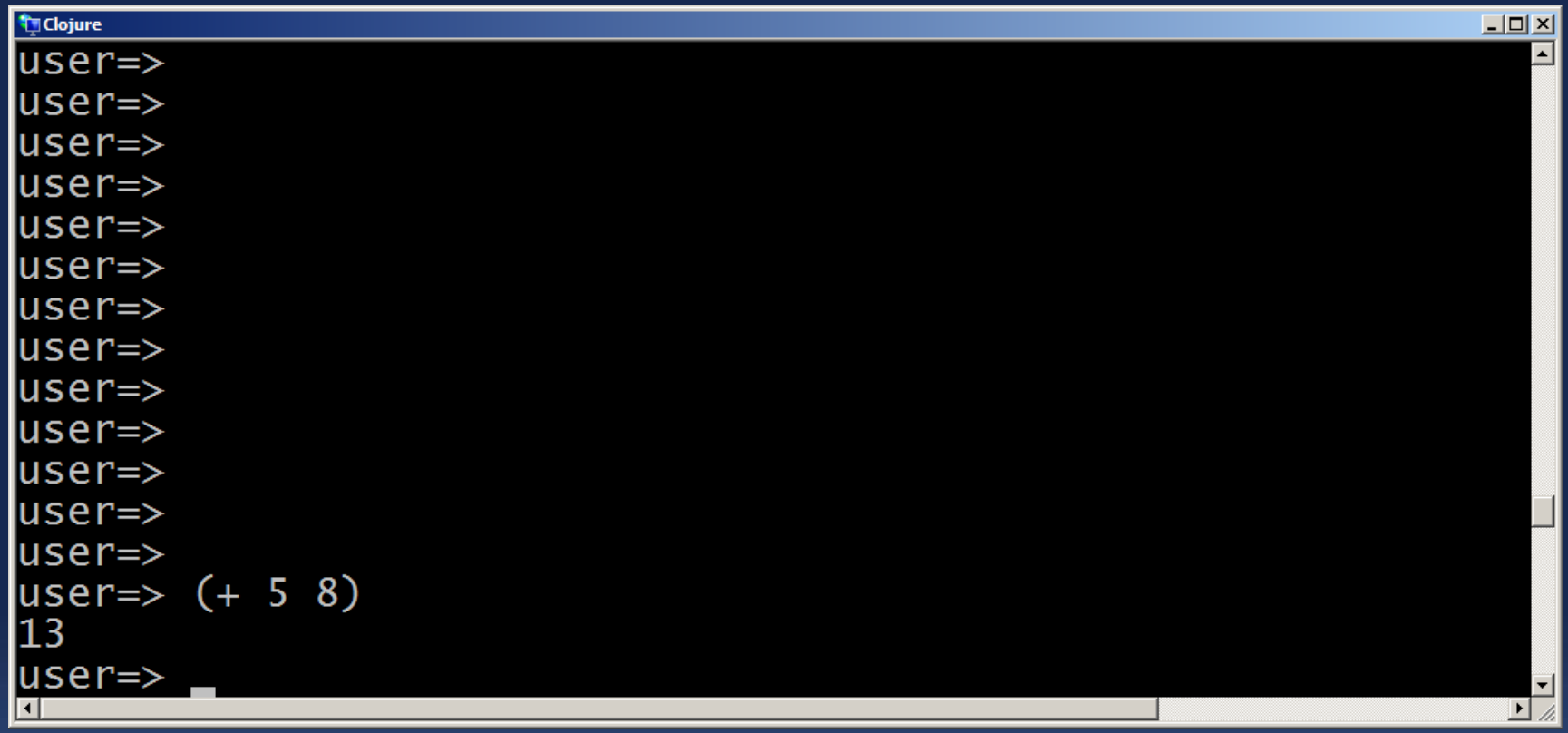
- ✓ Nesse exemplo, **duas** expressões **string** foram avaliadas sequencialmente, e cada qual foi **retornada** em duas linhas separadas;

Avaliando expressões aritméticas

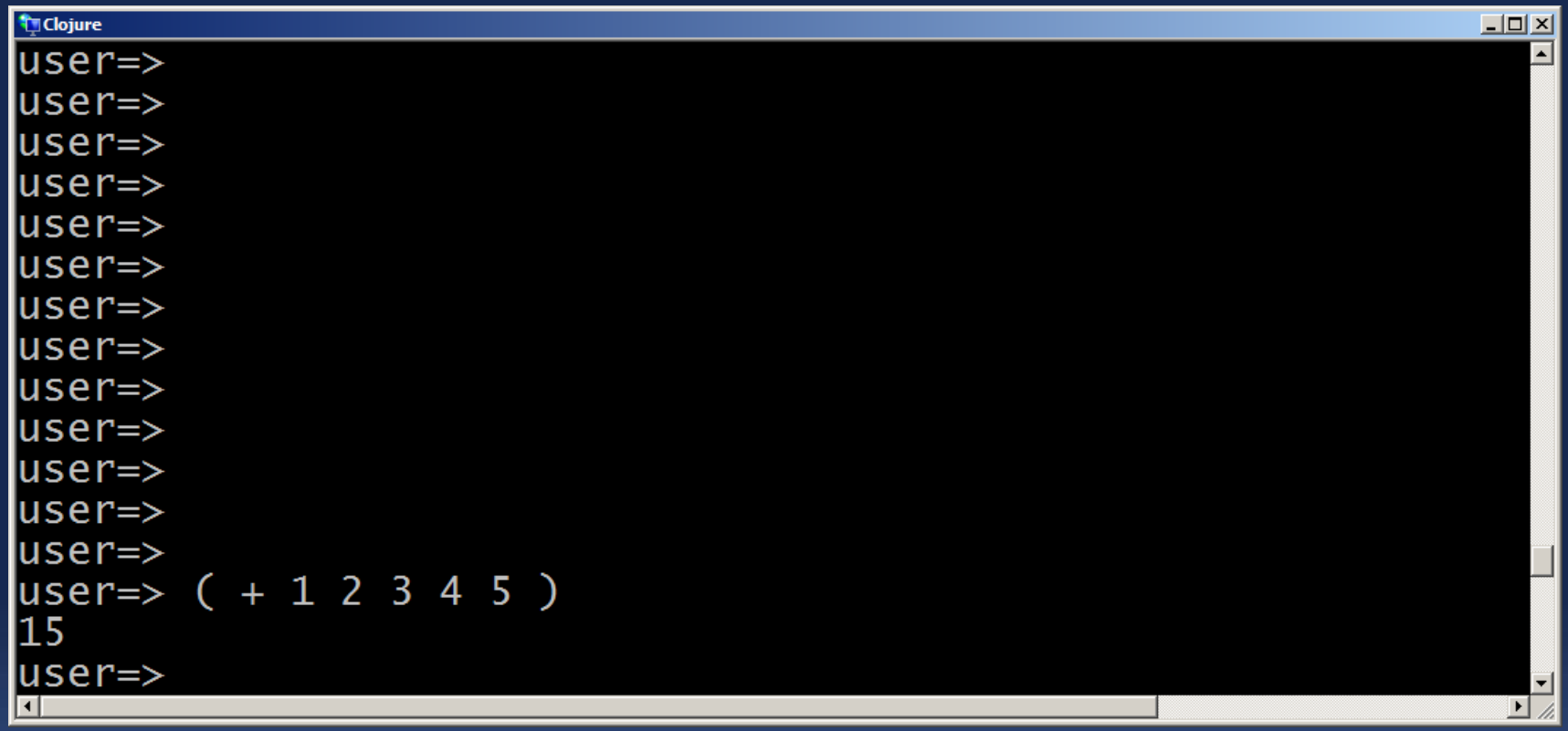


- ✓ Clojure retornou **erro**, pois o primeiro argumento deve ser uma **função**. No caso, o símbolo + está ligado (**bounded**) à uma função e deve ser o primeiro elemento da lista a ser avaliada, seguido pelos operandos (**argumentos**).

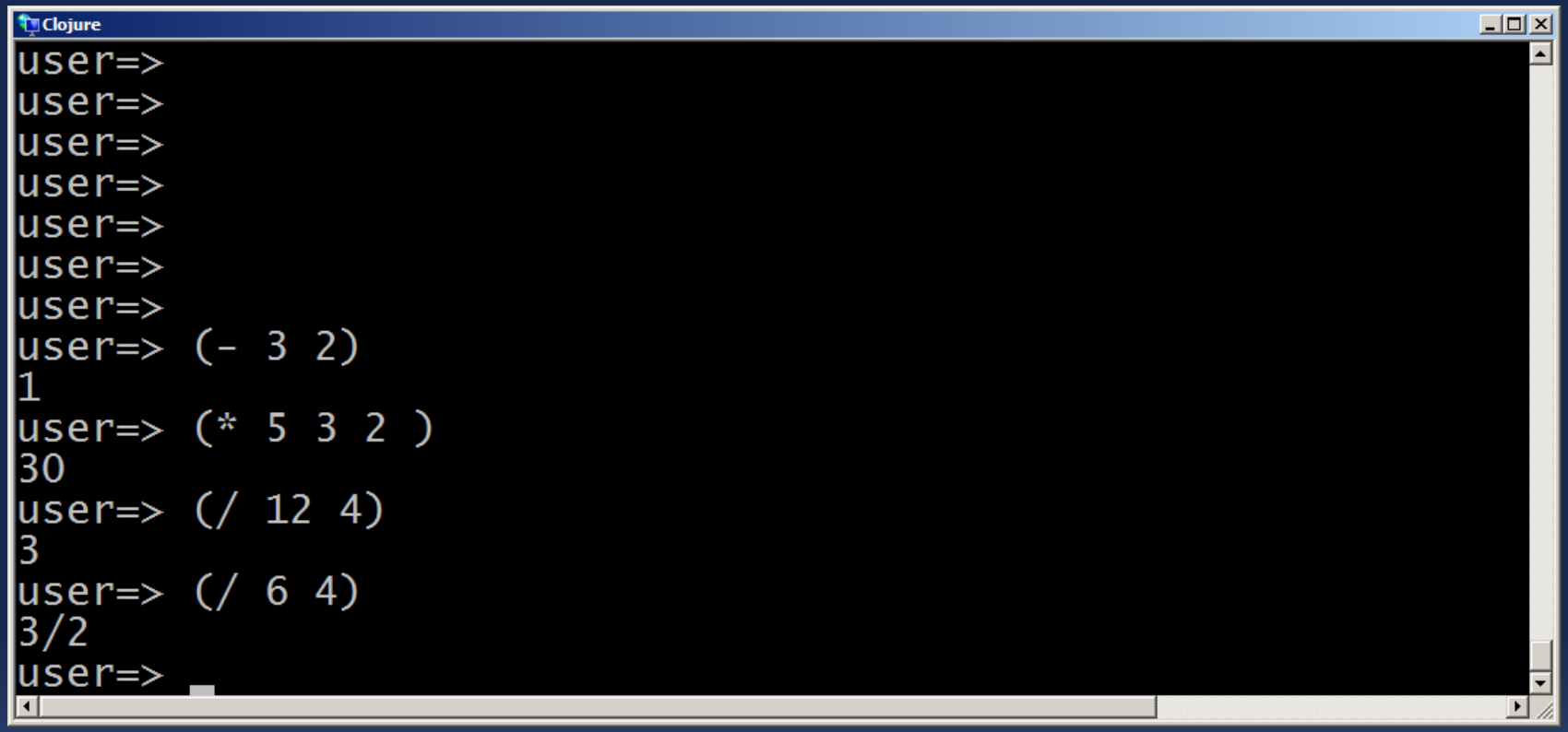
Avaliando expressões aritméticas



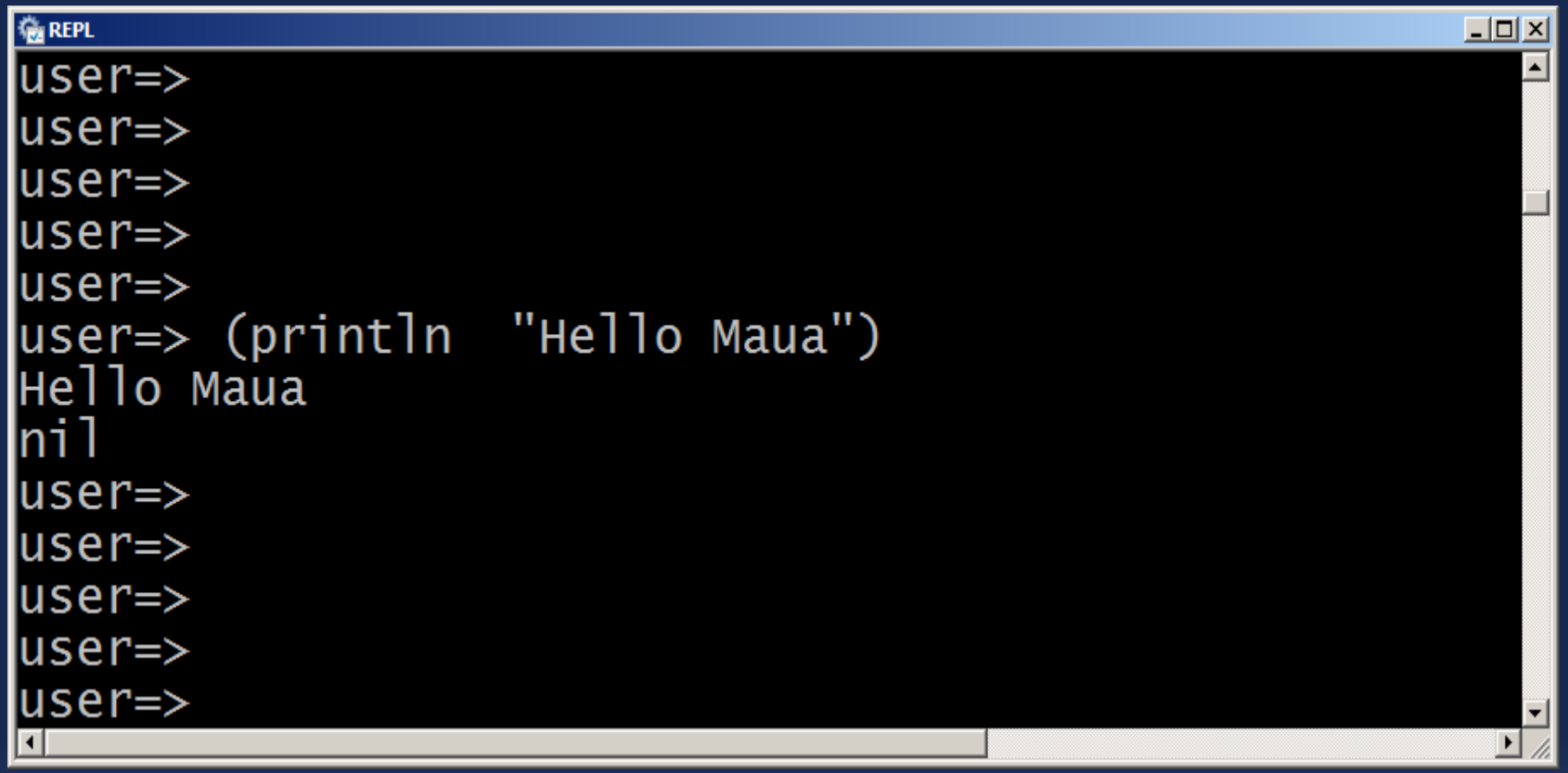
Avaliando expressões aritméticas



Avaliando expressões aritméticas



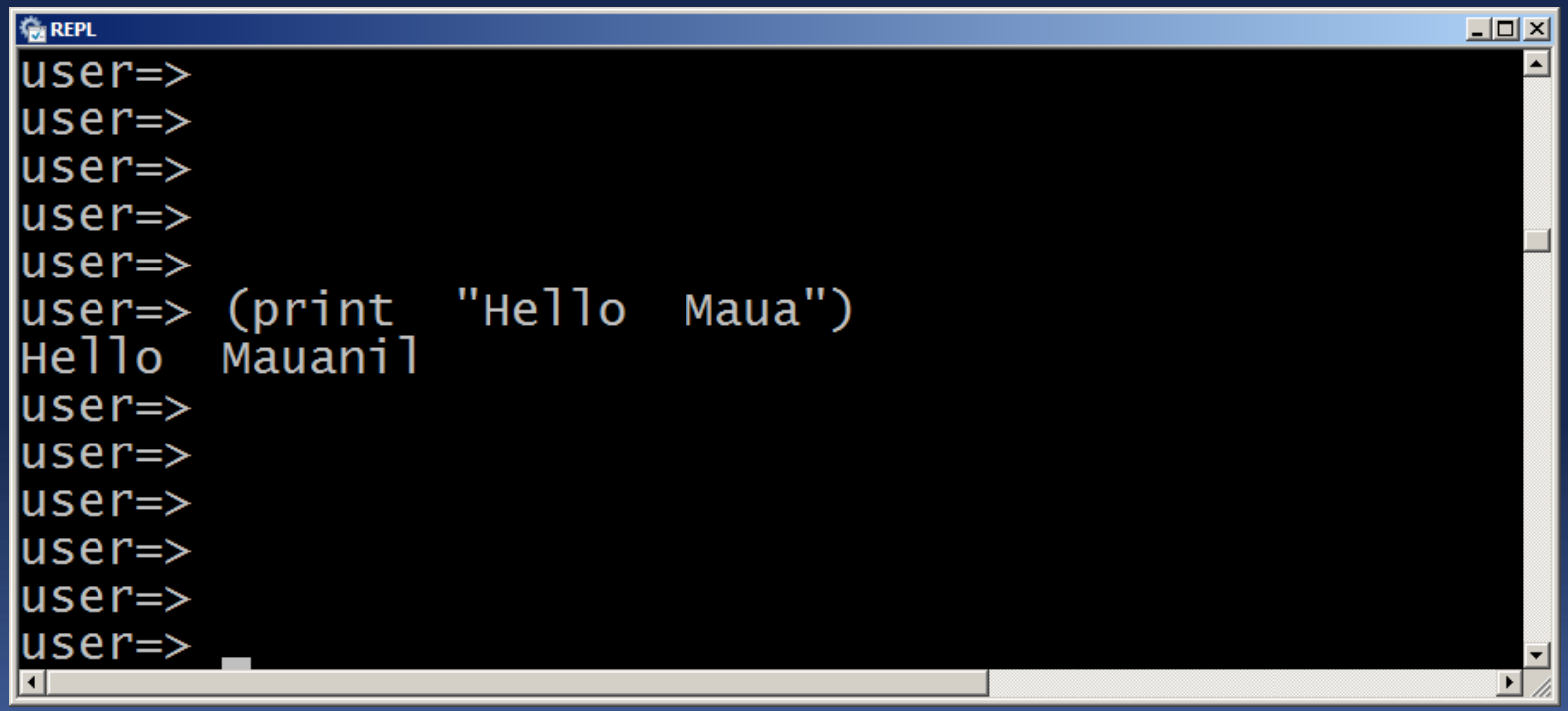
Função println



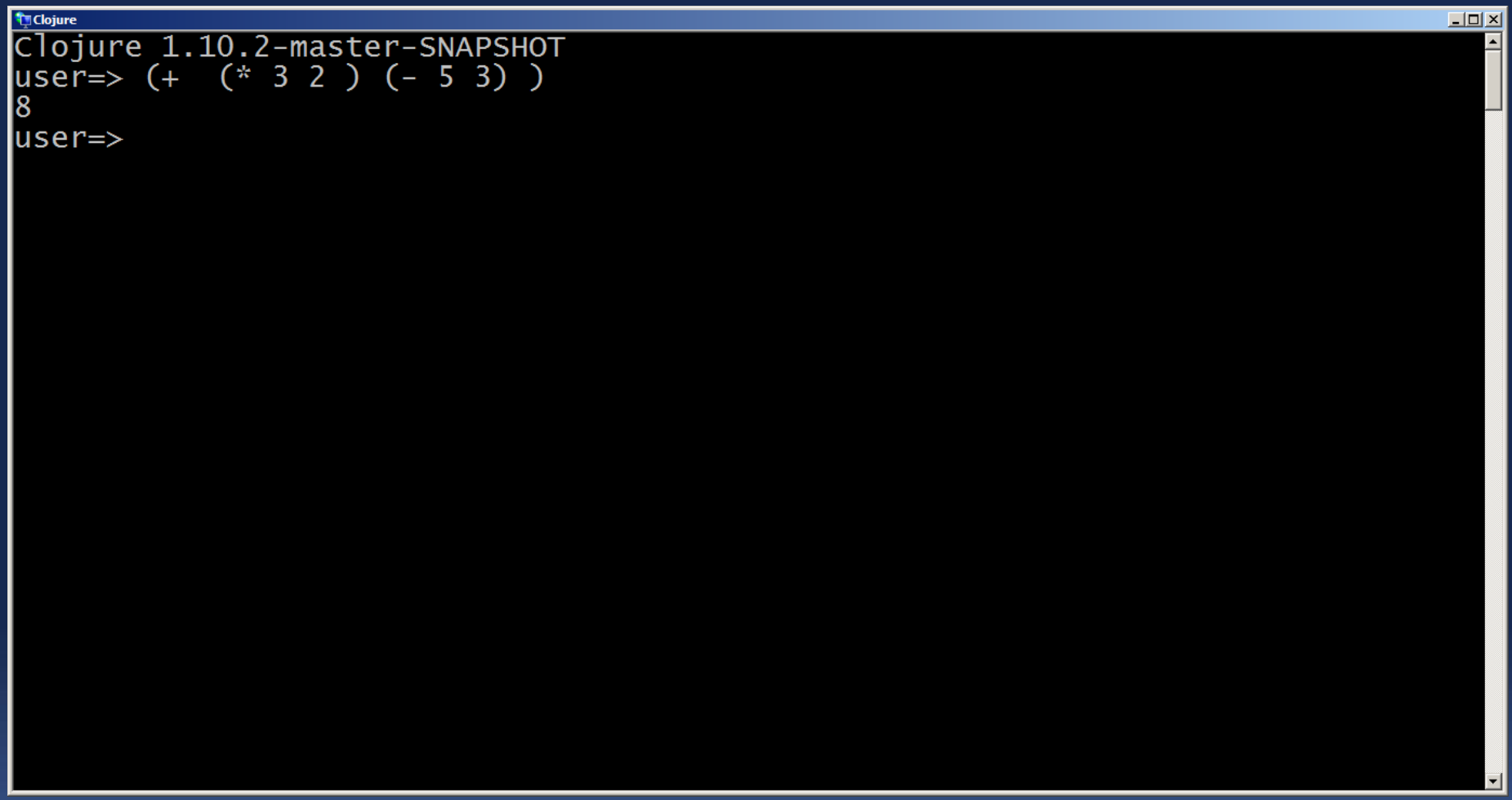
- ✓ O texto impresso pela função println foi um SIDE EFFECT;
- ✓ A função na verdade retornou nil.

nil

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

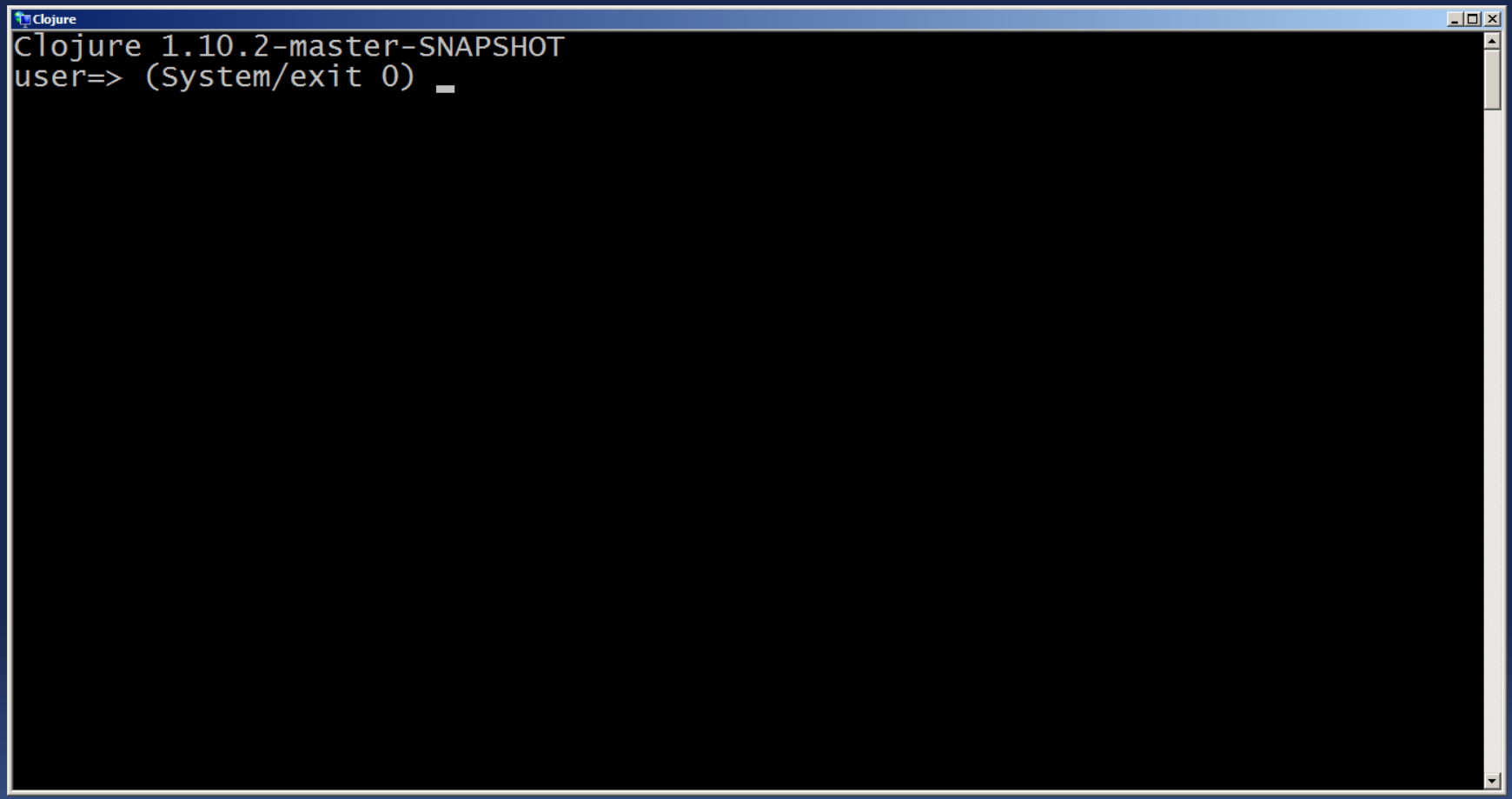


Funções nested

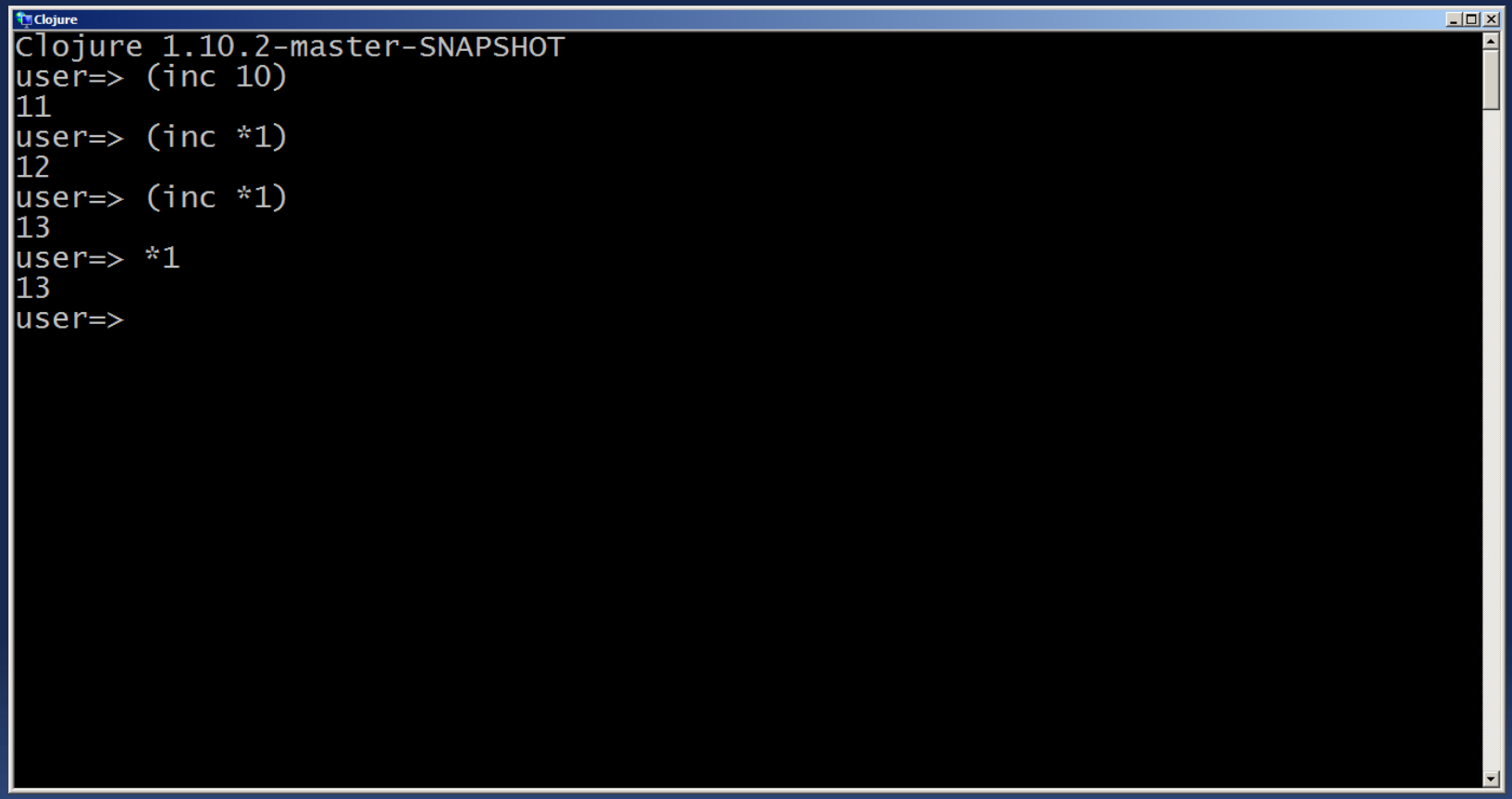


Exit REPL

(System/exit 0)



Operando com a última expressão avaliada => *1

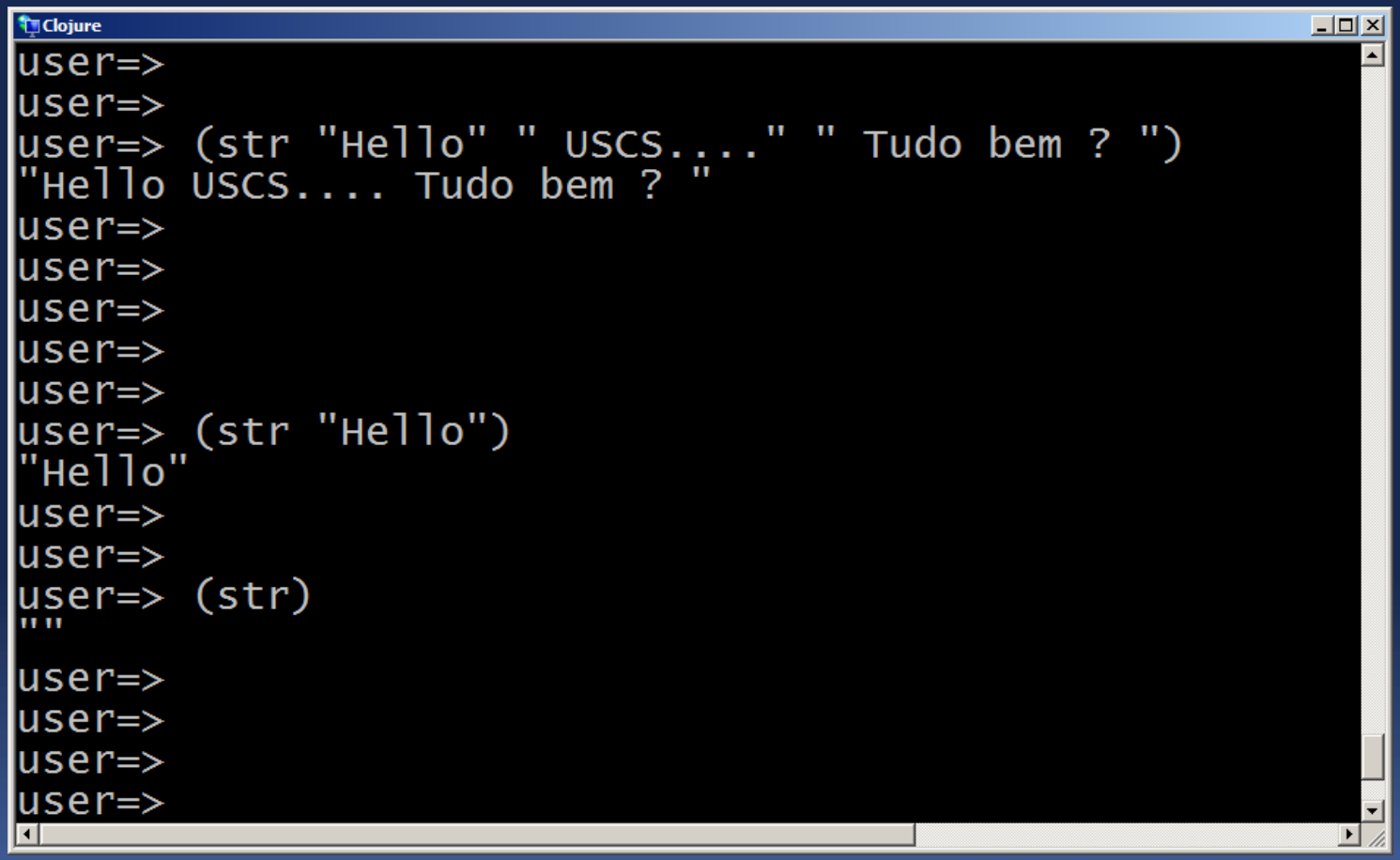


Operando com a variável que contém o resultado da última exceção => *e

```
Clojure
user=>
user=>
user=>
user=>
user=>
user=>
user=> (/ 1 0)
Execution error (ArithmeticException) at user/eval7 (REPL:1).
Divide by zero
user=>
user=>
user=>
user=>
user=> *e
#error {
 :cause "Divide by zero"
 :via
 [{:type java.lang.ArithmeticException
   :message "Divide by zero"
   :at [clojure.lang.Numbers divide "Numbers.java" 188]}]
 :trace
 [[clojure.lang.Numbers divide "Numbers.java" 188]
  [clojure.lang.Numbers divide "Numbers.java" 3901]
  [user$eval7 invokeStatic "NO_SOURCE_FILE" 1]
  [user$eval7 invoke "NO_SOURCE_FILE" 1]
```

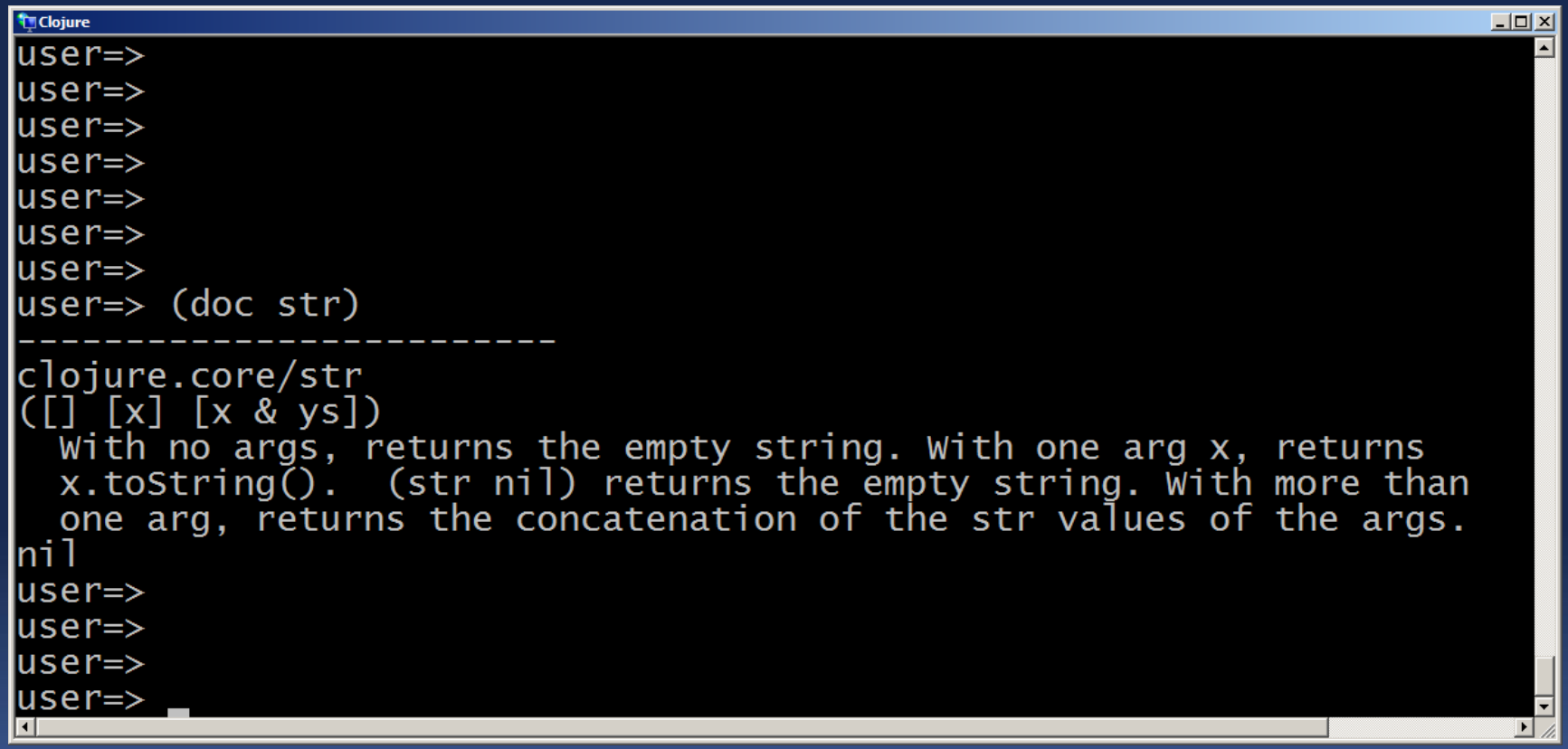
Concatenando Strings - str

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



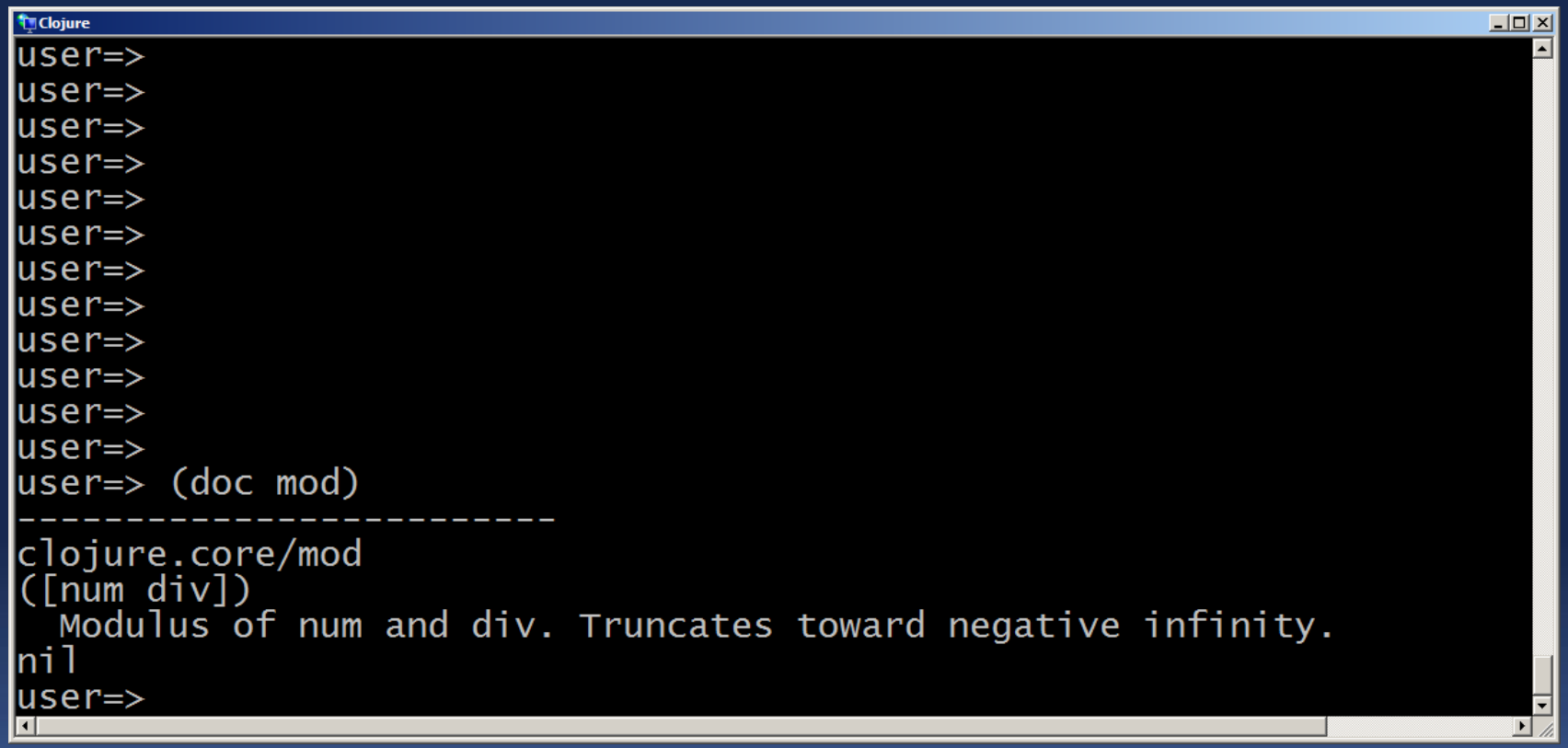
Função doc

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



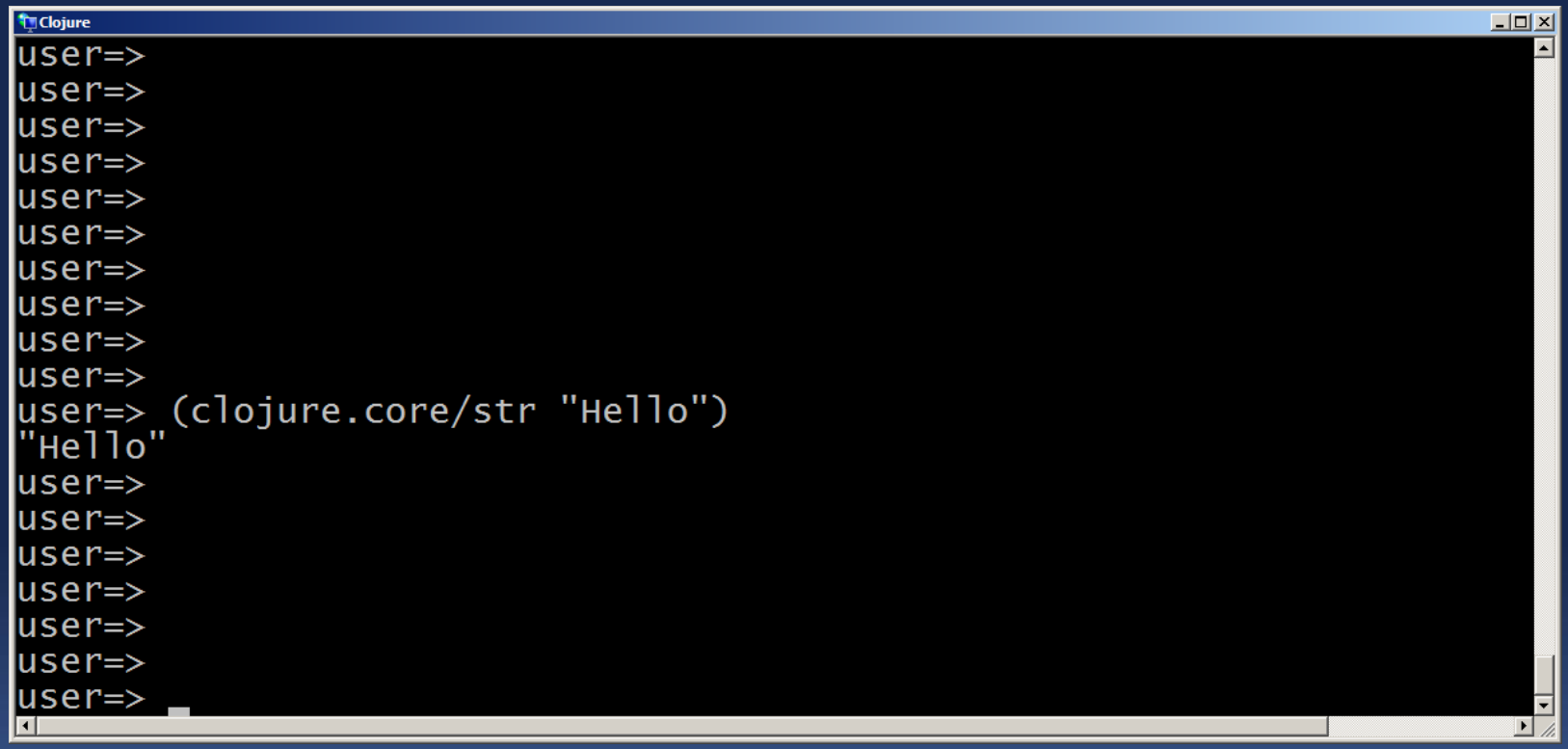
Função doc

- ✓ Exibe o nome completamente qualificado da função (incluindo o **namespace**) na primeira linha, os possíveis **parâmetros** (ou "aridades") na próxima linha e finalmente a descrição da função que se está pesquisando.
- ✓ Exemplo: **(doc mod)**



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")**



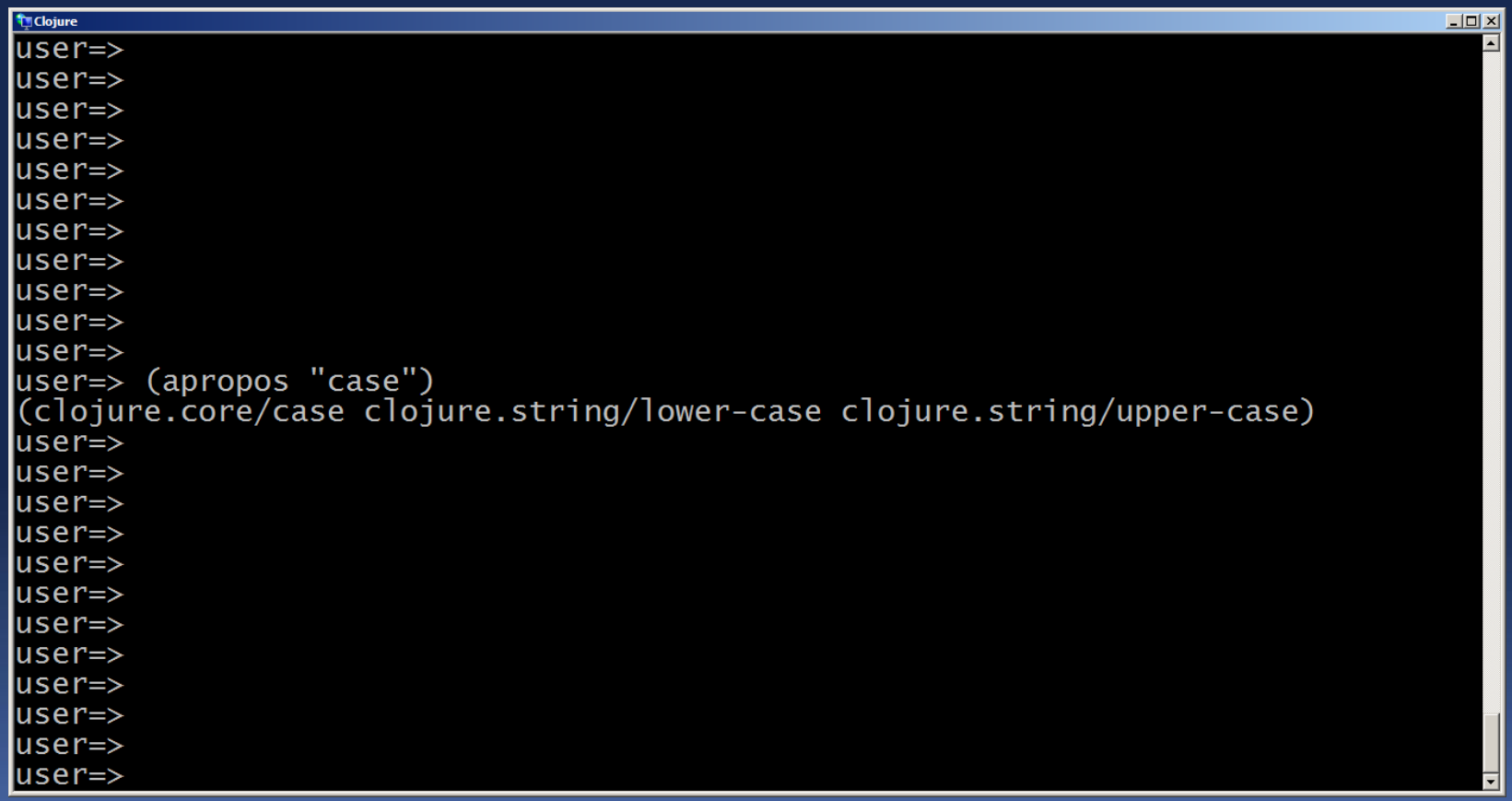
(find-doc)

- ✓ Quando **não** se sabe o nome da função, mas se tem uma ideia de sua descrição ou nome que pode conter, pode-se usar a função **find-doc**;

```
Clojure
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> (find-doc "float")
-------------------------
clojure.pprint/float-parts
([f])
  Take care of leading and trailing zeros in decomposed floats
-------------------------
clojure.pprint/float-parts-base
([f])
  Produce string parts for the mantissa (normalized 1-9) and exponent
-------------------------
clojure.core/aset-float
([array idx val] [array idx idx2 & idxv])
  Sets the value at the index/indices. Works on arrays of float. Returns val.
-------------------------
clojure.core/float
([x])
  Coerce to float
```

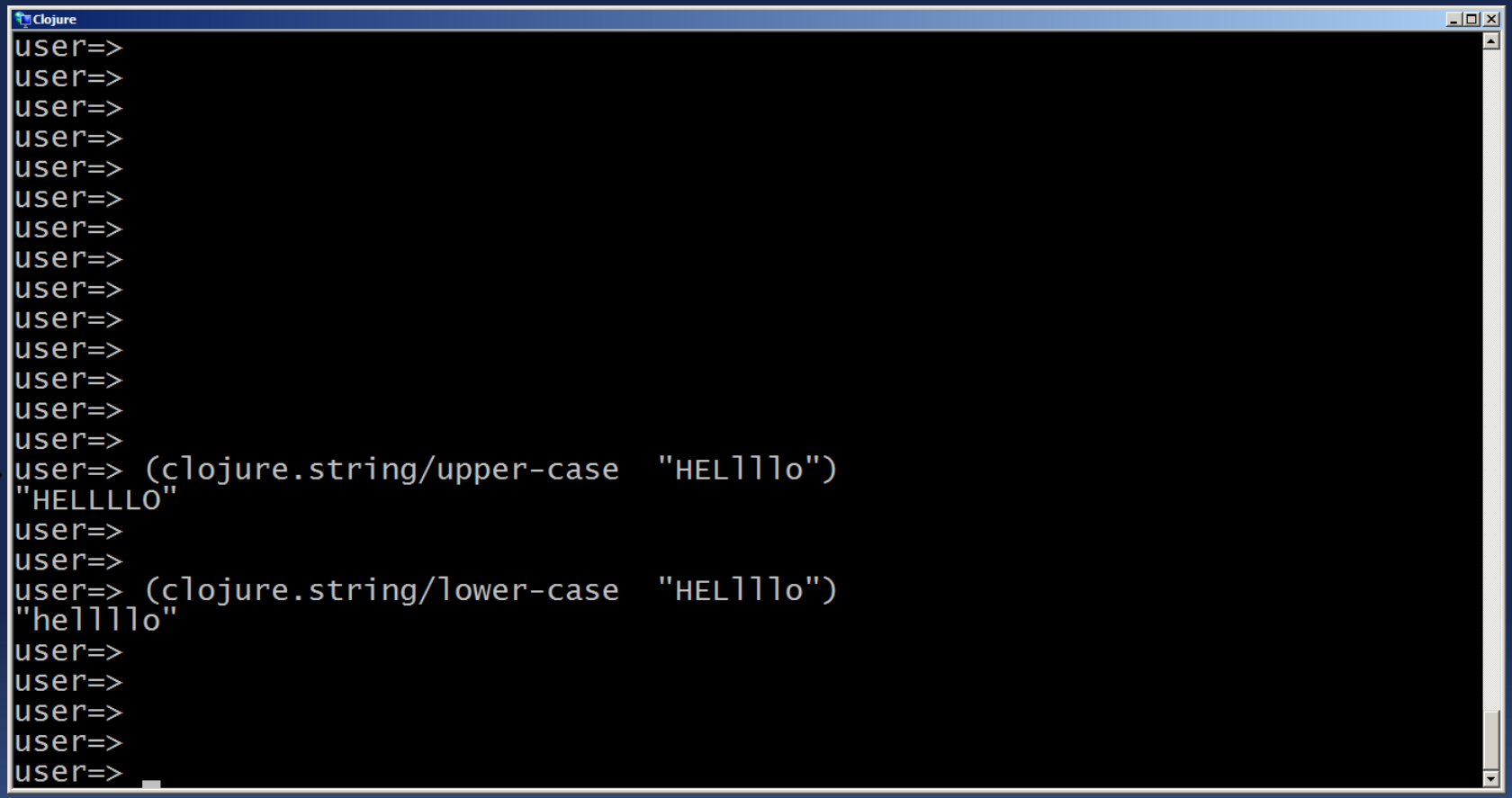
Função apropos

- ✓ Usada para pesquisar funções pelo nome, resultando assim uma saída mais sucinta. Por exemplo, pesquisar uma função que transforma um dado string em caracteres maiúsculos ou minúsculos;
- ✓ Observamos na saída que **lower-case** e **upper-case** estão definidas no namespace **clojure.string**.



lower-case e upper-case

- ✓ Observamos na saída que **lower-case** e **upper-case** estão definidas no namespace **clojure.string**.

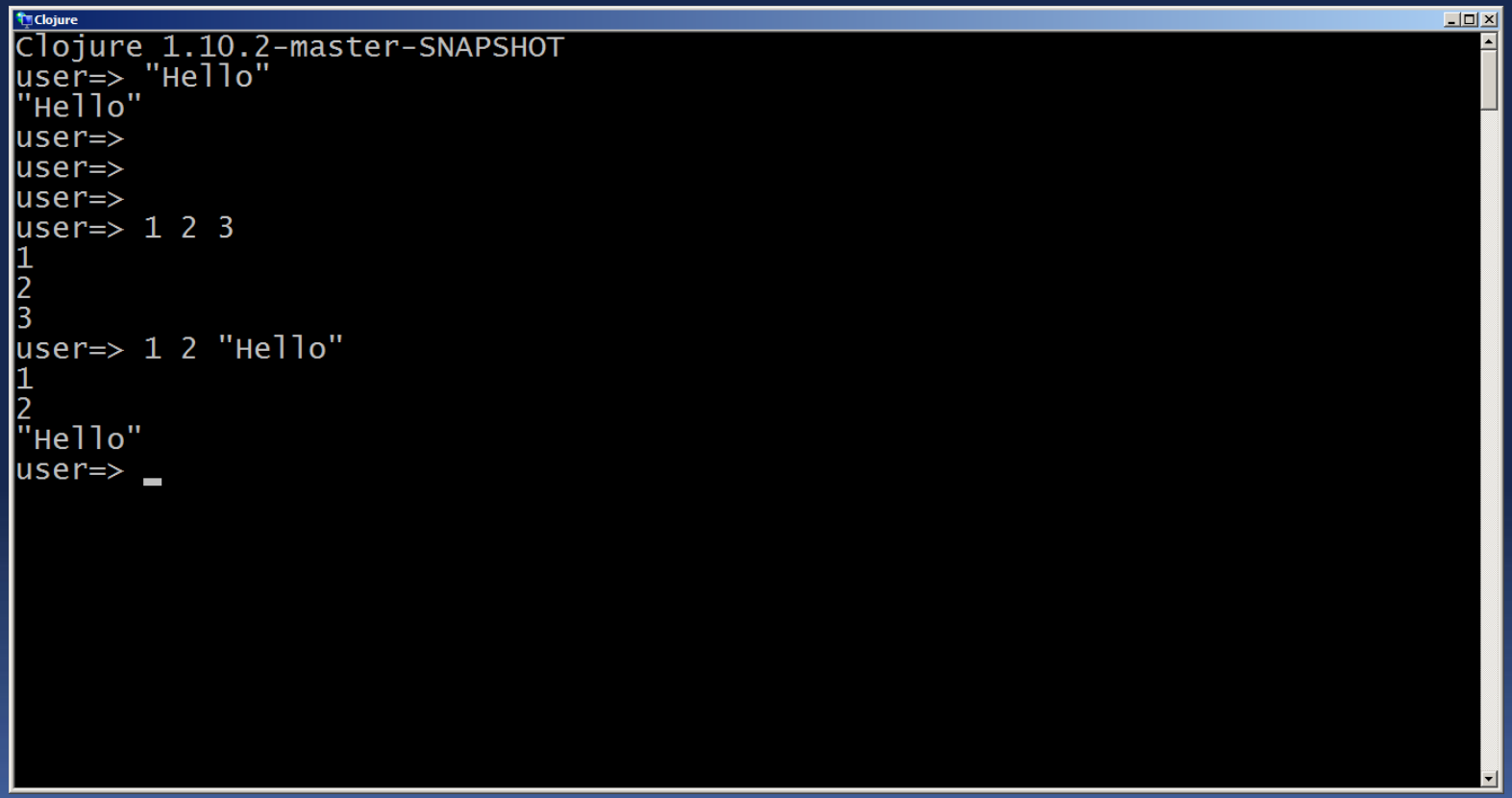


Atividade 1

- ✓ Abra o **REPL**;
- ✓ Imprima a mensagem: "**Eu não tenho medo de parênteses**"
- ✓ Some 1, 2 e 3 e multiplique o resultado por 10 menos 3;
- ✓ Imprima a mensagem "Atividade Concluída...";
- ✓ Encerre o **REPL**.

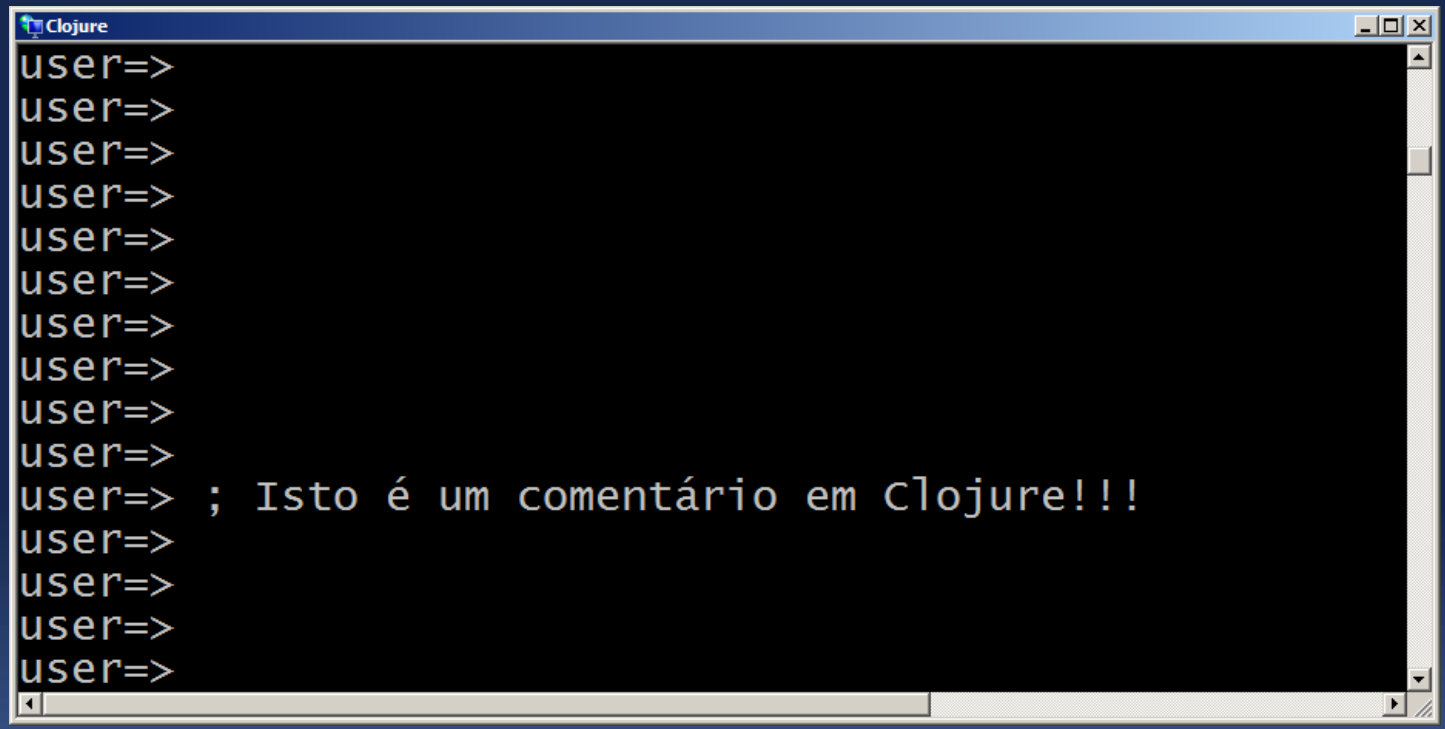
Avaliação de código em Clojure

- ✓ Em Clojure, literais são expressões válidas e ao serem avaliadas retornam os próprios literais.



Comentários em Clojure

- ✓ Qualquer linha iniciada com ;



Funções em Clojure

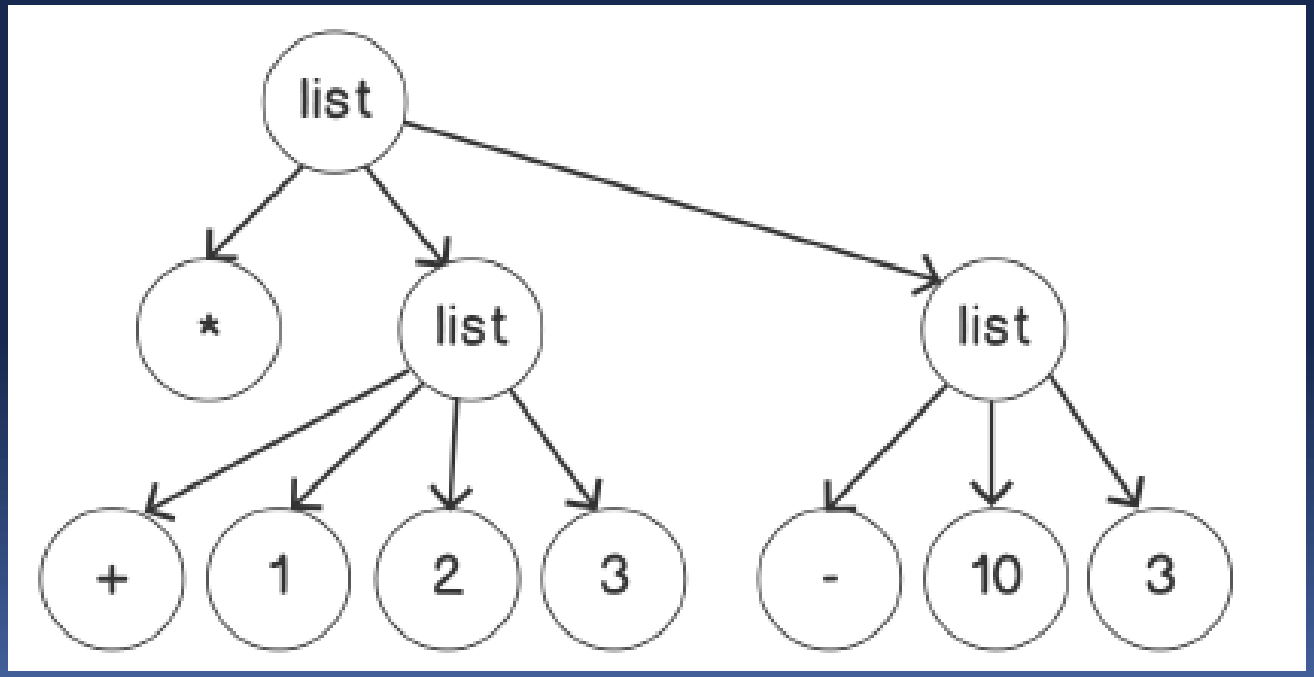
- ✓ São invocadas com a seguinte estrutura:

```
; (operator operand-1 operand-2 operand-3 …)
; for example:
user=> (* 2 3 4)
24
```

Expressões Simbólicas (s-expression)

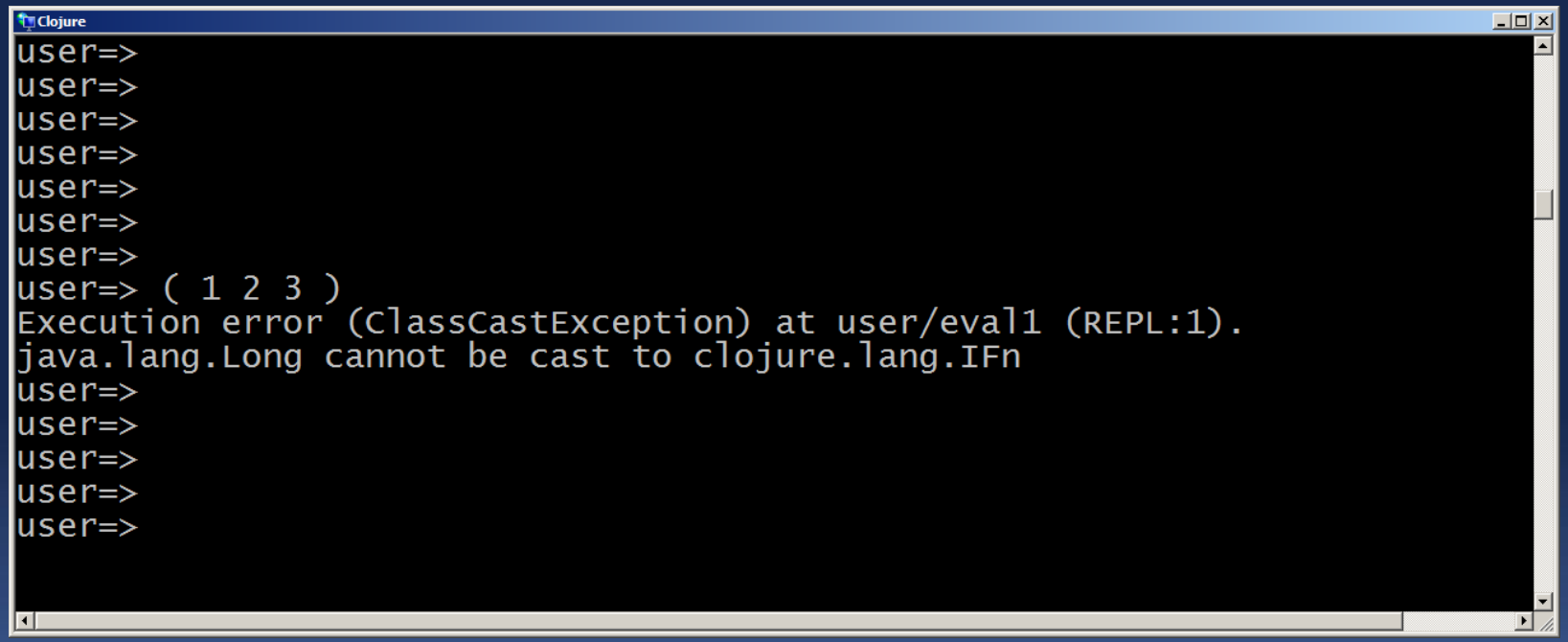
- ✓ Podem ser visualizadas em uma árvore

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



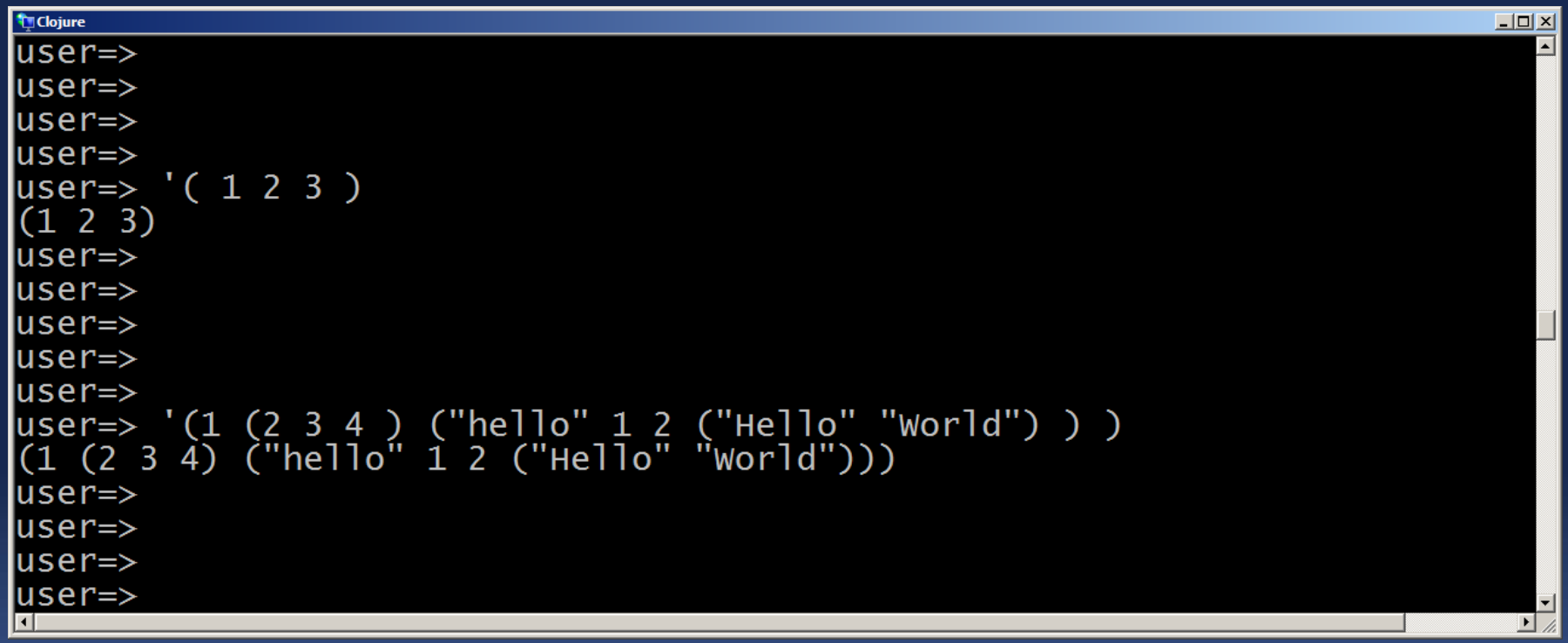
Listas

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



Listas como dados

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



Código Clojure

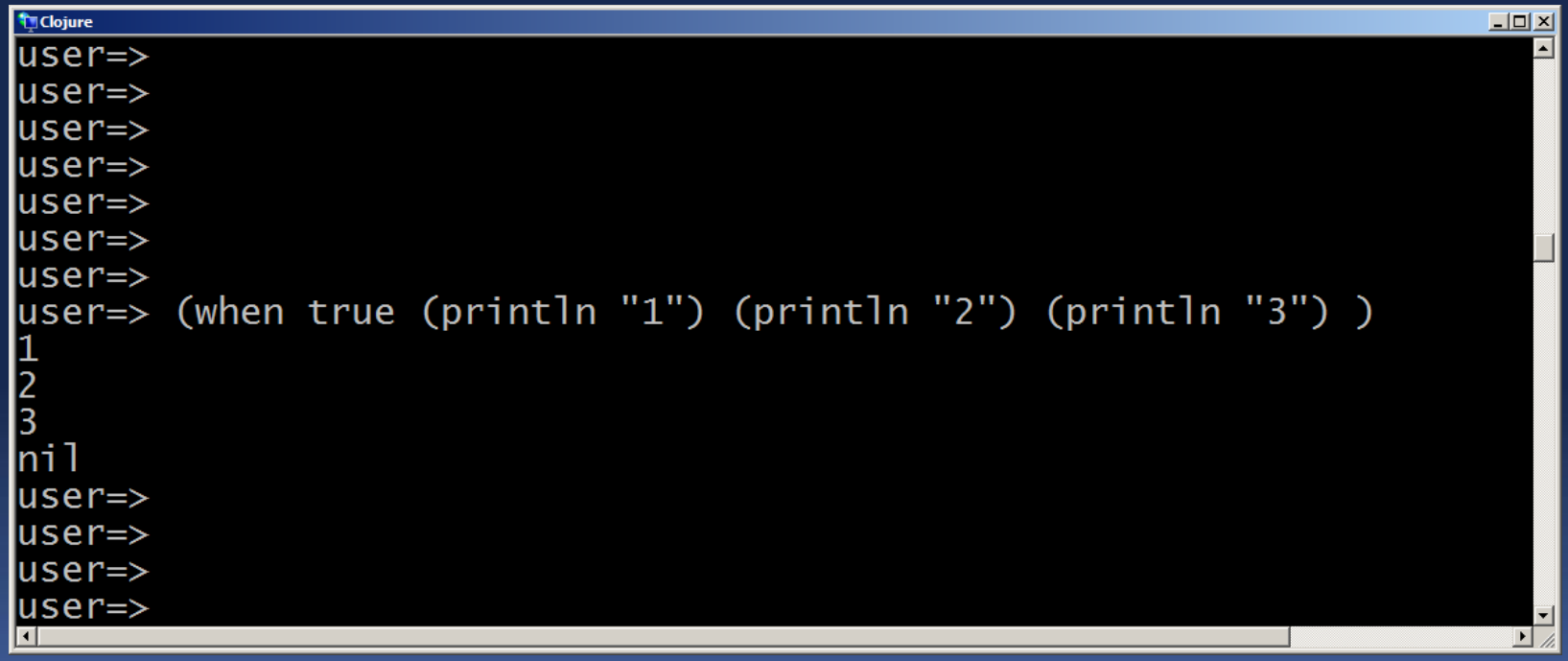
- ✓ Código **Clojure** é constituído por estruturas de dados;
- ✓ Nossos programas podem gerar essas estruturas de dados;
- ✓ Assim, num jargão **Lisp**, programas em tempo de execução podem gerar código;
- ✓ Esse conceito é conhecido **por Meta-Programação**.

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;

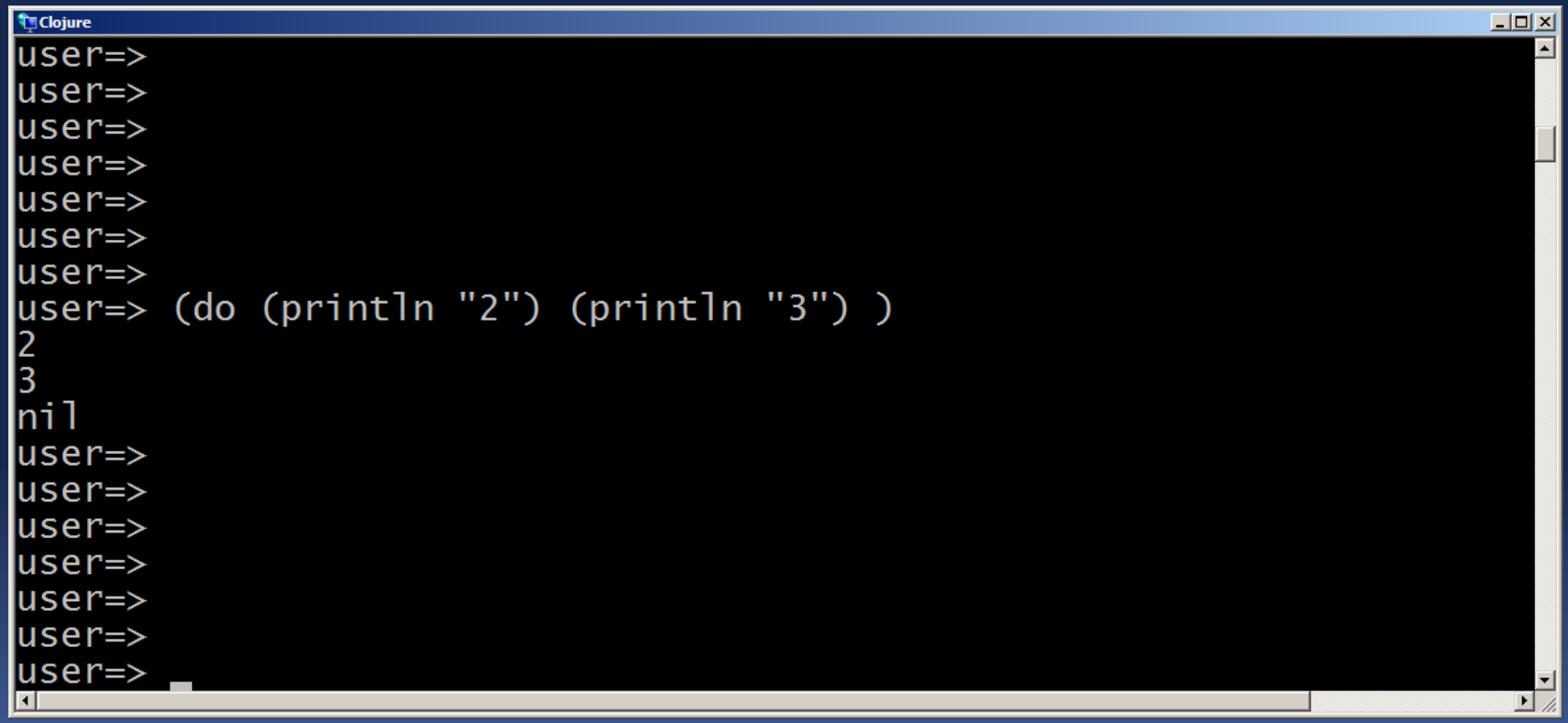
Macro – when

- ✓ Usado quando somente estamos interessados no caso em que a condição lógica for verdadeira;
- ✓ É semelhante a um **if**, mas não contém ramificação **else** e pode ser tratado como um "**do**" **implícito**.



Special Forms - do

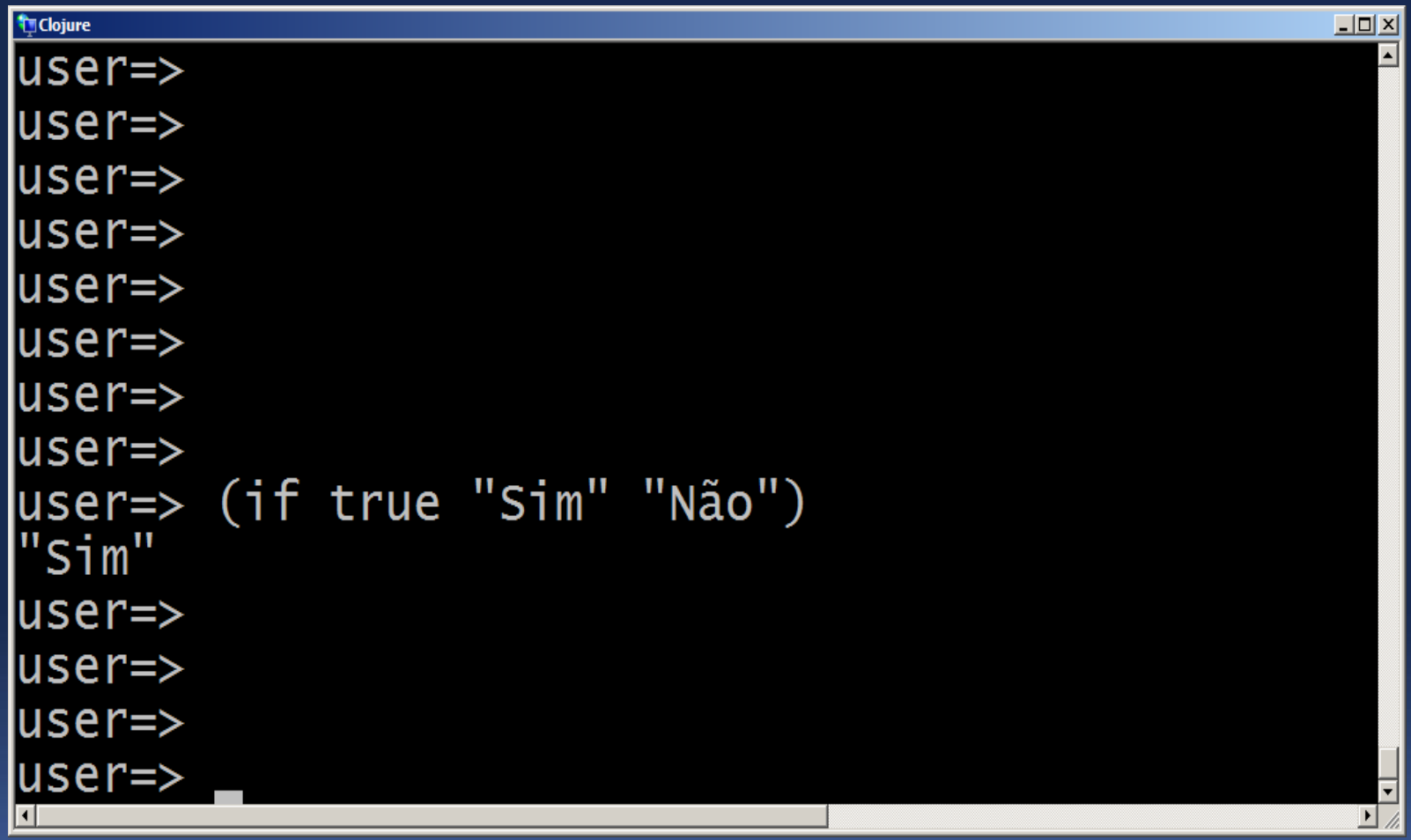
- ✓ Usado para executar uma série de expressões e retornar o valor da última expressão.



Atividade 2

- ✓ Abra o **REPL**;
- ✓ Escreva a função if que retorna "Sim" se a condição for verdadeira e "Não" caso contrário.
- ✓ Encerre o **REPL**.

Atividade 2

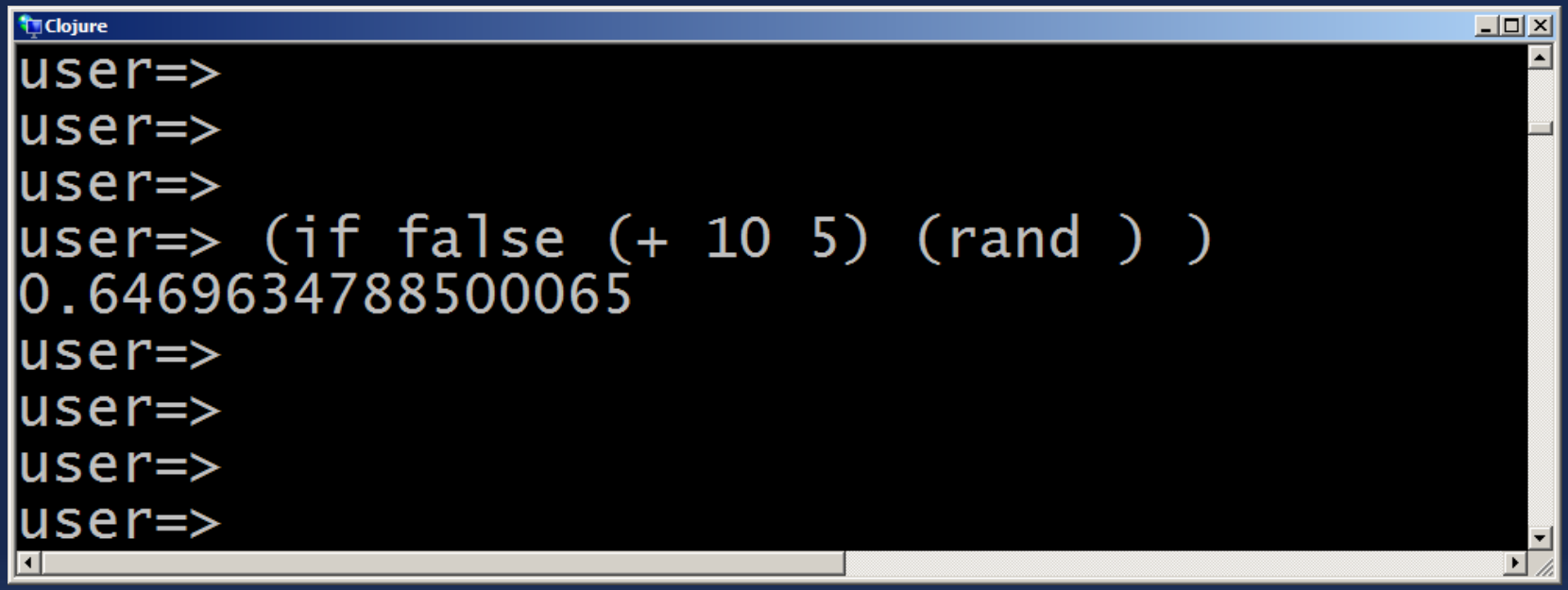

Clojure
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> (if true "Sim" "Não")
"Sim"
user=>
user=>
user=>
user=>

Atividade 3

- ✓ Abra o **REPL**;
- ✓ Escreva a função if que retorna a soma de 10 com 5 se a condição for falsa. Caso contrário retorna um número randômico.
- ✓ Encerre o **REPL**.

Dica. A função rand retorna um número rand.

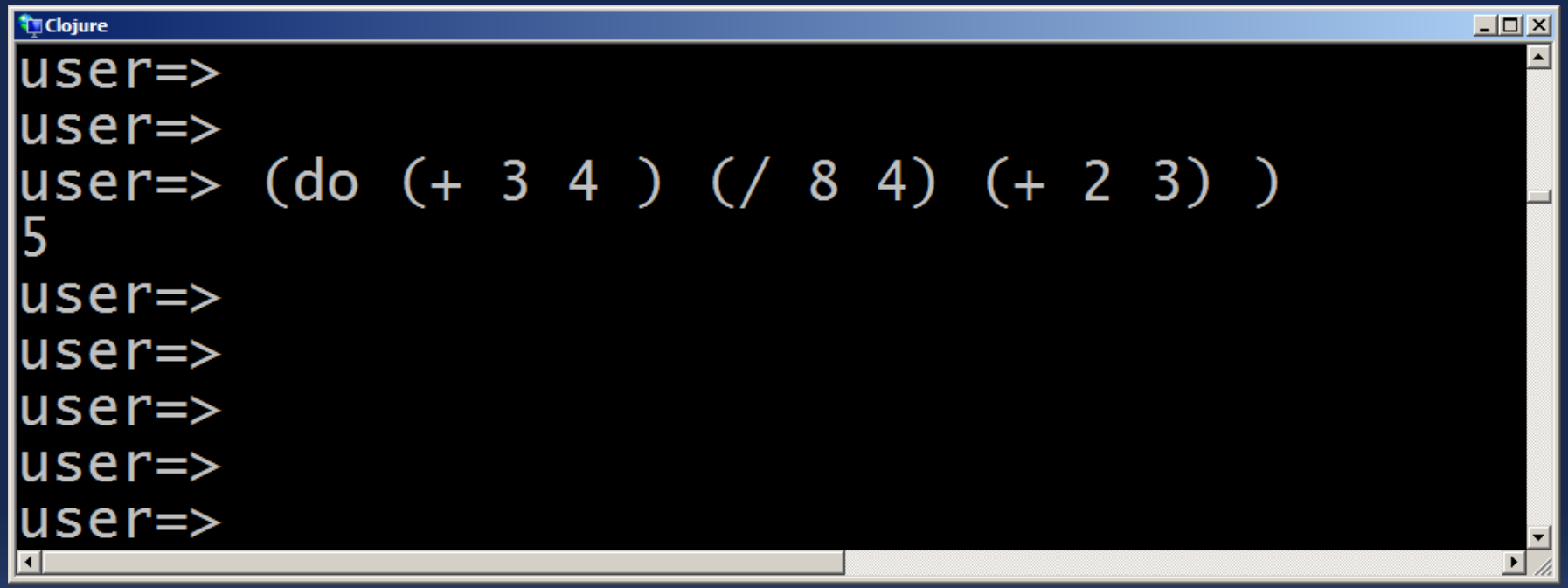
Atividade 3


Clojure
user=>
user=>
user=>
user=> (if false (+ 10 5) (rand))
0.6469634788500065
user=>
user=>
user=>
user=>

Atividade 4

- ✓ Abra o **REPL**;
- ✓ Escreva a função "**do**" que executa sequencialmente as seguintes avaliações:
  - ✓ Multiplica 3 por 4;
  - ✓ Divide 8 por 4;
  - ✓ Soma 2 com 3.
- ✓ Encerre o **REPL**.

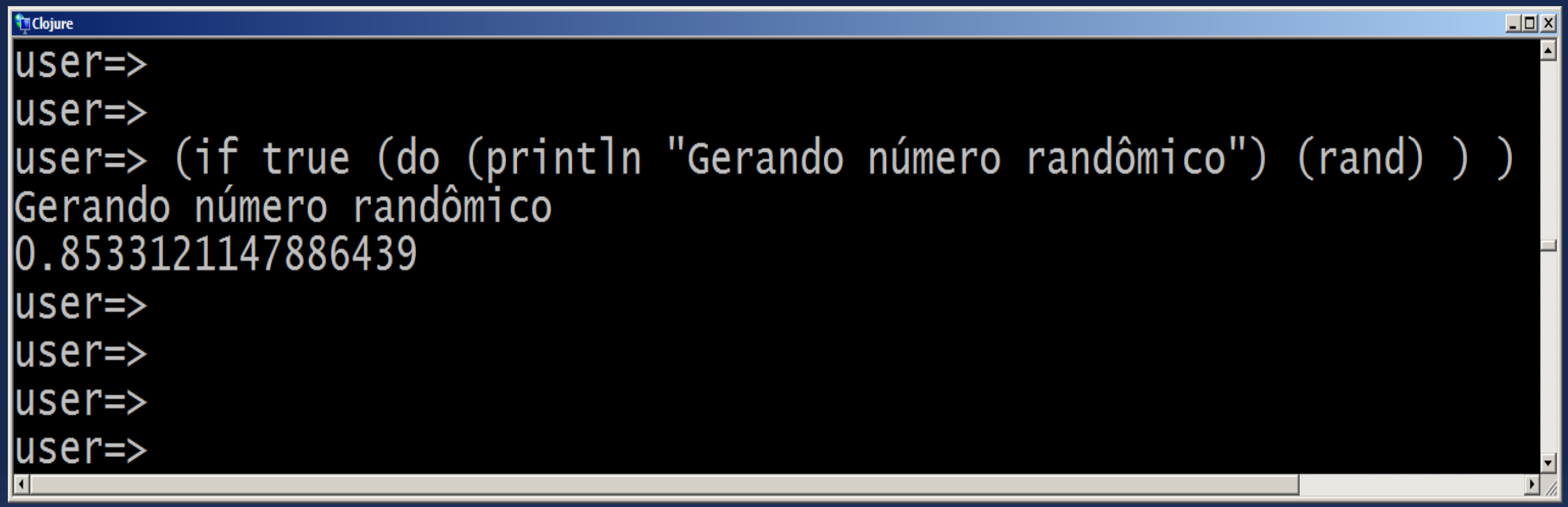
Atividade 4



Atividade 5

- ✓ Abra o **REPL**;
- ✓ Escreva a função "**if**" que imprime a mensagem "Gerando um número randômico" e exibe o número gerado, caso a condição seja verdadeira;
- ✓ Encerre o **REPL**.

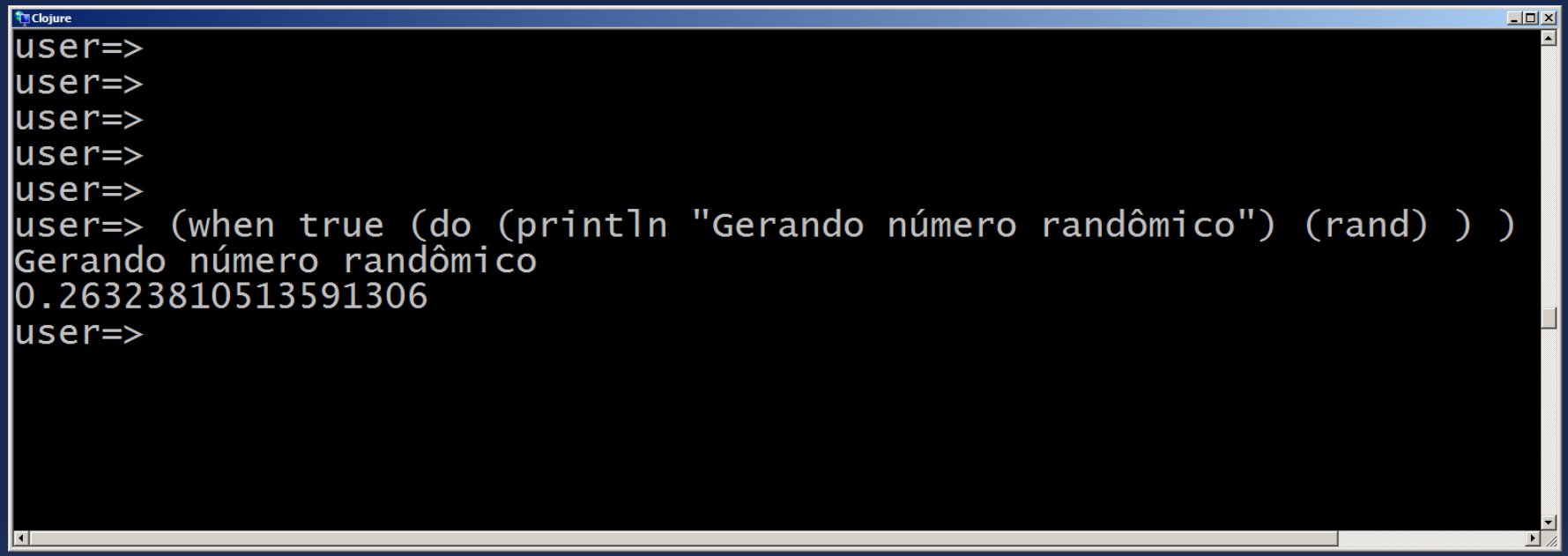
Atividade 5


Clojure
user=>
user=>
user=> (if true (do (println "Gerando número randômico") (rand)))
Gerando número randômico
0.8533121147886439
user=>
user=>
user=>
user=>

Atividade 6

- ✓ Abra o **REPL**;
- ✓ Escreva a função "**when**" que imprime a mensagem "Gerando um número randômico" e exibe o número gerado, caso a condição seja verdadeira;
- ✓ Encerre o **REPL**.

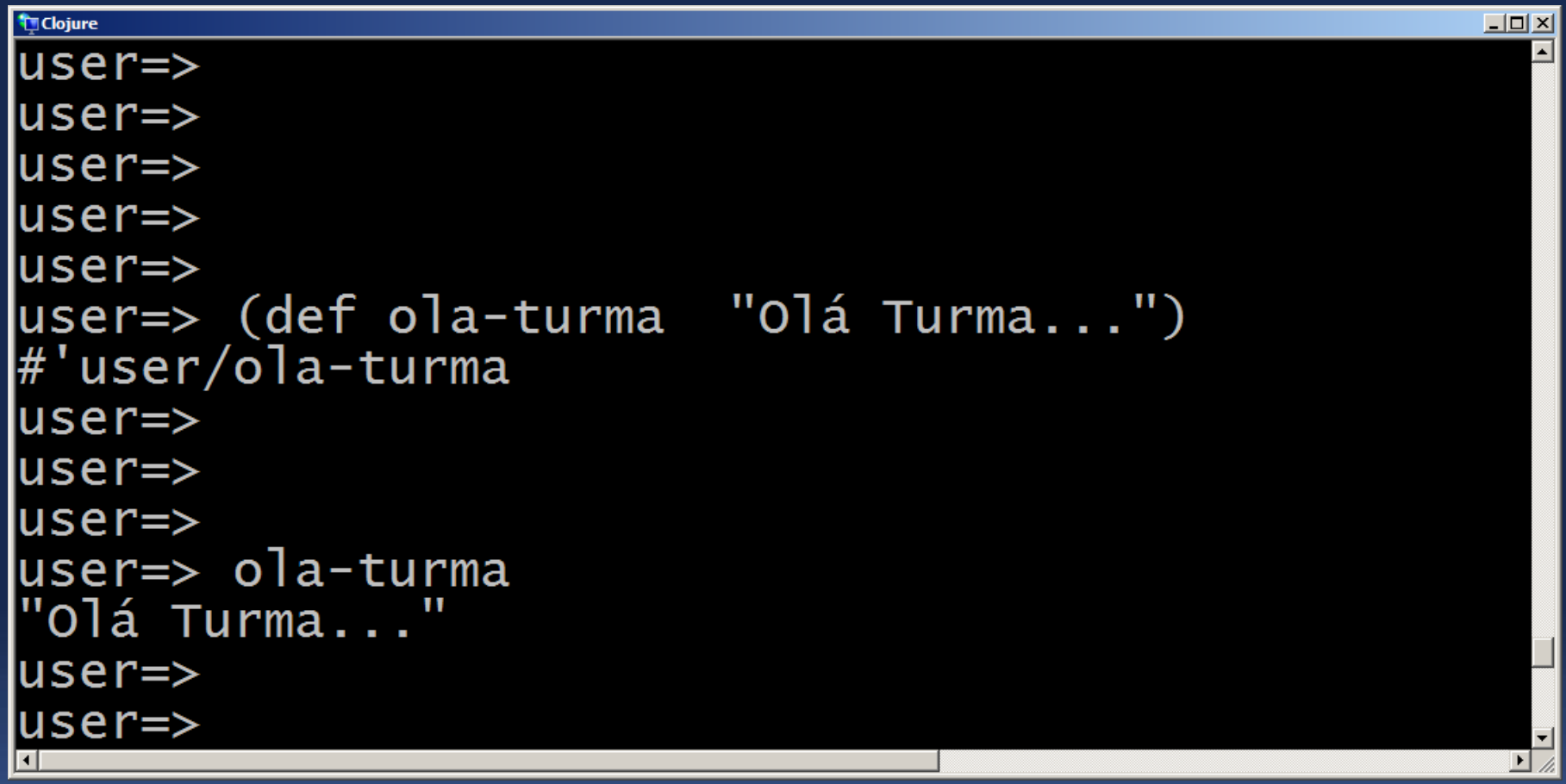
Atividade 6


Clojure
user=>
user=>
user=>
user=>
user=>
user=> (when true (do (println "Gerando número randômico") (rand)))
Gerando número randômico
0.2632381051359306
user=>

Bindings

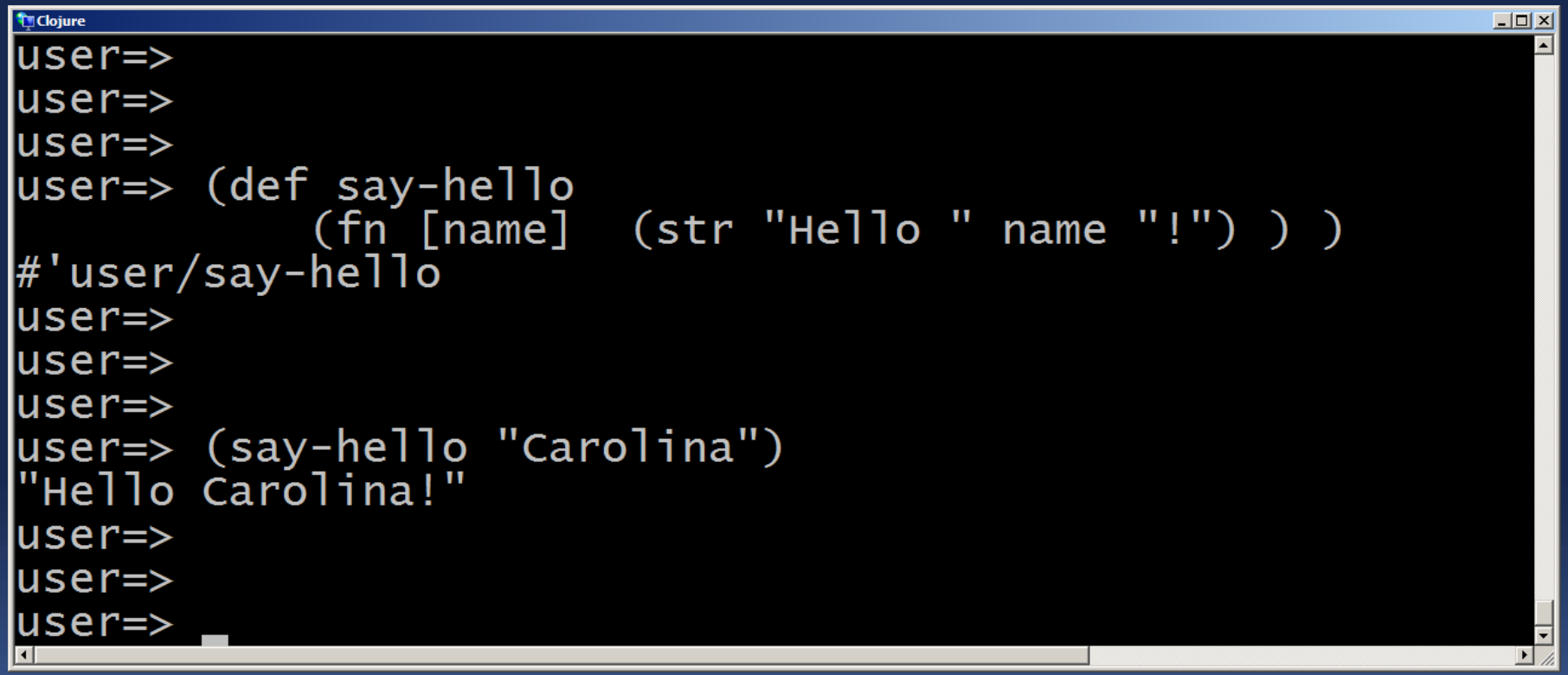
- ✓ Em **Clojure**, **Binding** significa **ligação** de símbolos à valores;
- ✓ O termo **Binding** é preferível ao invés de atribuição, pois essas ligações quase sempre são feitas uma única vez;
- ✓ Diferentemente de atribuição em variáveis que, na grande maioria das linguagens, ocorrem com frequência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

Macro - def


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

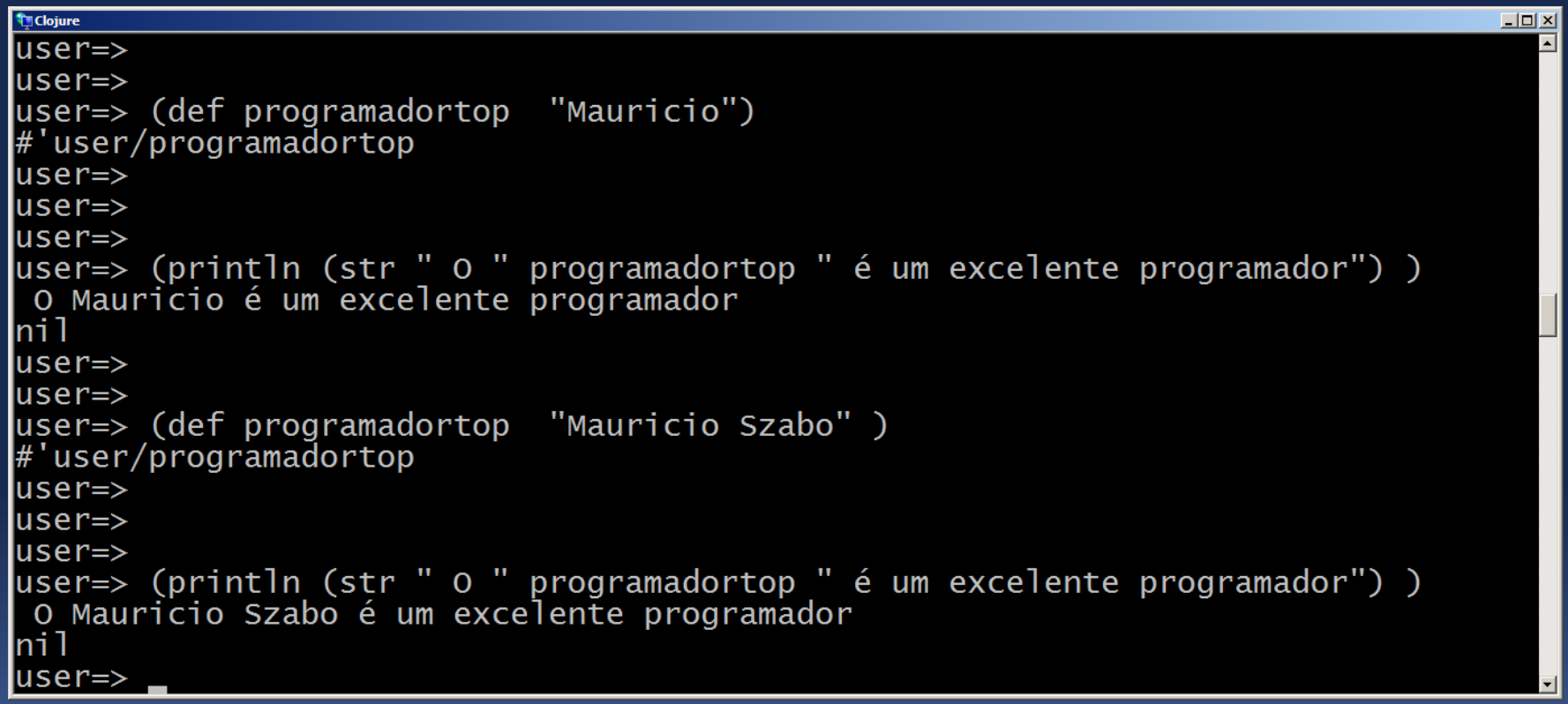
Macro - def

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



Macro - def

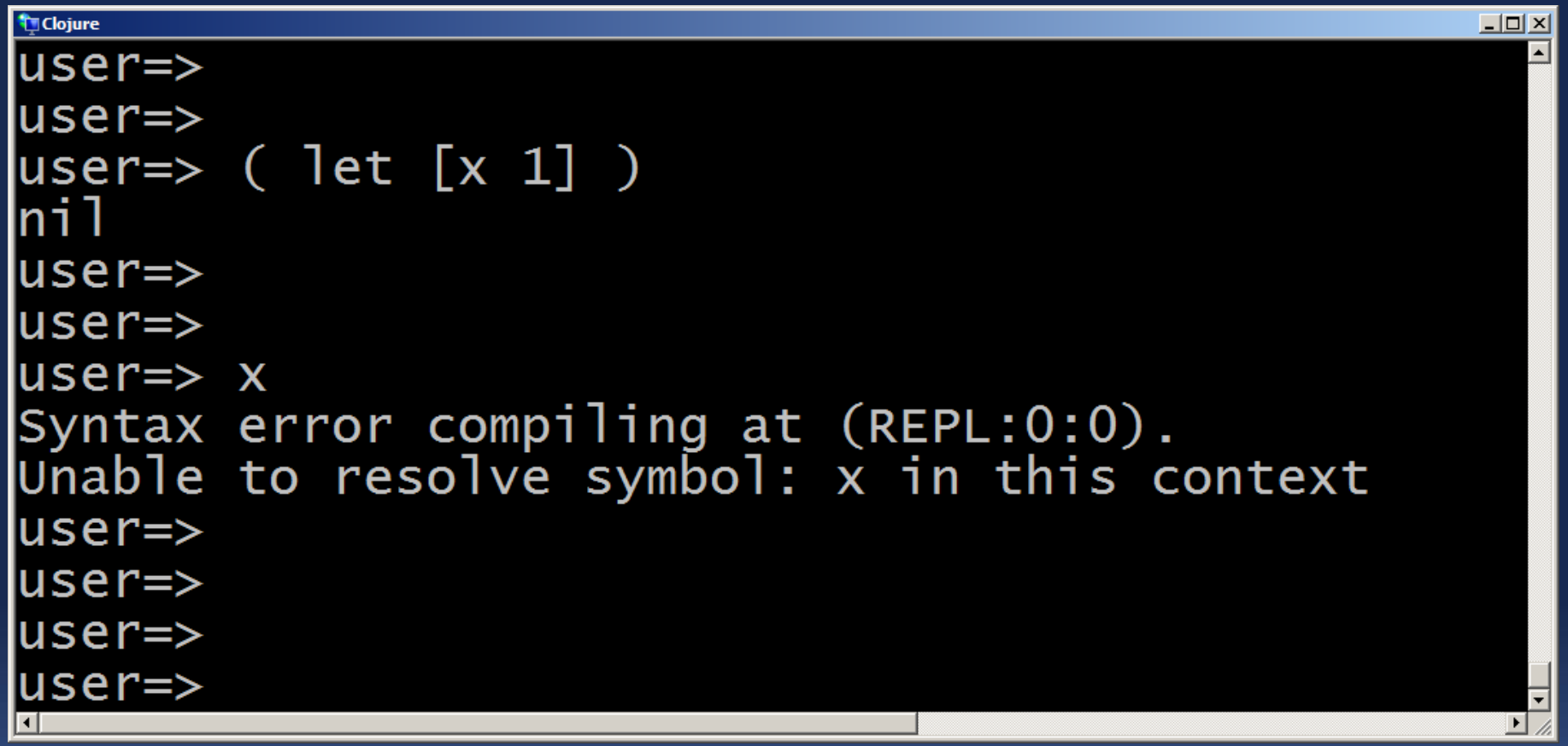
- ✓ Usado para se ligar (binding) símbolos a determinados valores;



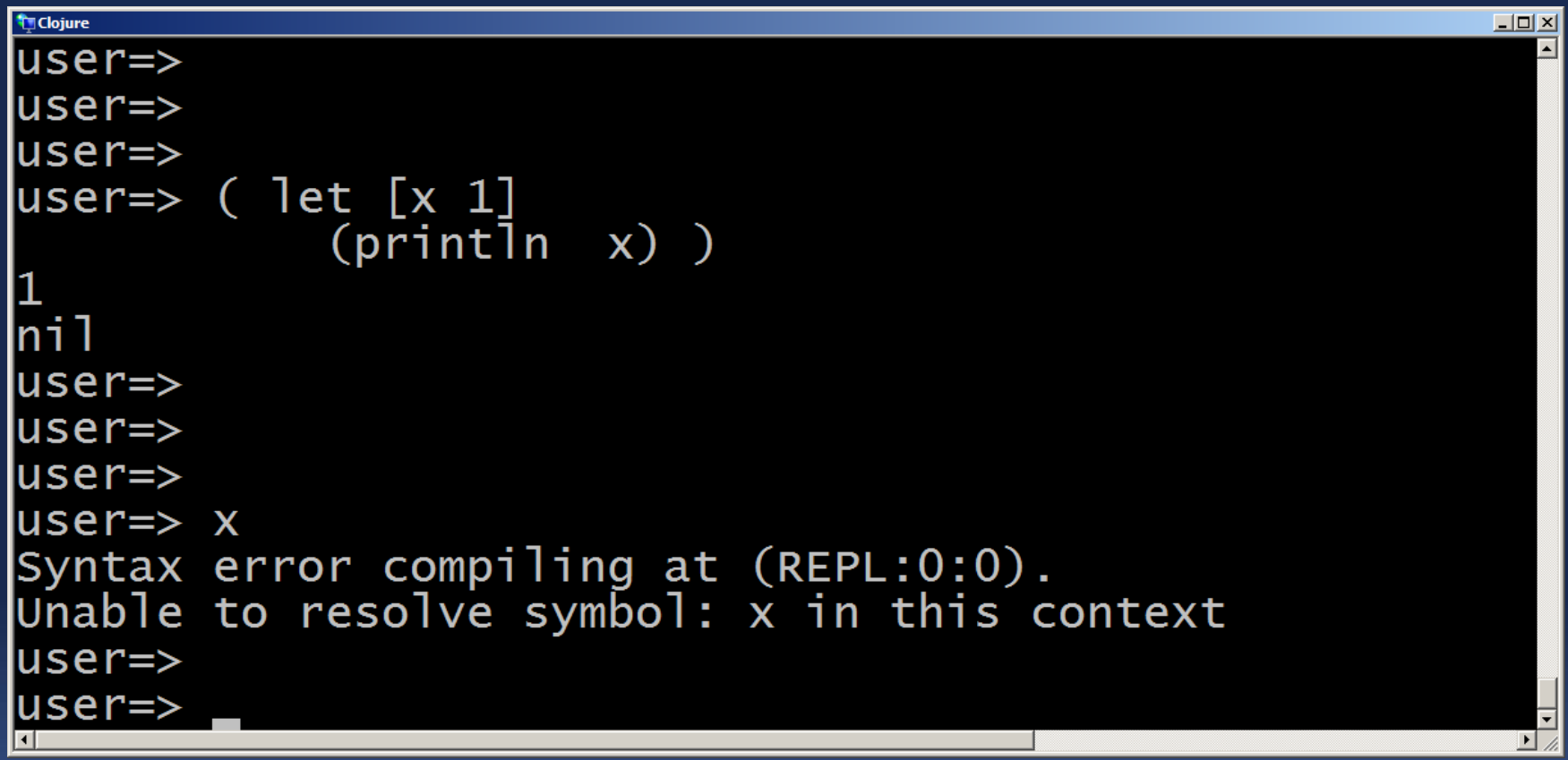
Macro – let

- ✓ Usado para se ligar (**binding**) símbolos a determinados valores;
- ✓ Opera com um **vetor** o qual associa um **símbolo** especificado no primeiro elemento a um **valor** especificado no segundo elemento;
- ✓ Os símbolos definidos no **let** **não** podem ser usados fora do let. Este comportamento é semelhante à variáveis **private** em outras linguagens;
- ✓ Avalia a expressão em um contexto léxico.

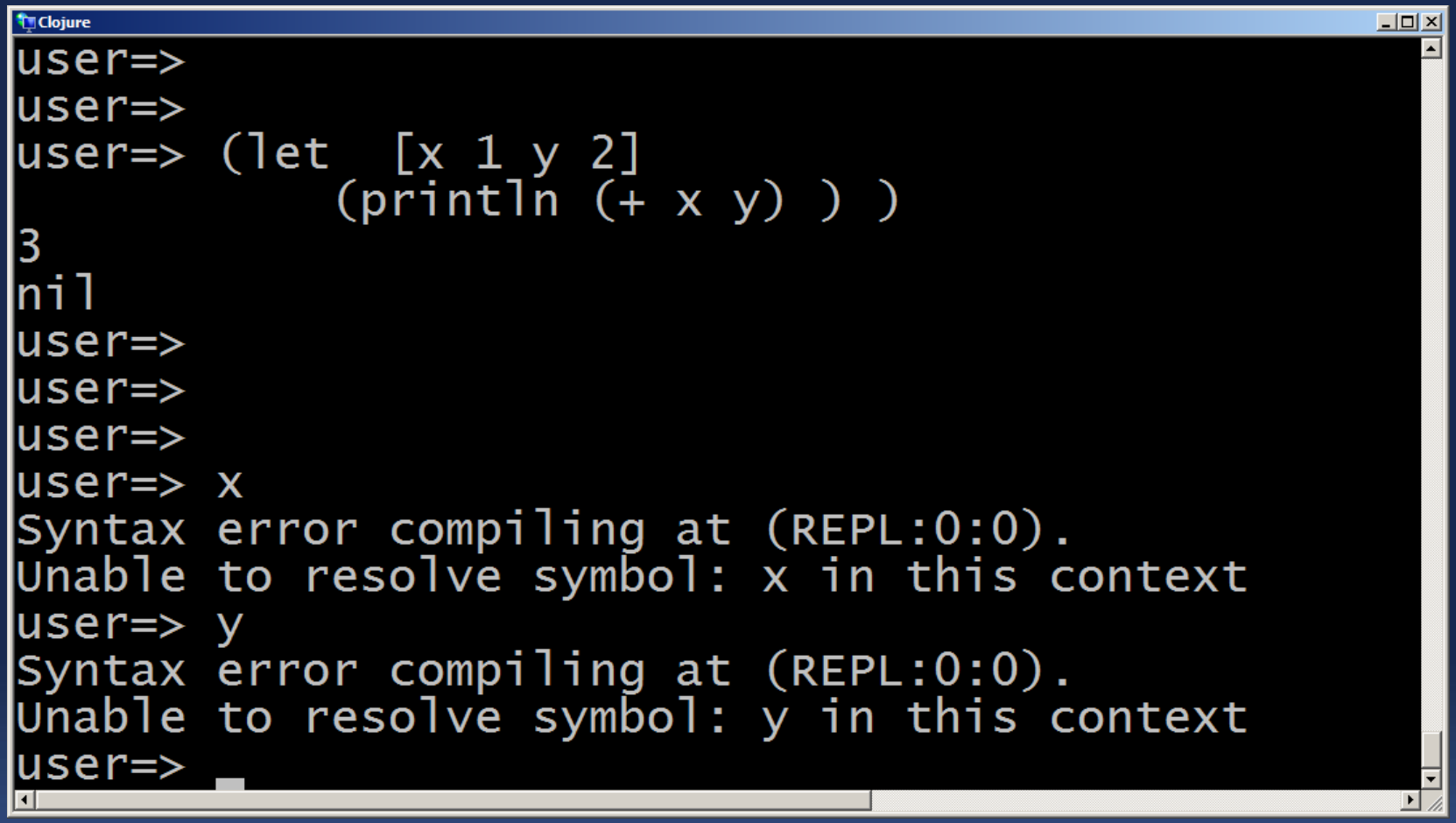
Macro - let


Clojure
user=>
user=>
user=> (let [x 1])
nil
user=>
user=>
user=> x
Syntax error compiling at (REPL:0:0).
Unable to resolve symbol: x in this context
user=>
user=>
user=>
user=>

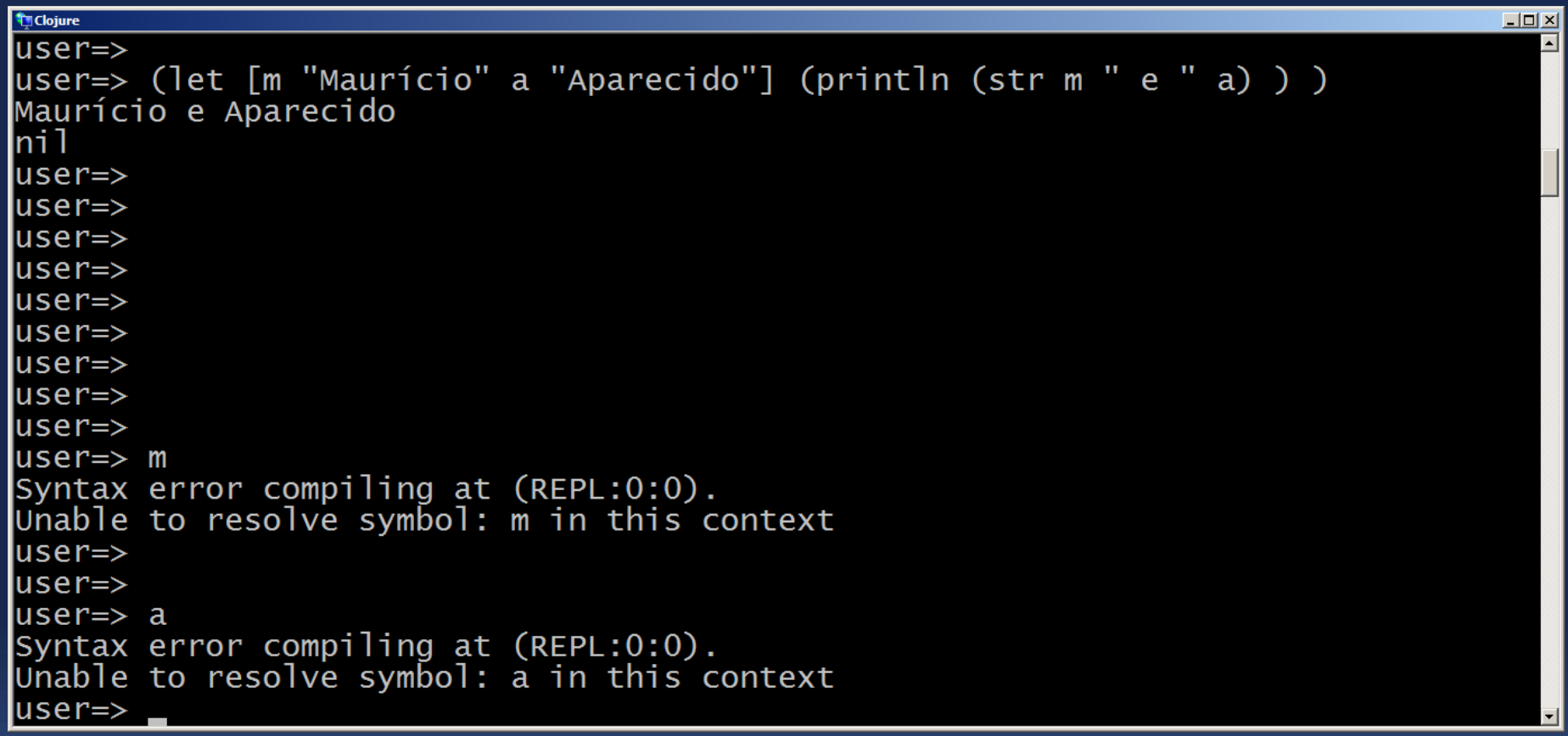
Macro - let


Clojure
user=>
user=>
user=>
user=> (let [x 1]
(println x))
1
nil
user=>
user=>
user=>
user=> x
Syntax error compiling at (REPL:0:0).
Unable to resolve symbol: x in this context
user=>
user=>

Macro - let


Clojure
user=>
user=>
user=> (let [x 1 y 2]
(println (+ x y)))
3
nil
user=>
user=>
user=>
user=> x
Syntax error compiling at (REPL:0:0).
Unable to resolve symbol: x in this context
user=> y
Syntax error compiling at (REPL:0:0).
Unable to resolve symbol: y in this context
user=>

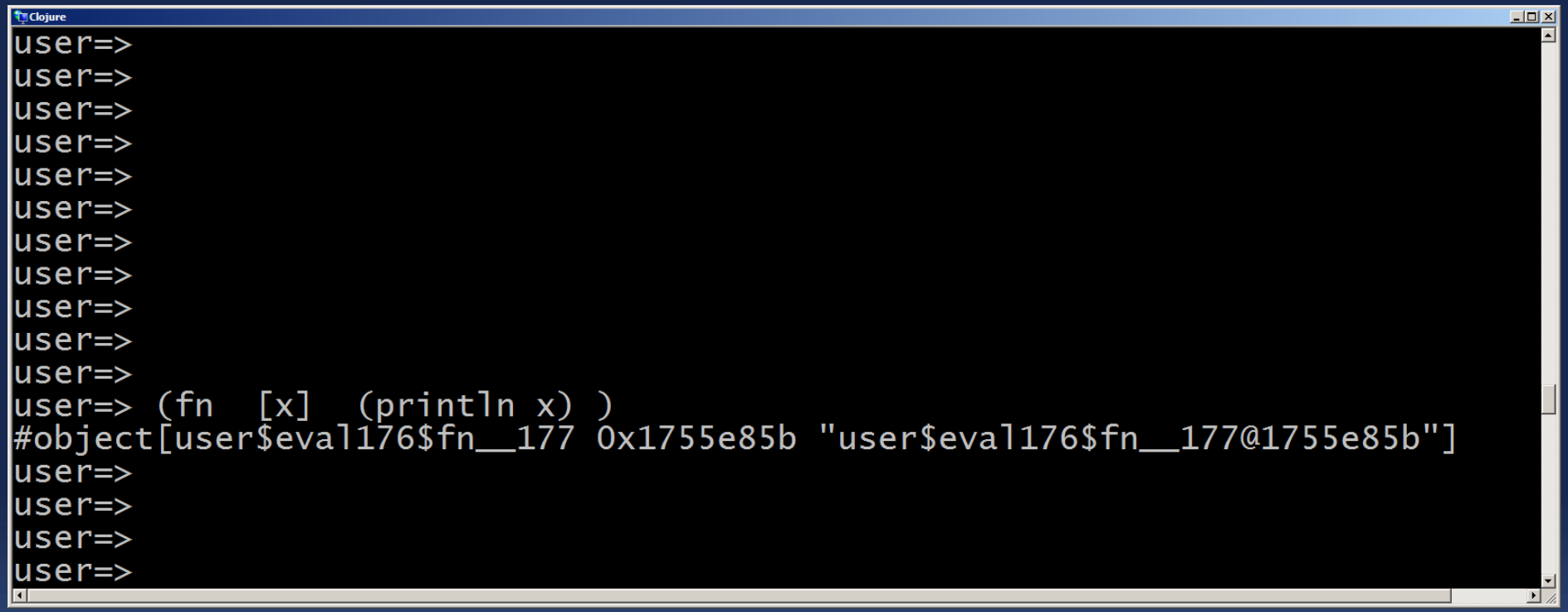
Macro - let


Clojure
user=>
user=> (let [m "Maurício" a "Aparecido"] (println (str m " e " a)))
Maurício e Aparecido
nil
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> m
Syntax error compiling at (REPL:0:0).
Unable to resolve symbol: m in this context
user=>
user=>
user=> a
Syntax error compiling at (REPL:0:0).
Unable to resolve symbol: a in this context
user=>

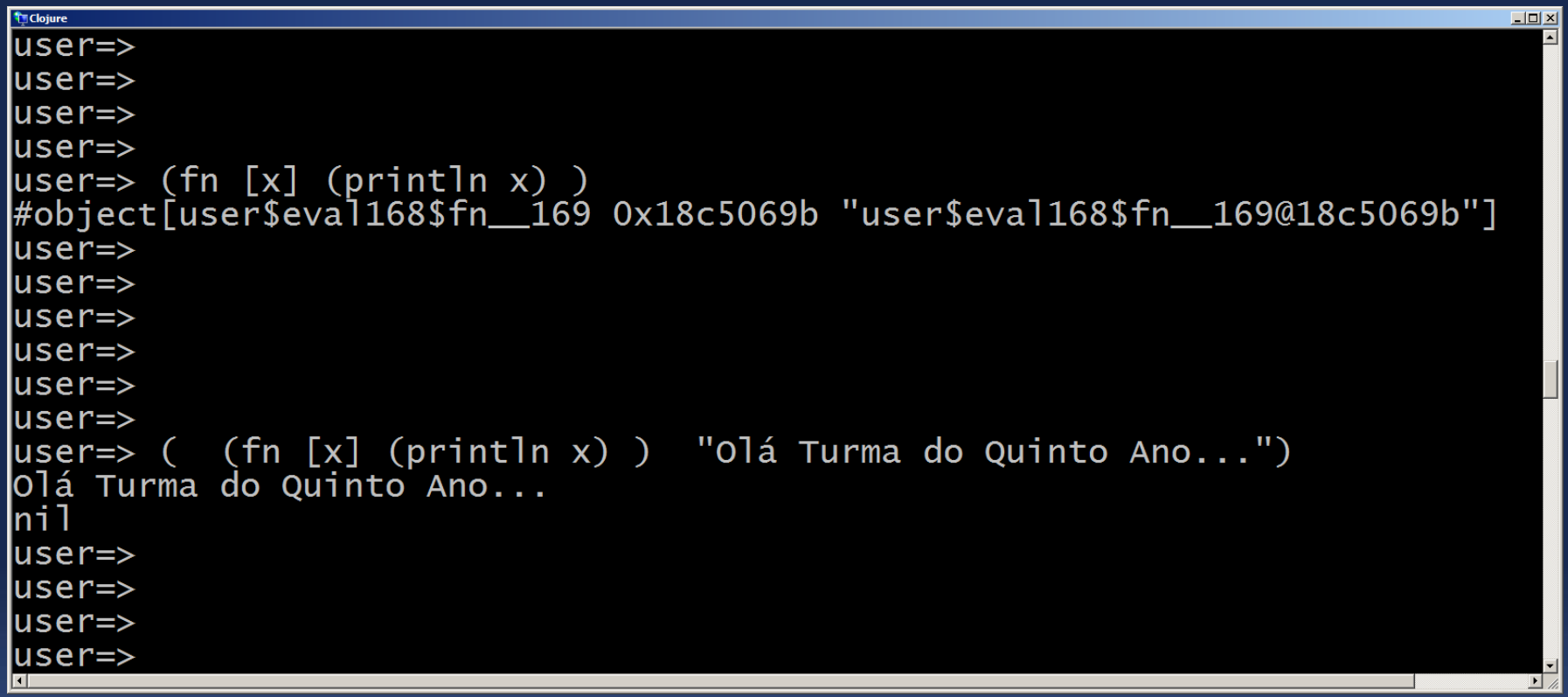
Macro - fn

- ✓ Funções são **objetos de primeira ordem em Clojure**; Isso significa que em **Clojure** pode-se fazer todas as operações básicas com funções;
- ✓ Ou seja, pode-se passar funções à outras funções, retornar funções de outras funções e ligar (binding) funções à símbolos (variáveis);
- ✓ A macro **fn** é usada para se criar **funções anônimas**;
- ✓ **#()** é o shortcut para **fn**;
- ✓ **%** será substituído pelos argumentos passados para a função. Quando houver múltiplos argumentos, **%1** para o primeiro argumento, **%2** para o segundo argumento e assim por diante.

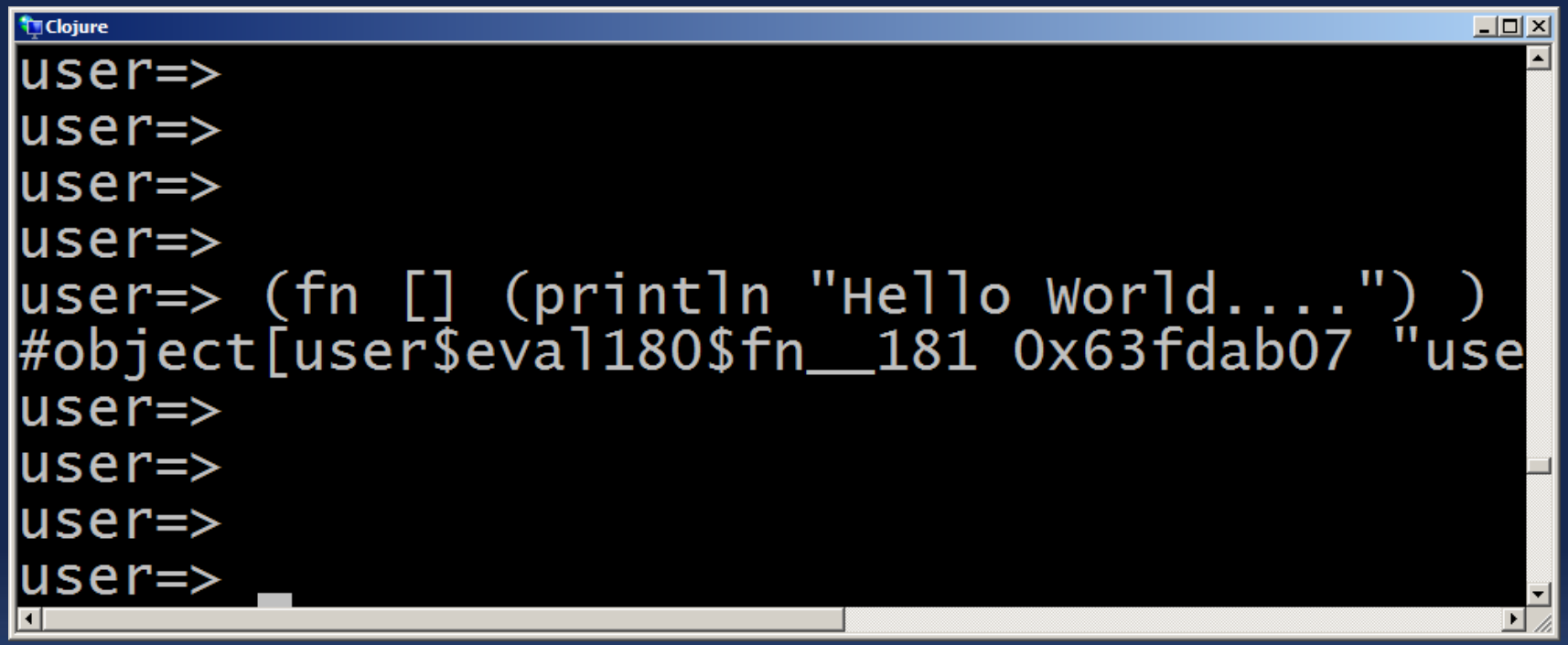
Macro - fn


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

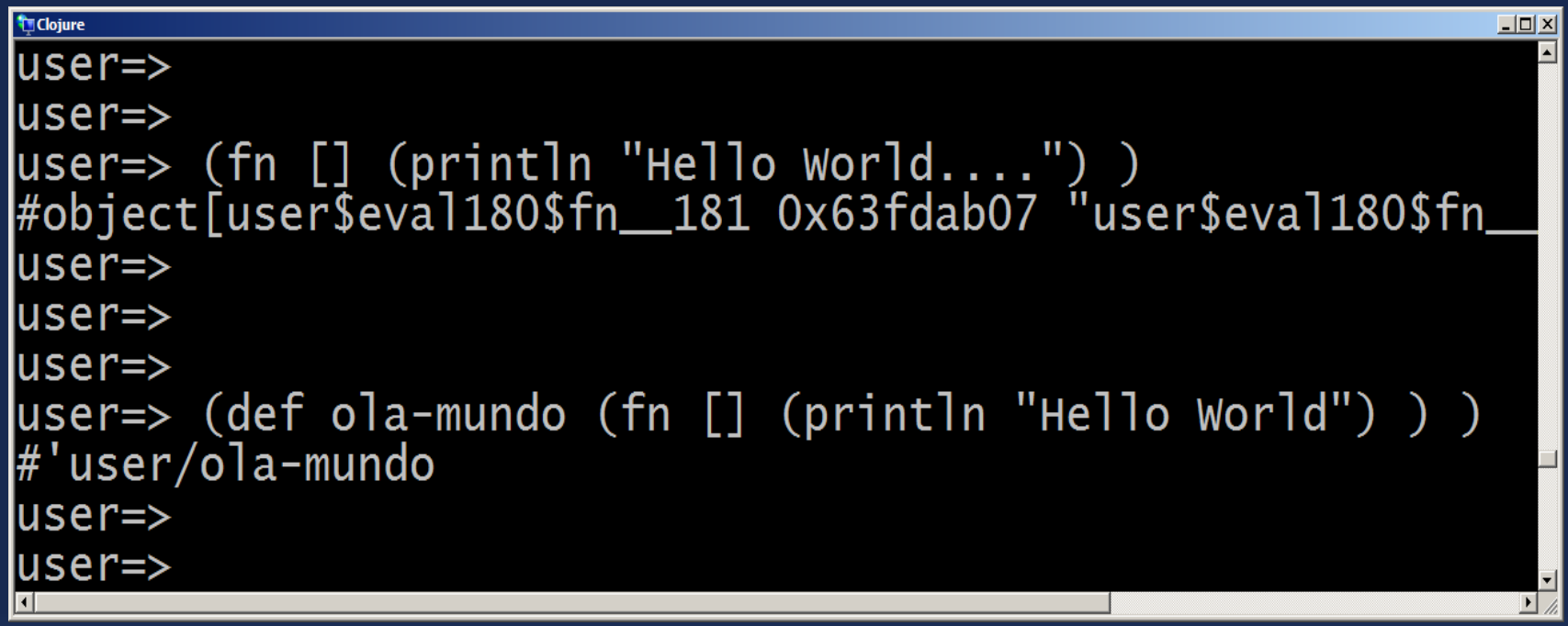
Macro - fn


Clojure
user=>
user=>
user=>
user=>
user=> (fn [x] (println x))
#object[user$eval168$fn__169 0x18c5069b "user$eval168$fn__169@18c5069b"]
user=>
user=>
user=>
user=>
user=>
user=>
user=> ((fn [x] (println x)) "Olá Turma do Quinto Ano...")
Olá Turma do Quinto Ano...
nil
user=>
user=>
user=>
user=>

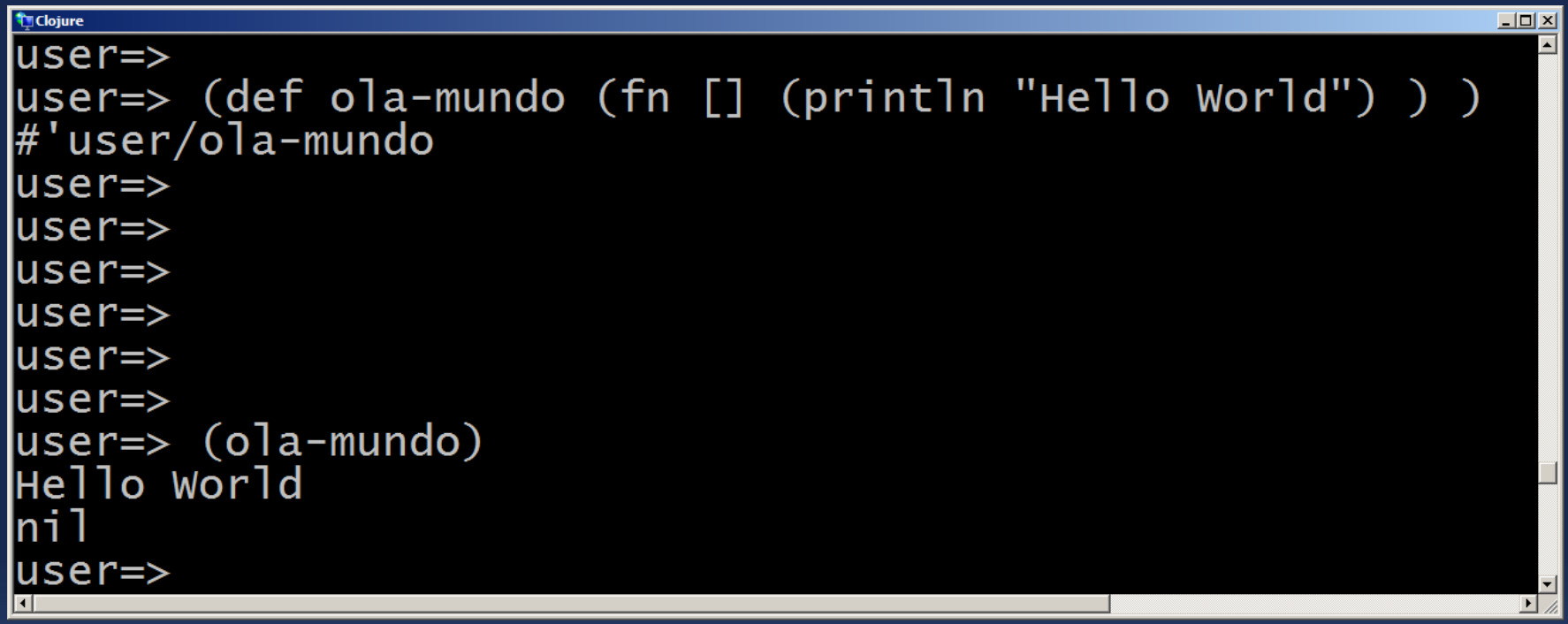
Macro - fn


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

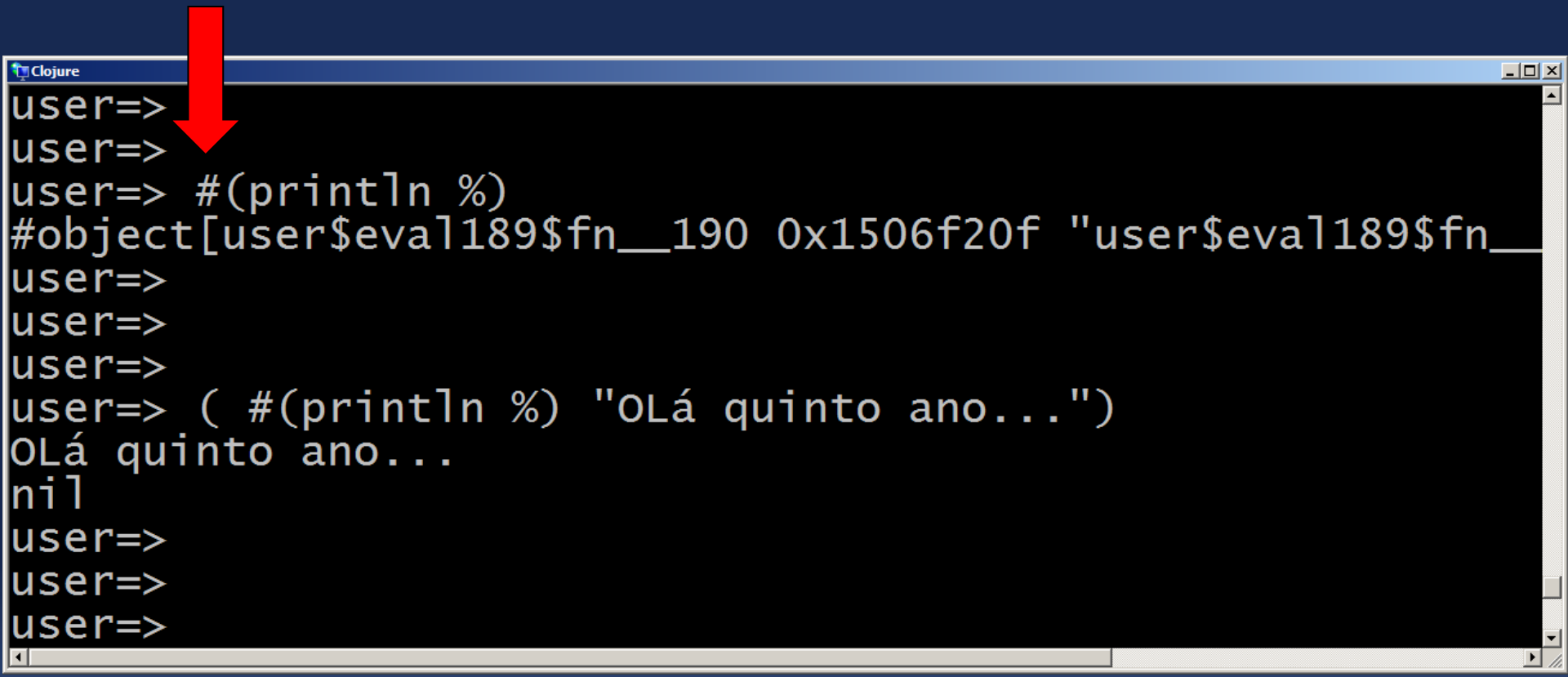
Macro - fn


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

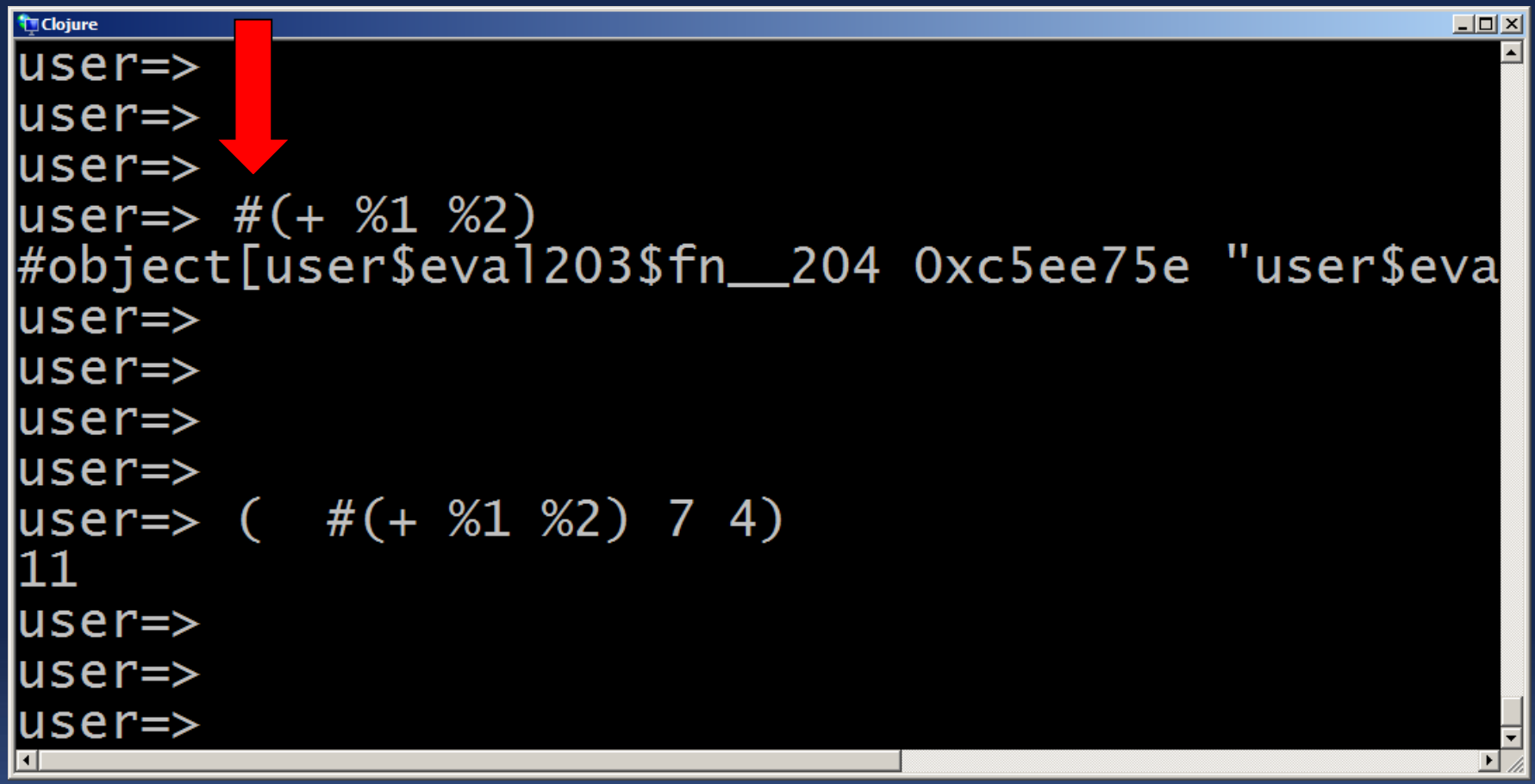
Macro - fn


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

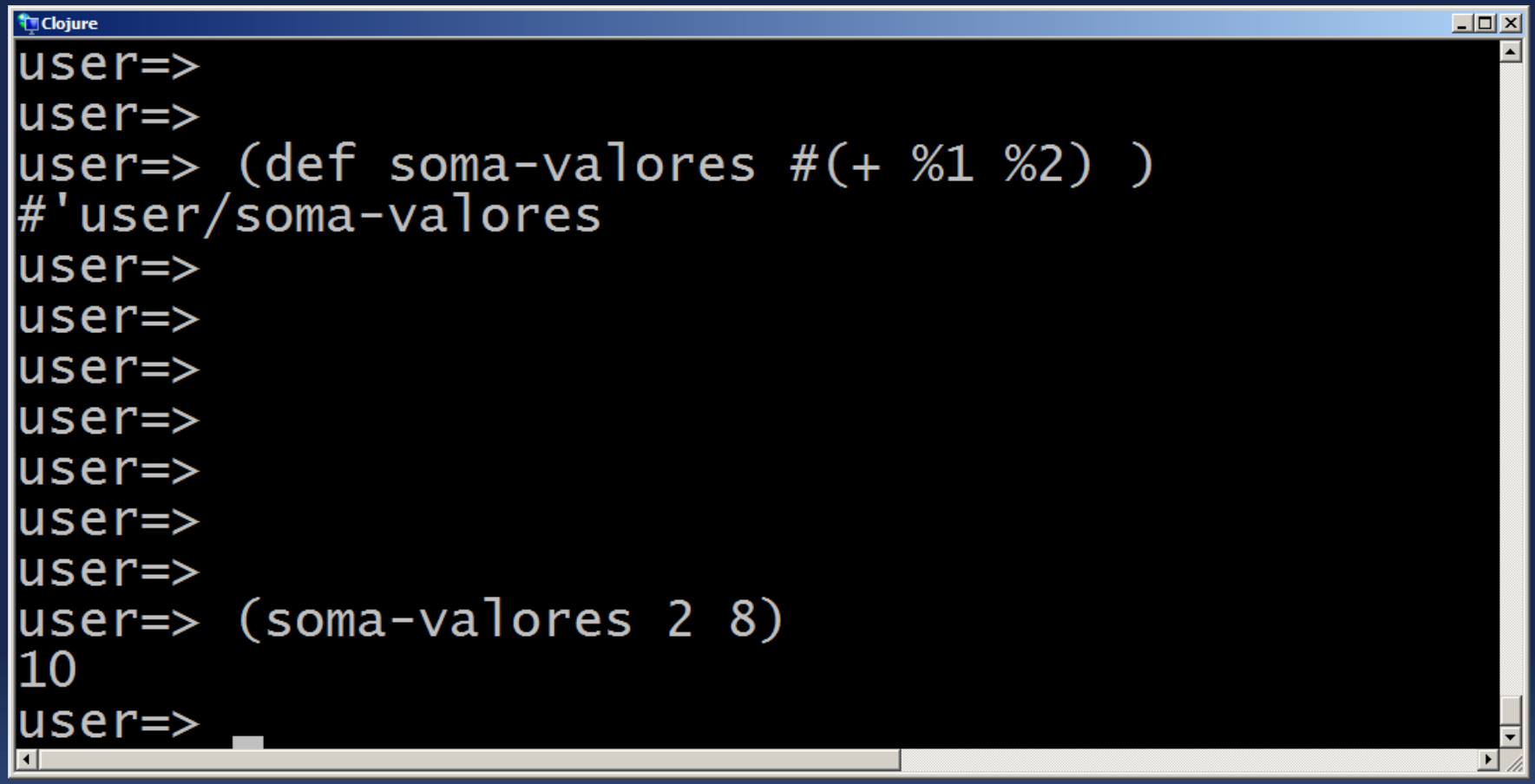
Macro - fn


Clojure
user=>
user=>
user=> #(println %)
#object[user$eval189$fn__190 0x1506f20f "user$eval189$fn__
user=>
user=>
user=>
user=> (#(println %) "OLá quinto ano...")
OLá quinto ano...
nil
user=>
user=>
user=>

Macro - fn


Clojure
user=>
user=>
user=>
user=> #(+ %1 %2)
#object[user$eval203$fn__204 0xc5ee75e "user$eva
user=>
user=>
user=>
user=>
user=> (#(+ %1 %2) 7 4)
11
user=>
user=>
user=>

Macro - fn


Clojure
user=>
user=>
user=> (def soma-valores #(+ %1 %2))
#'user/soma-valores
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> (soma-valores 2 8)
10
user=>

Macro - fn
Passando funções à funções

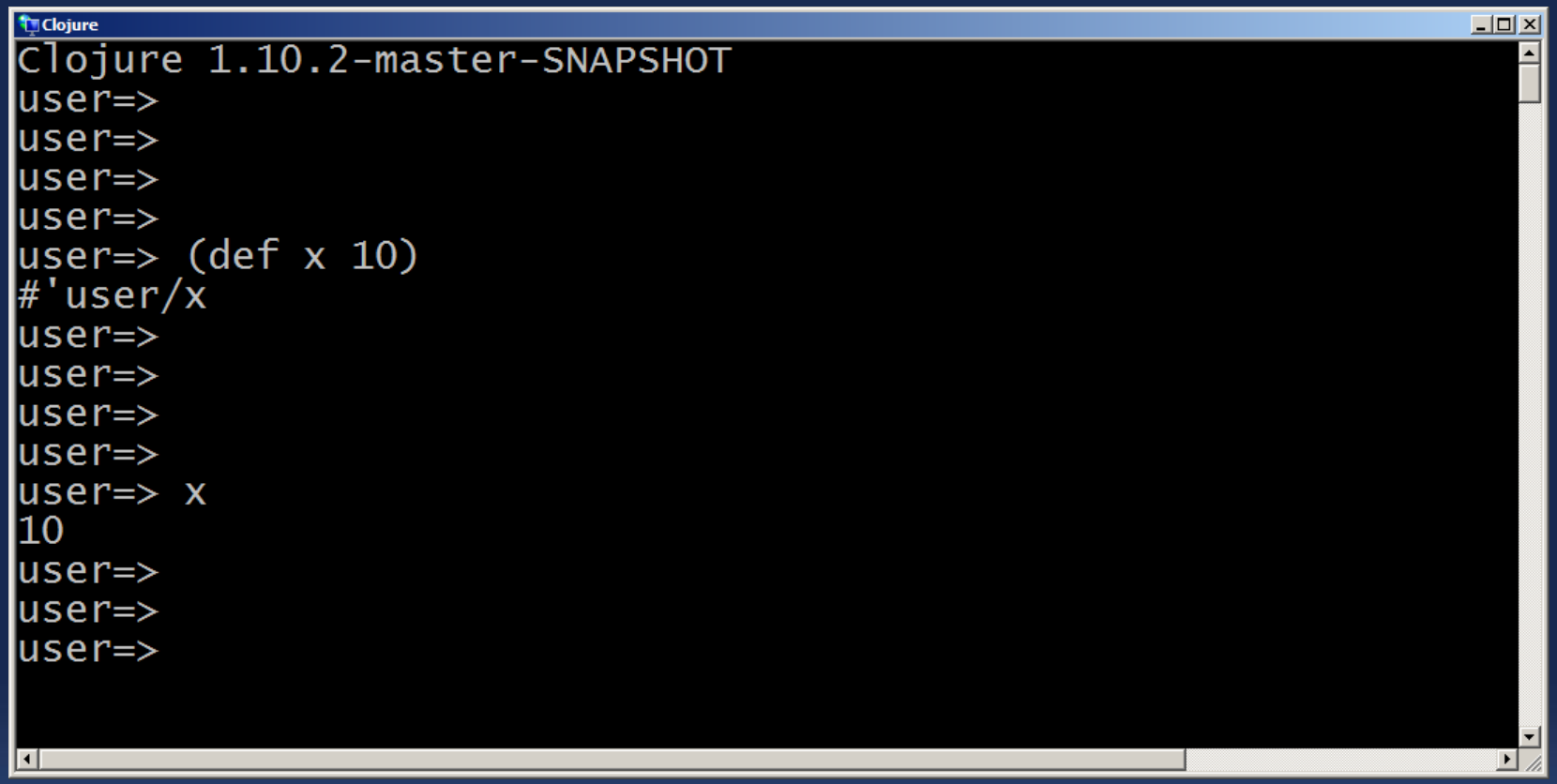
Clojure

```
user=>
user=>
user=> (def funcao-hello (fn [nome] (println (str "Hello, " nome) ) ) )
#'user/funcao-hello
user=>
user=>
user=>
user=> (def funcao-tchau (fn [nome] (println (str "Tchau, " nome) ) ) )
#'user/funcao-tchau
user=>
user=>
user=> (def saudacoes (fn [funcao nome] (funcao nome) ) )
#'user/saudacoes
user=>
user=>
user=> (saudacoes funcao-hello "Maurício")
Hello, Maurício
nil
user=>
user=> (saudacoes funcao-tchau "Maurício")
Tchau, Maurício
nil
user=>
user=>
```

Atividade 7

- ✓ Abra o **REPL**;
- ✓ Com o emprego da special form **def**, escreva o código que efetua a ligação do símbolo x ao valor 10;
- ✓ Em seguida, avalie o valor do símbolo x;
- ✓ Encerre o **REPL**.

Atividade 7



- ✓ Observação: Quando **REPL** retorna **#'user/x**, ele está retornando uma referência ao símbolo **x** (variável) criado. A parte **user** indica o **namespace** onde o símbolo **x** foi definido

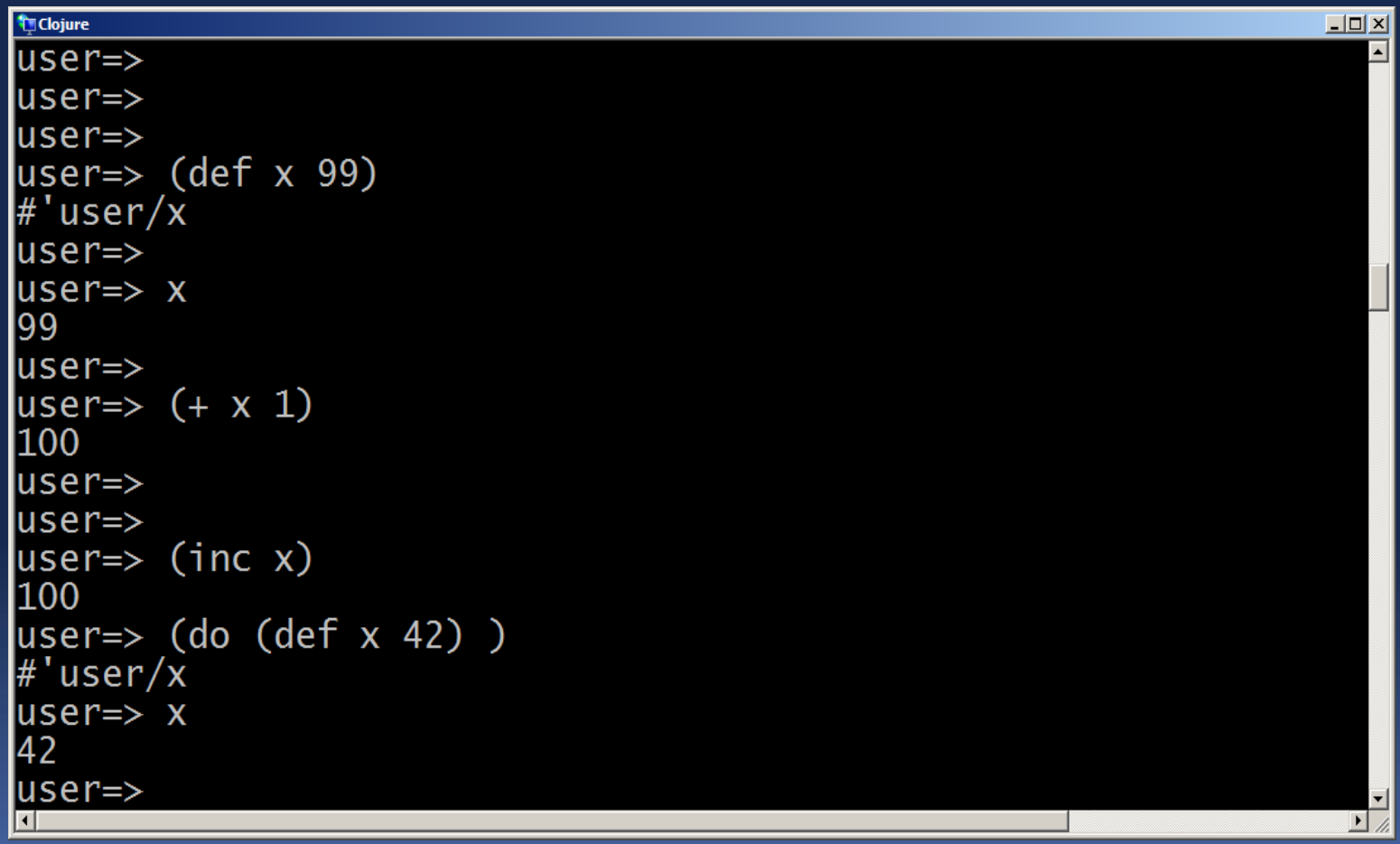
Macro - fn

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**;

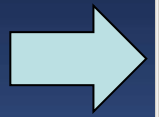


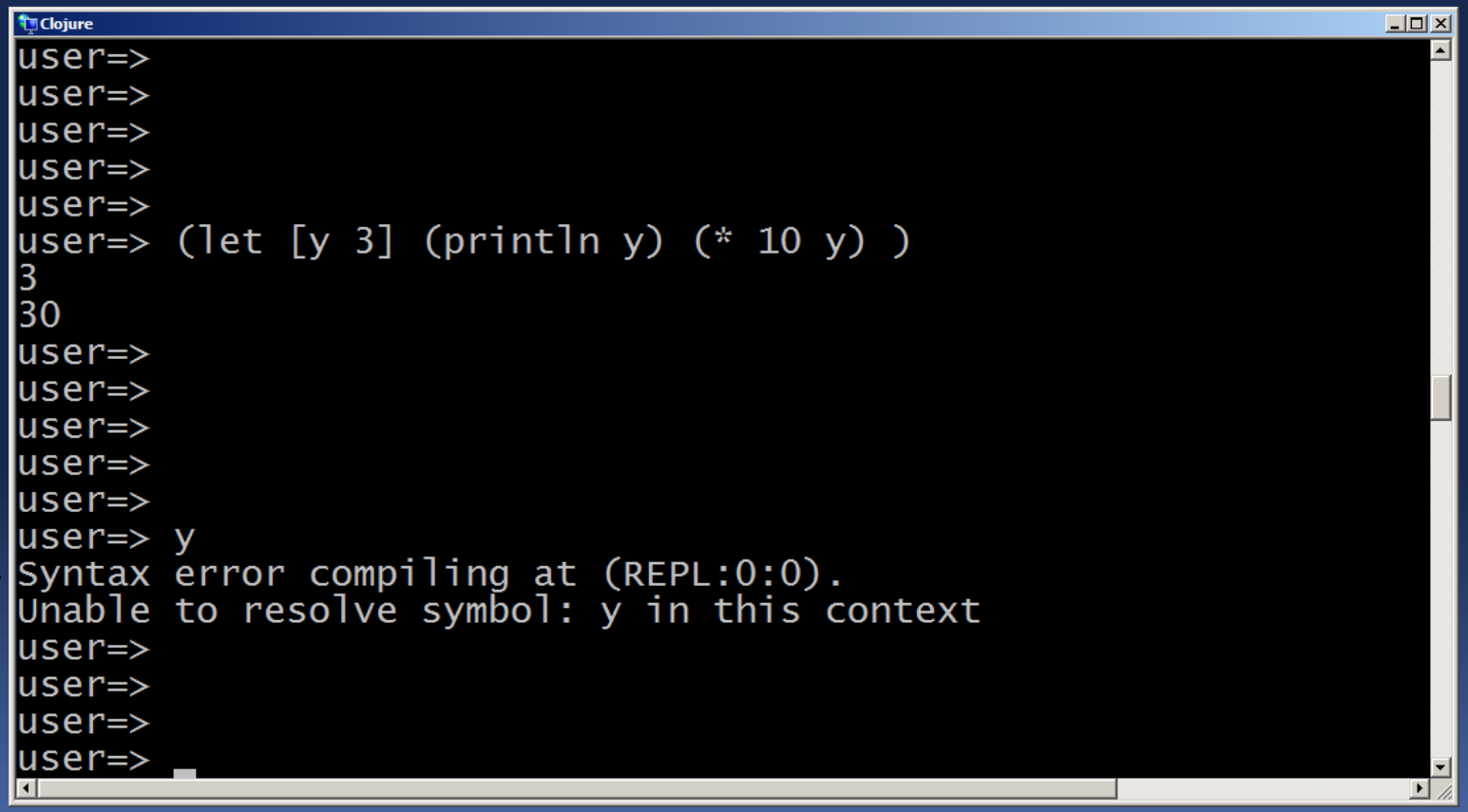
Binding e Escopo

- ✓ **Bindings** criados pela special form **def** têm **escopo dinâmico** e podem ser considerados como "**globais**". Eles são, dessa forma, automaticamente namespaced, o qual é útil para evitar conflitos com nomes existentes;

- ✓ Para se definir bindings com escopo **local** (ou escopo **léxico**), deve-se usar a special form **let**.

y está fora do escopo definido em let !!!

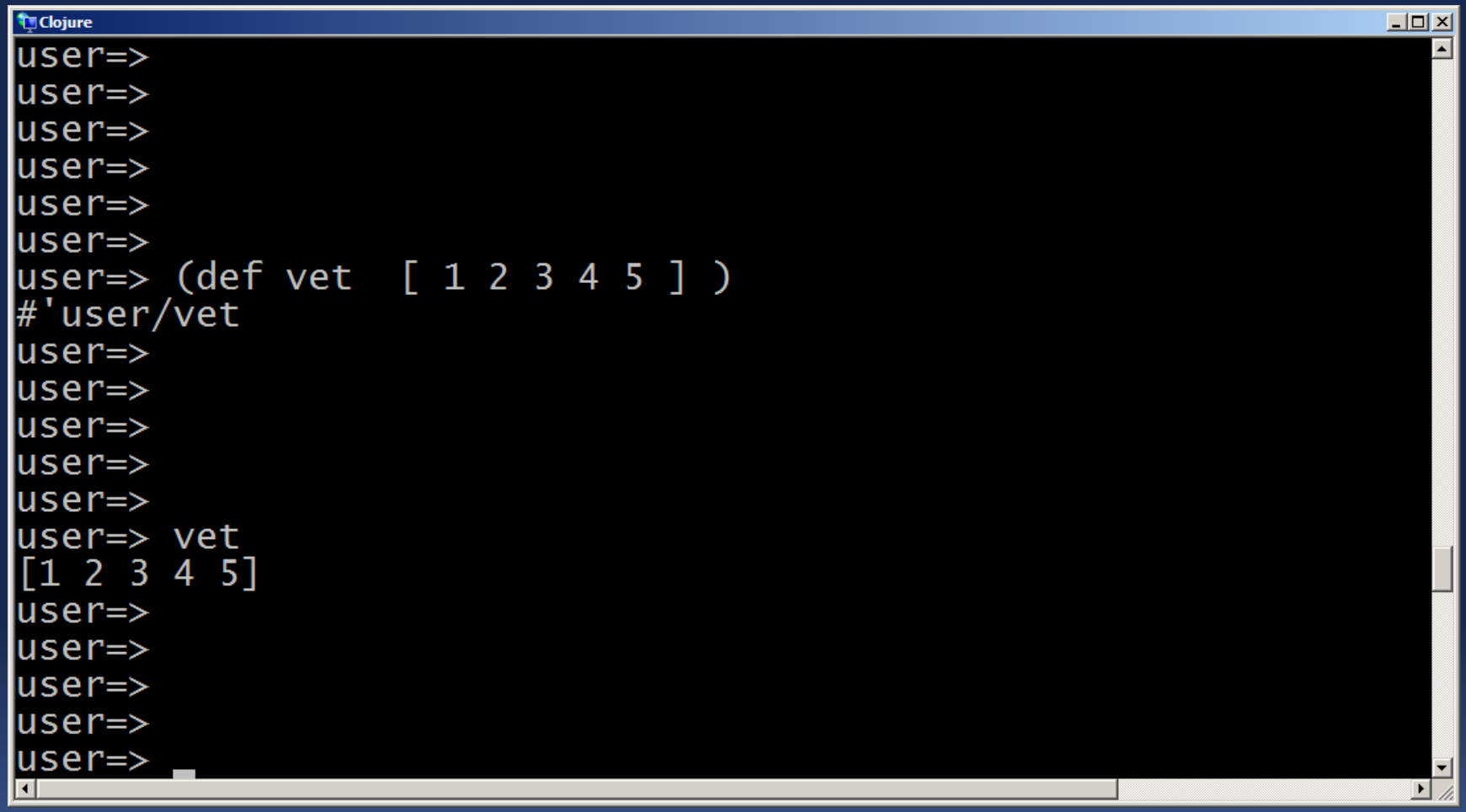




Observação let

- ✓ A special form **let** emprega um vetor como um parâmetro para criar bindings locais e uma série de expressões que serão avaliadas como se estivessem em um bloco;
- ✓ Um **vetor** é similar a uma **lista**, no sentido em que ambos correspondem a uma coleção sequencial de valores;
- ✓ Na próxima unidade veremos com mais detalhes os vetores;
- ✓ Por ora, apenas é importante saber que **vetores** podem ser criados por **colchetes**;
- ✓ Por exemplo: `[1 2 3 4 5 ]`

Definindo vetores



Escopos Global e Local

Escopo Global

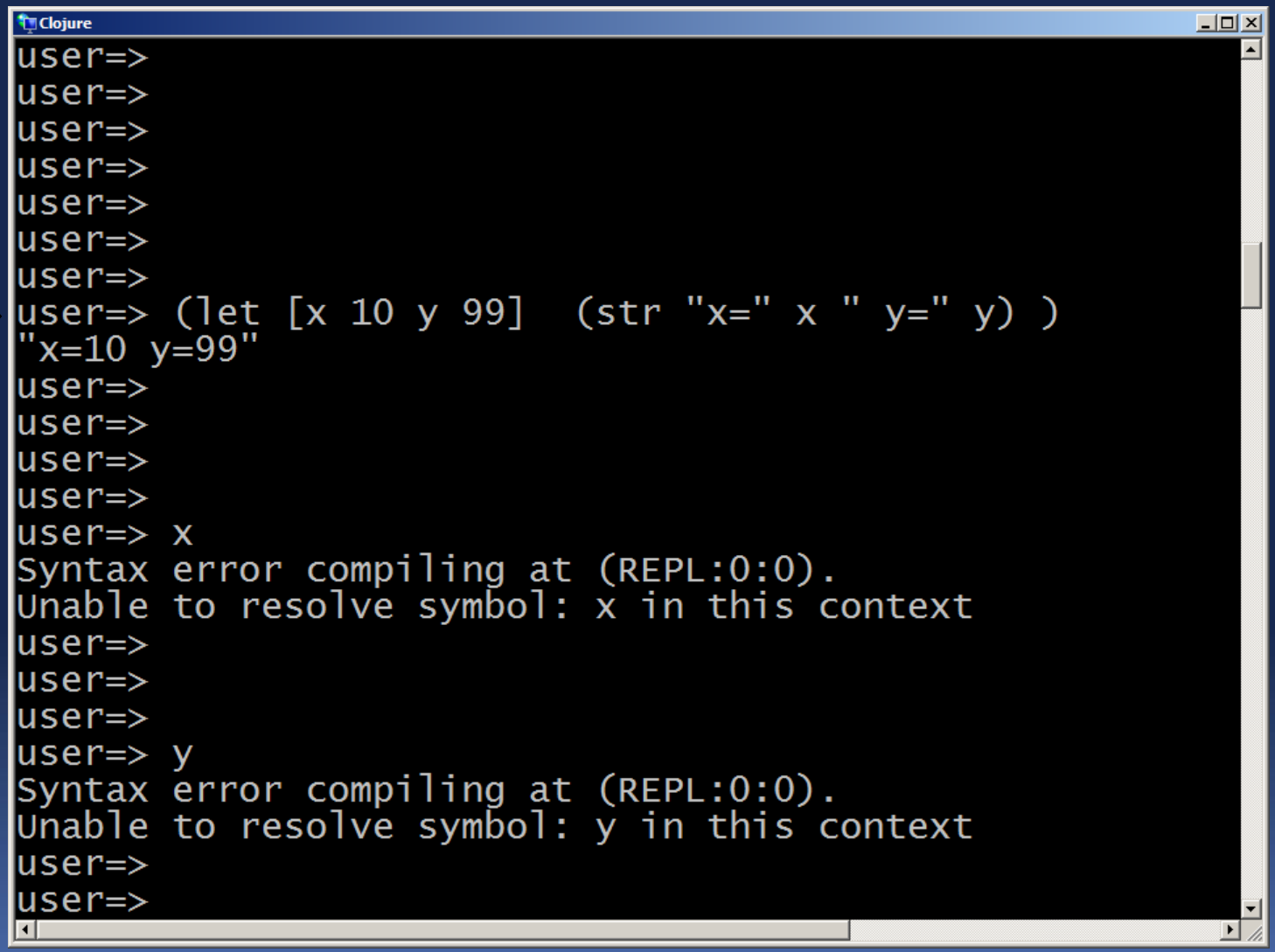
Escopo Local

Escopo Global

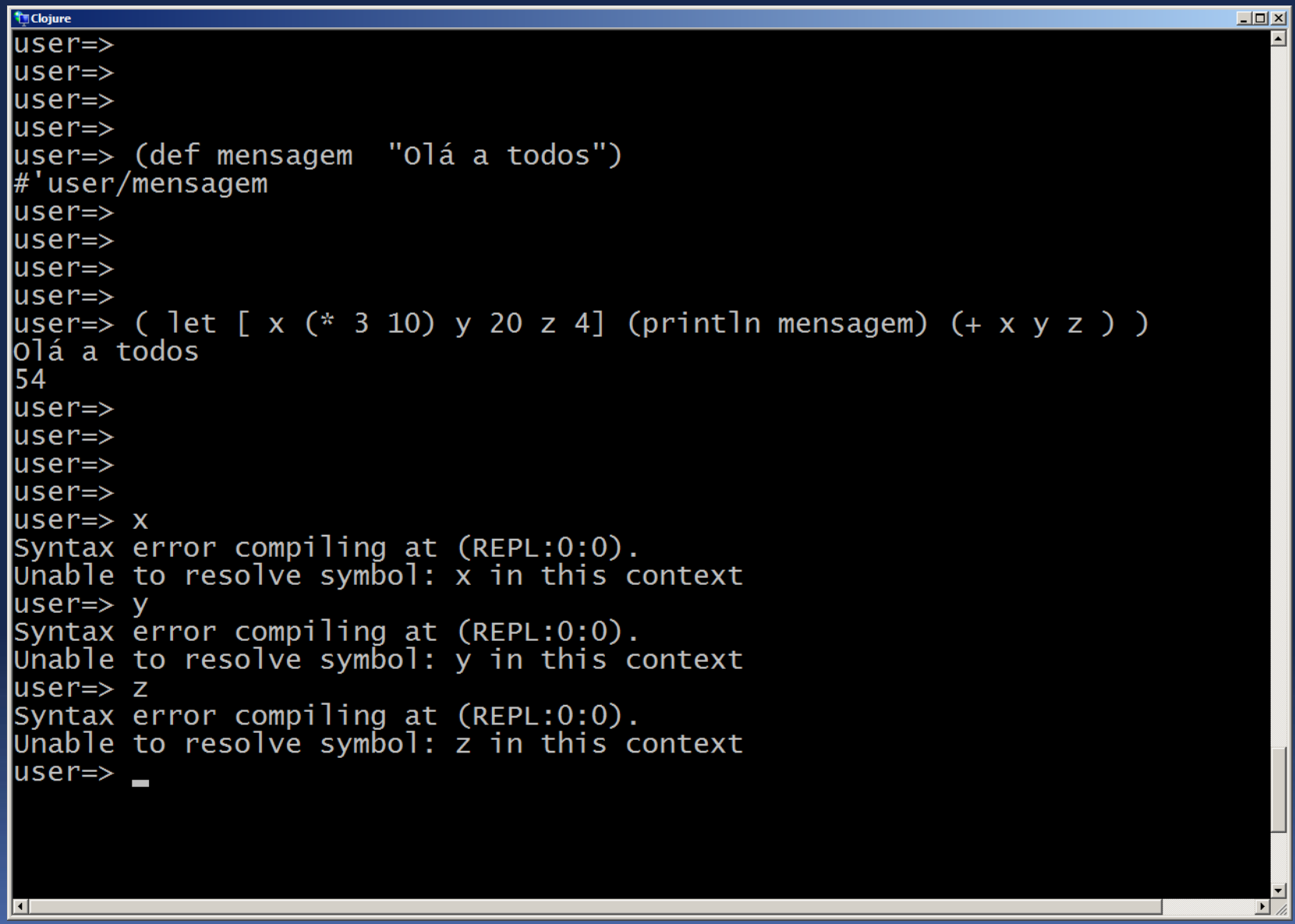
```
Clojure
user=>
user=>
user=>
user=>
user=>
user=>
user=>
user=> (def x 99)
#'user/x
user=> x
99
user=> (let [x 5] (println x) )
5
nil
user=> x
99
user=>
user=>
user=>
user=>
```

Macro let – Escopo local

Escopo Local



Macro let – Escopo local

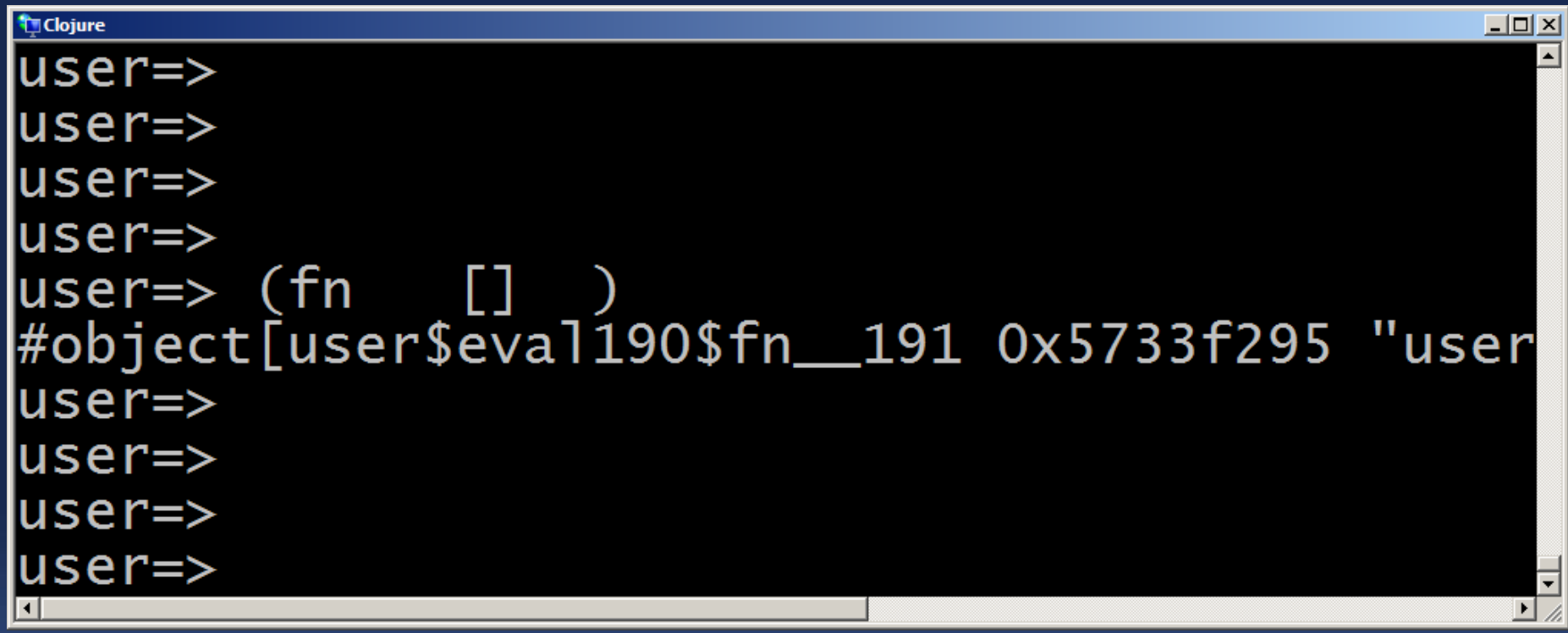


Atividade 8

- ✓ Abra o **REPL**;
- ✓ Com a **macro** **fn** defina uma função que **não** recebe argumentos e que também **nada** faz;
- ✓ Encerre o **REPL**.

Atividade 8

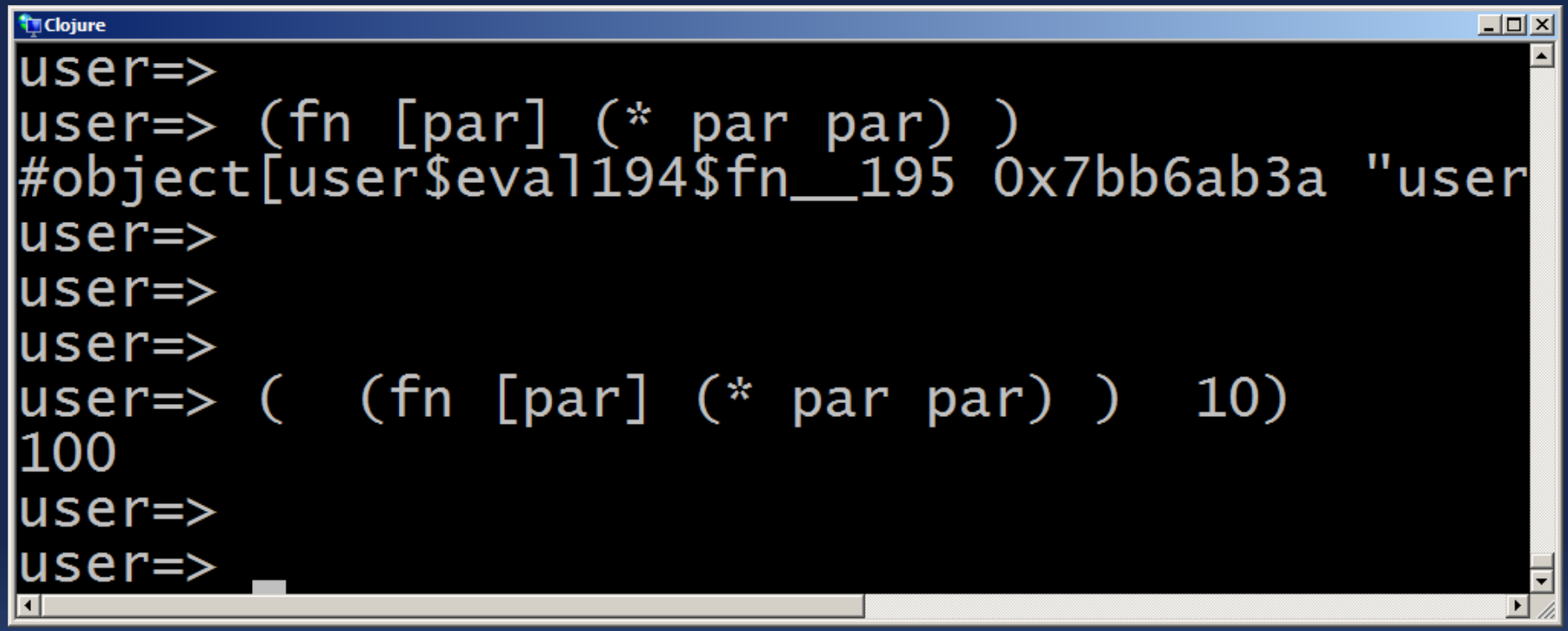
(fn [ ])



Atividade 9

- ✓ Abra o **REPL**;
- ✓ Com a **macro** **fn** defina uma função que recebe um parâmetro chamado **par** e retorna seu valor ao **quadrado** (**par** multiplicado por ele mesmo)
- ✓ Encerre o **REPL**.

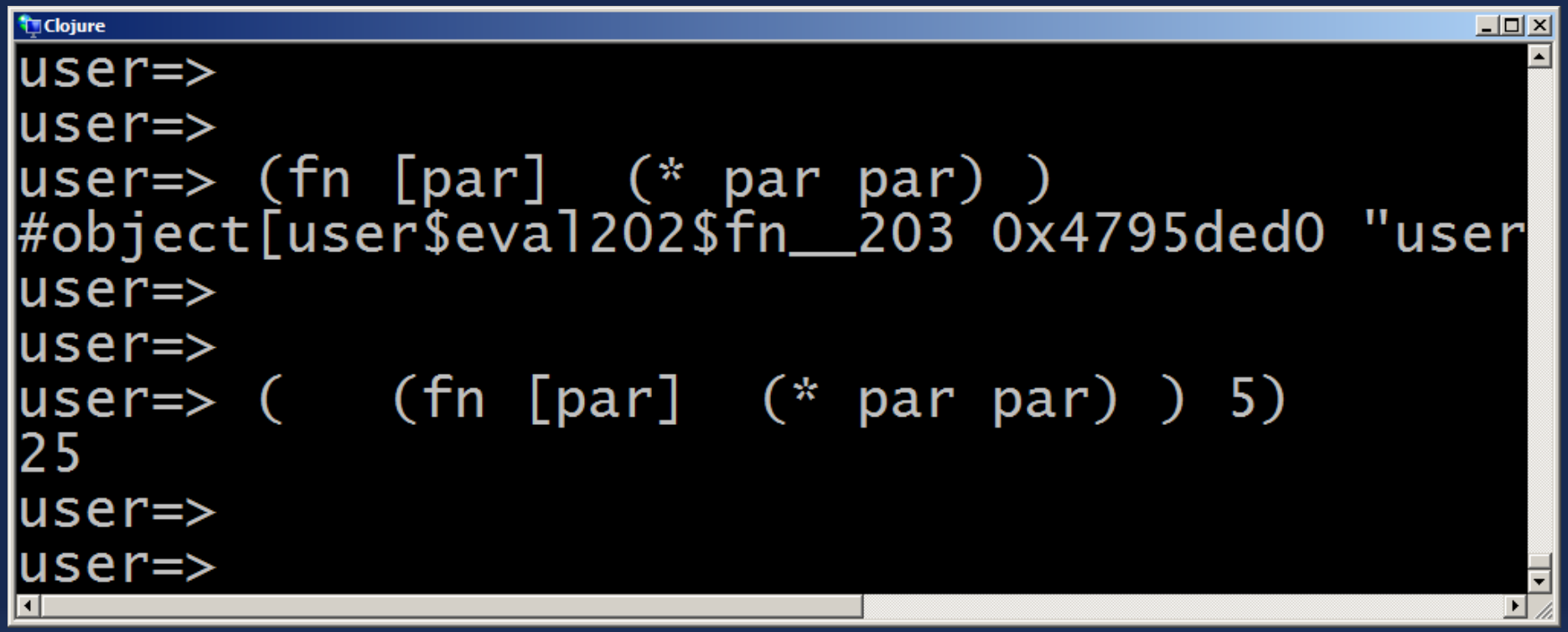
Atividade 9


Clojure
user=>
user=> (fn [par] (* par par))
#object[user$eval194$fn__195 0x7bb6ab3a "user
user=>
user=>
user=>
user=> ((fn [par] (* par par)) 10)
100
user=>
user=>

Atividade 10

- ✓ Abra o **REPL**;
- ✓ Com a macro **fn** defina uma função que recebe um parâmetro chamado **par** e retorna seu valor ao **quadrado** (**par** multiplicado por ele mesmo).
- ✓ Após a definição da função anônima (com fn) efetuar a aplicação da função passando a ela um argumento com o valor 5.
- ✓ Encerre o **REPL**.

Atividade 10



Observação: Nessa atividade, definimos uma função anônima e junto com a definição fizemos uma aplicação da função (execução) passando a ela um argumento com o valor 5. A função anônima retornou o quadrado de 5 que é 25.

Atividade 11

- ✓ Abra o **REPL**;

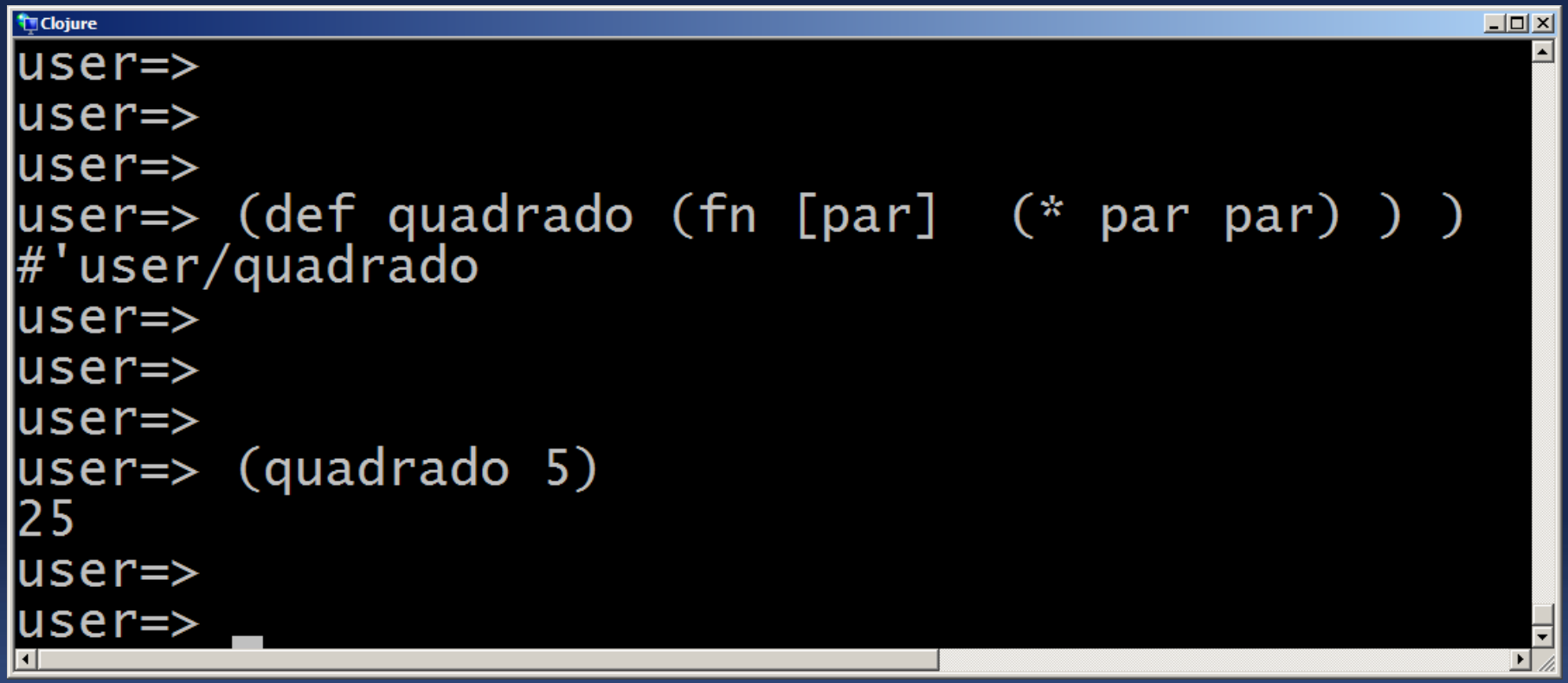
- ✓ Com a macro **fn** desenvolvida na atividade anterior, foi possível definirmos uma função de forma anônima e executá-la por meio da aplicação da função com o argumento passado, no caso 5.

- ✓ Mas, e se quizermos reaproveitar essa função ? Ou seja, reusá-la em outra ocasião ! Para esse caso, precisaríamos associar à essa função um nome que permitisse chamá-la em outra ocasião. Repita a atividade de forma que a função deixe de ser anônima e passe assim a ter um nome.

- ✓ Encerre o REPL.

Dica: Nessa atividade, para definirmos função com nome devemos usar a macro def.

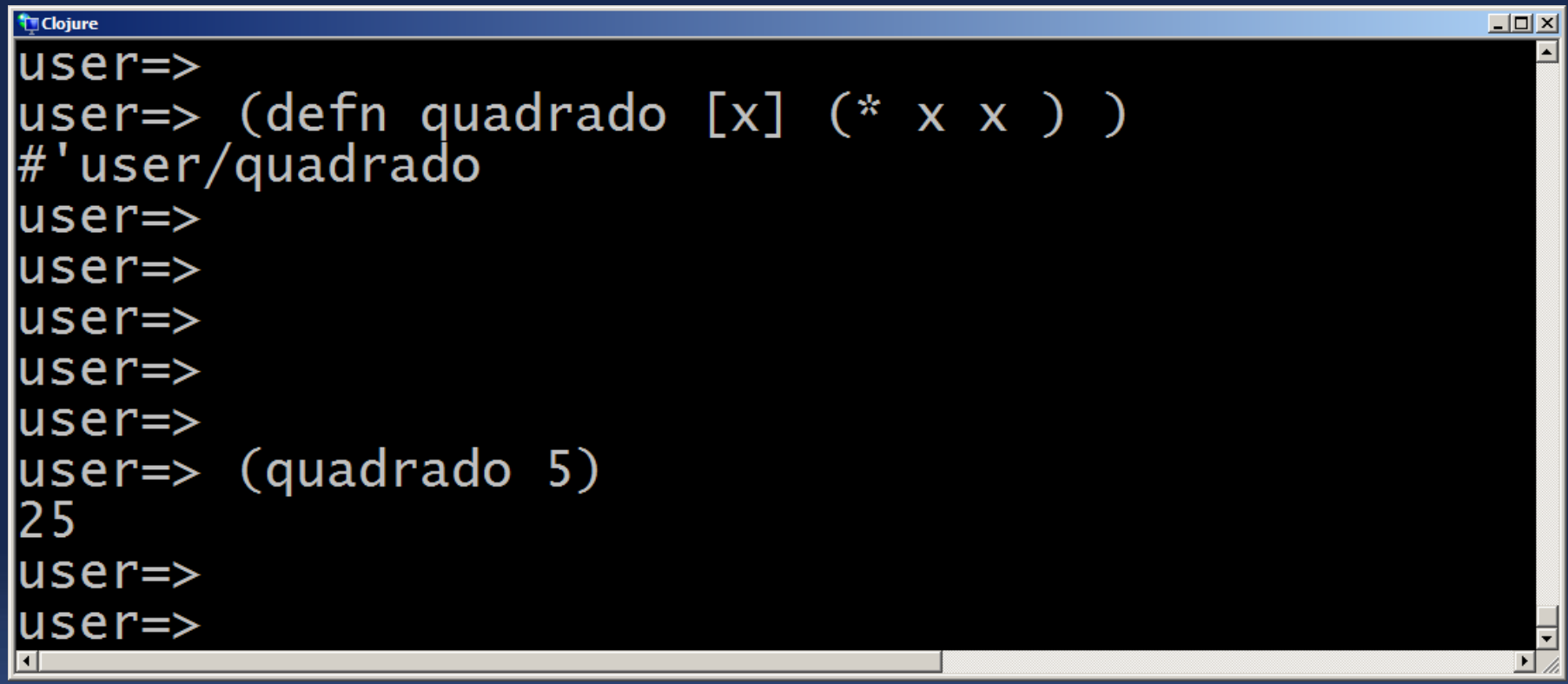
Atividade 11



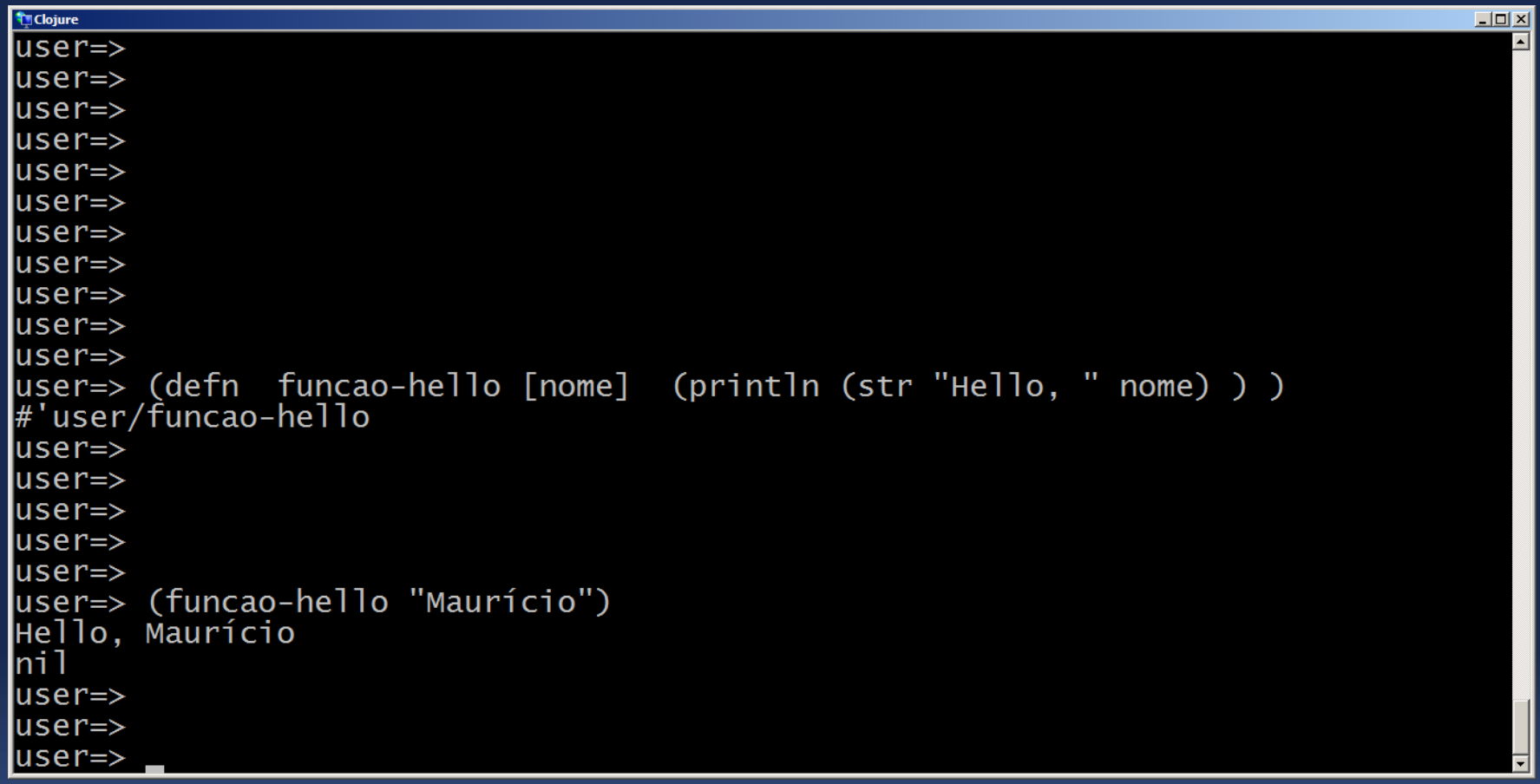
Macro defn

- ✓ 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

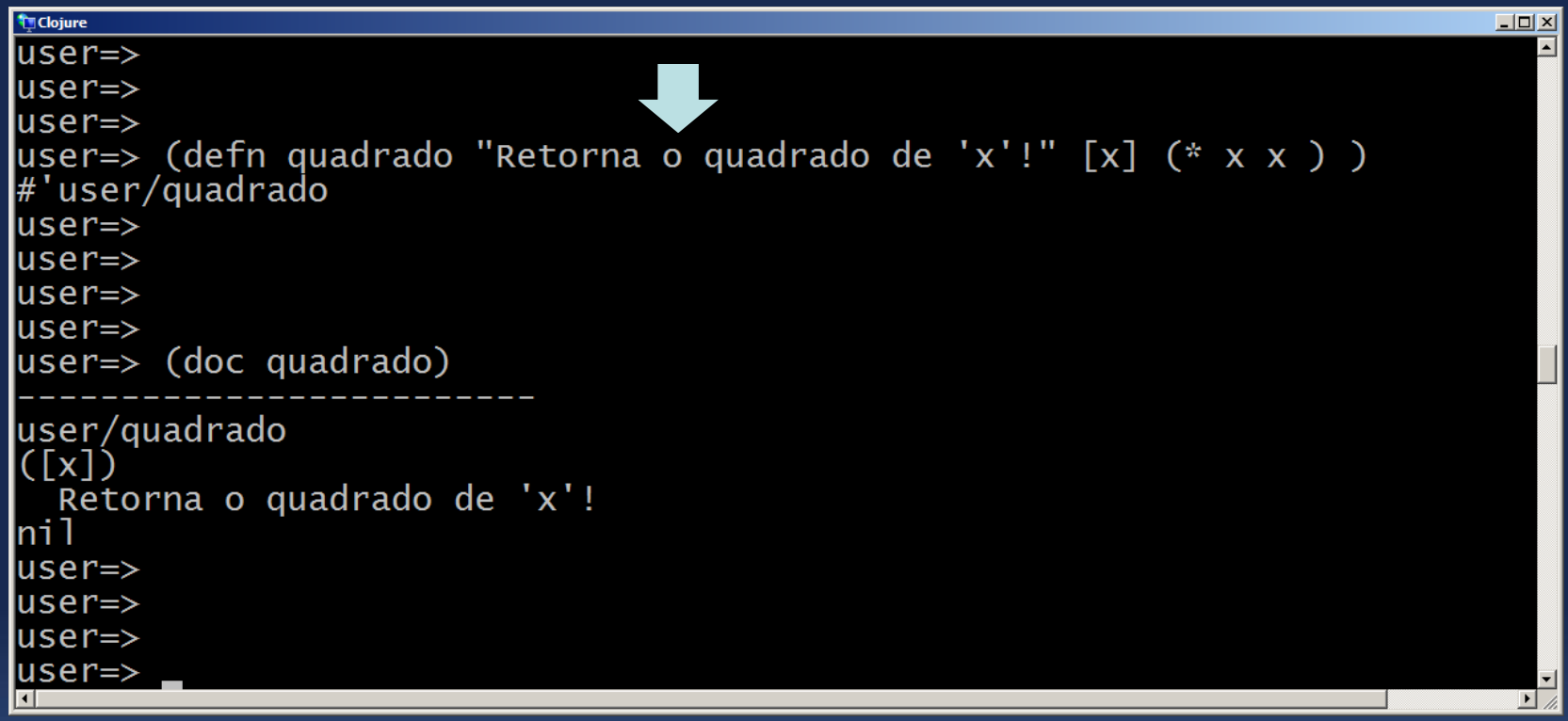


Exemplo - Macro defn



Adicionando documentação à função

- ✓ Para adicionar alguma documentação em uma função definida com a macro **defn**, basta acrescentar um **doc-string** antes dos argumentos da função !

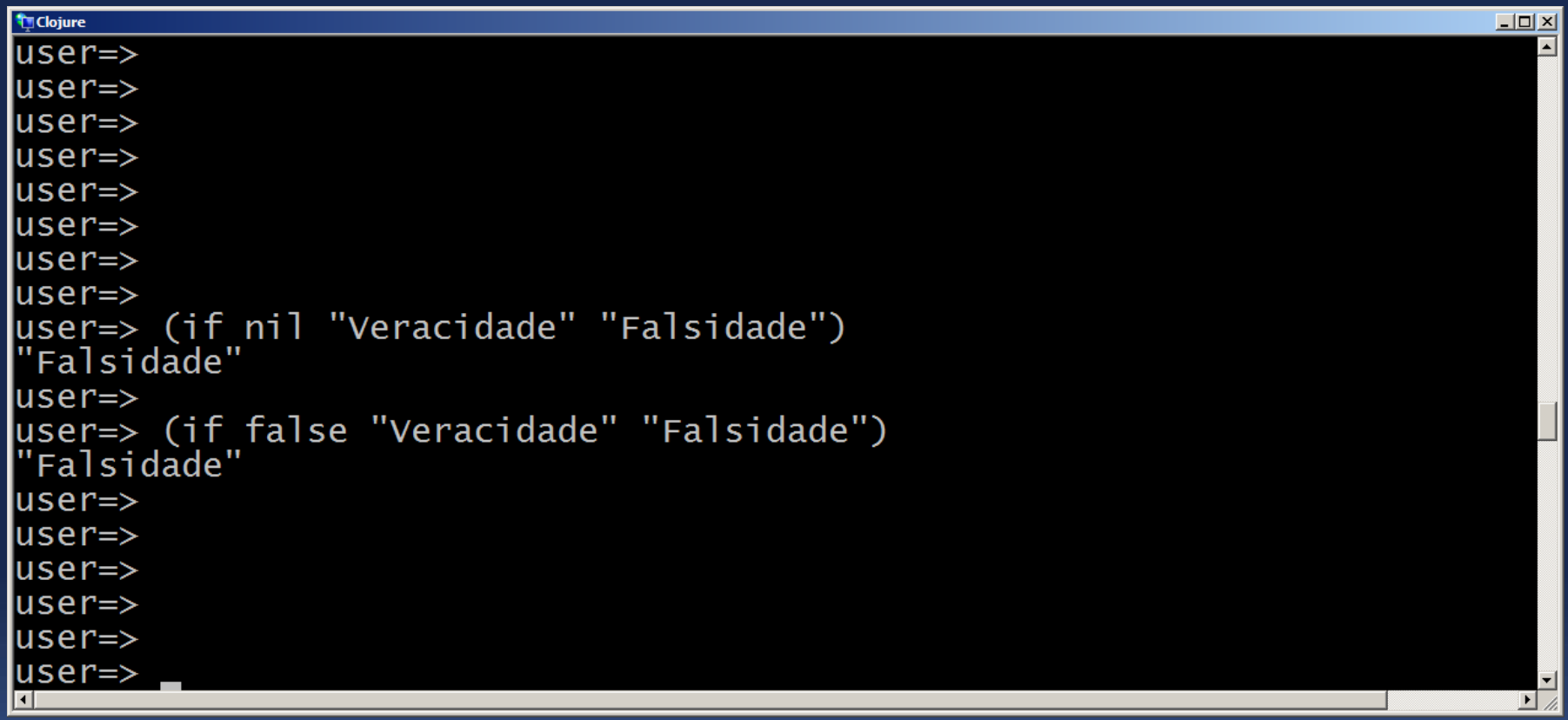


Observação: O doc-string deve vir antes da definição dos parâmetros da função. Se for escrito após, será avaliado sequencialmente como parte do corpo da função, resultando naturalmente em erro!

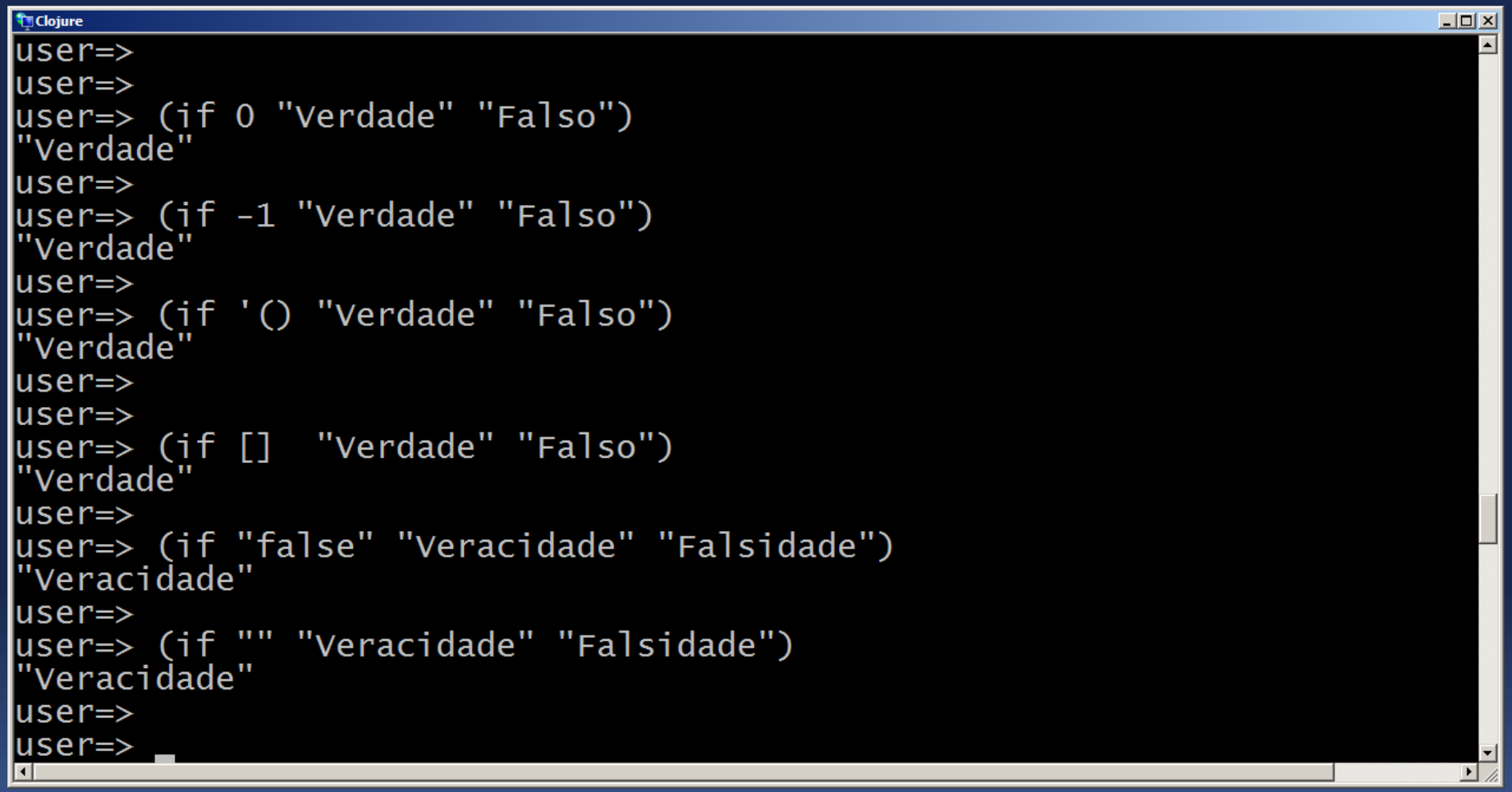
Veracidade, Falsidade e nil

- ✓ Na linguagem **Clojure**, **nil** representa ausência de valor; Corresponde à **NULL** em outras linguagens de programação;
- ✓ Representar ausência de valor é útil, pois permitir saber que "algo" está faltando;
- ✓ Em **Clojure**, **nil** se comporta como "**false**" quando for avaliado em uma expressão booleana;
- ✓ Em **Clojure**, **false** e **nil** são os **únicos** valores que são tratados como "**falsidade**". Tudo o mais é verdadeiro. Essa simples regra torna código **Clojure** mais robusto e confiável.

Veracidade, Falsidade e nil - Exemplos



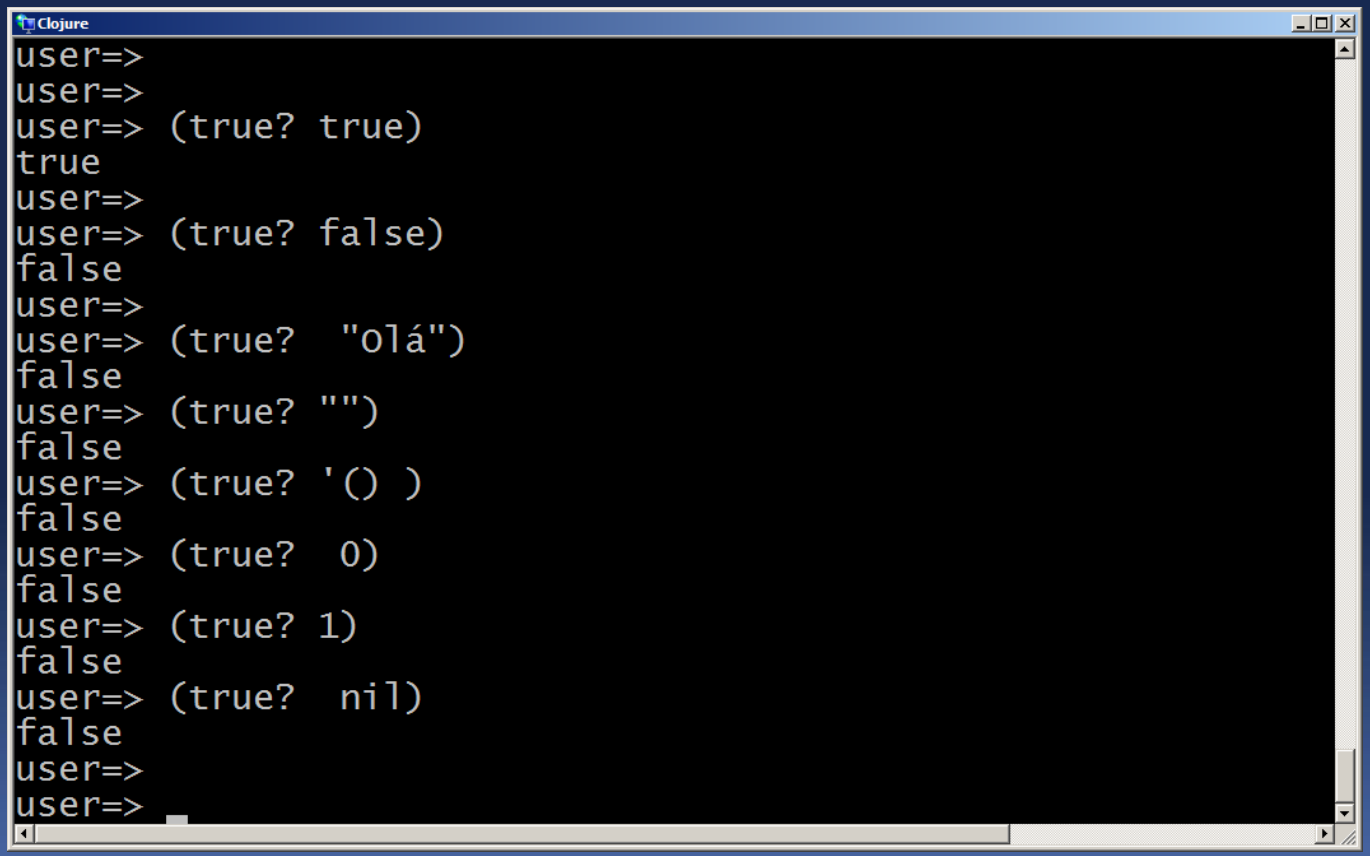
Veracidade, Falsidade e nil - Exemplos



Função true?

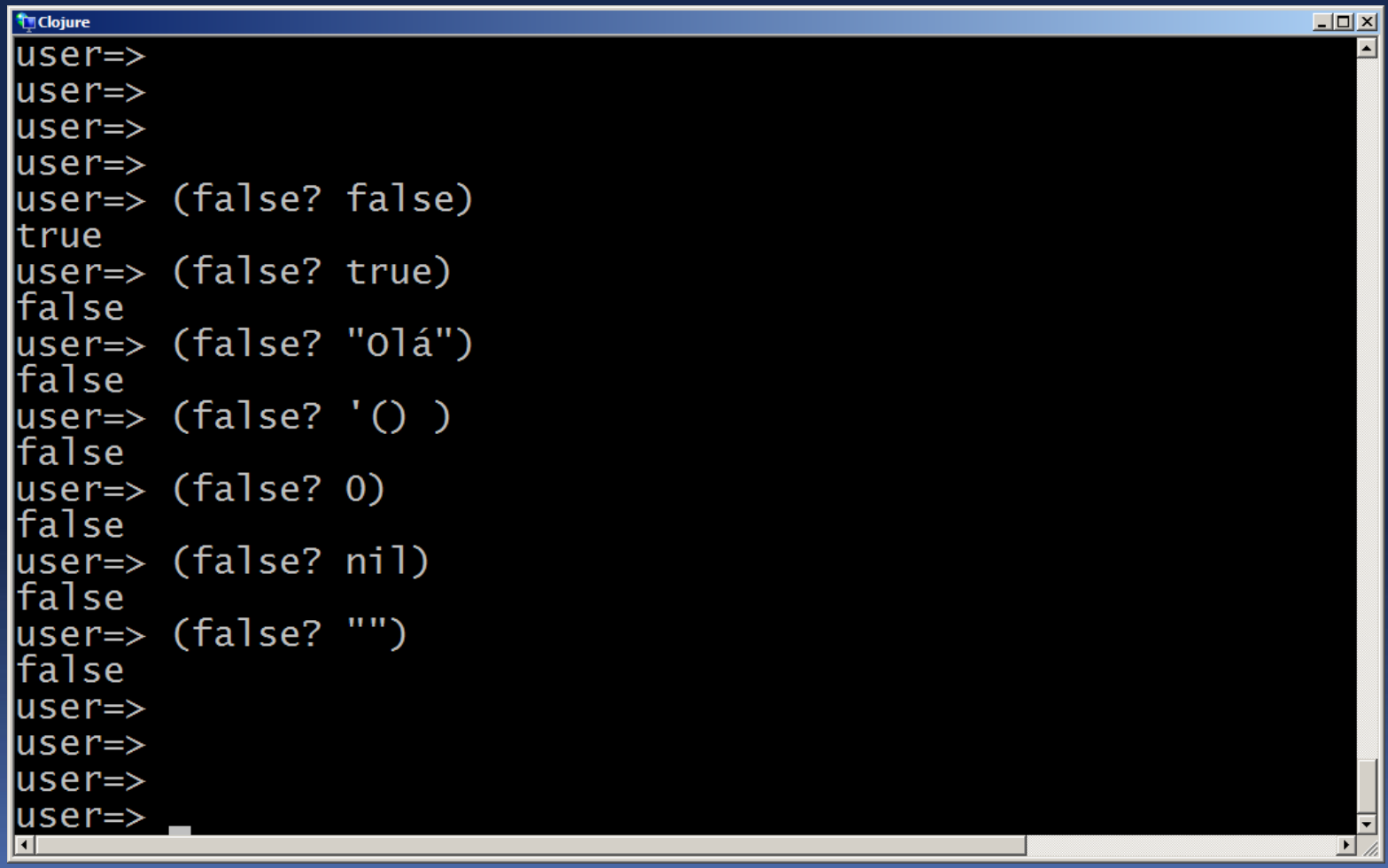
- ✓ A função **true?** retorna um valor booleano **true** ou **false** e permite checar se o parâmetro é **exatamente true** e não simplesmente tratado como **veracidade** ou **falsidade**.

- ✓ Assim, **true?** retorna true apenas quando o parâmetro for verdadeiramente **true**; Do contrário, sempre retornará **false**.



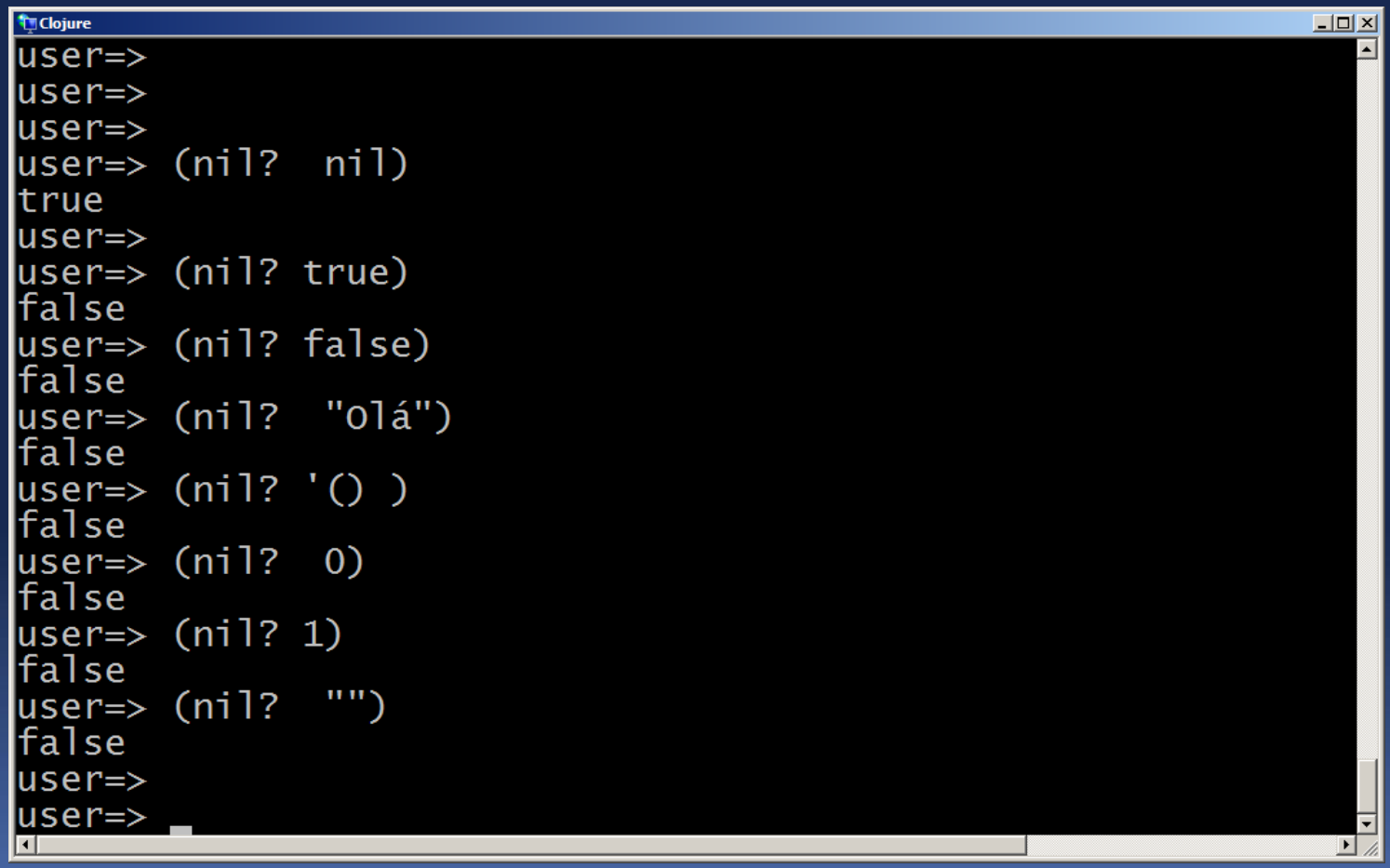
Função false?

- ✓ A função **false?** retorna um valor booleano **true** ou **false** e permite checar se o parâmetro é **exatamente false** e não simplesmente tratado como **veracidade** ou **falsidade**.
- ✓ Assim, **false?** retorna true apenas quando o parâmetro for verdadeiramente **false**; Do contrário, sempre retornará **false**.

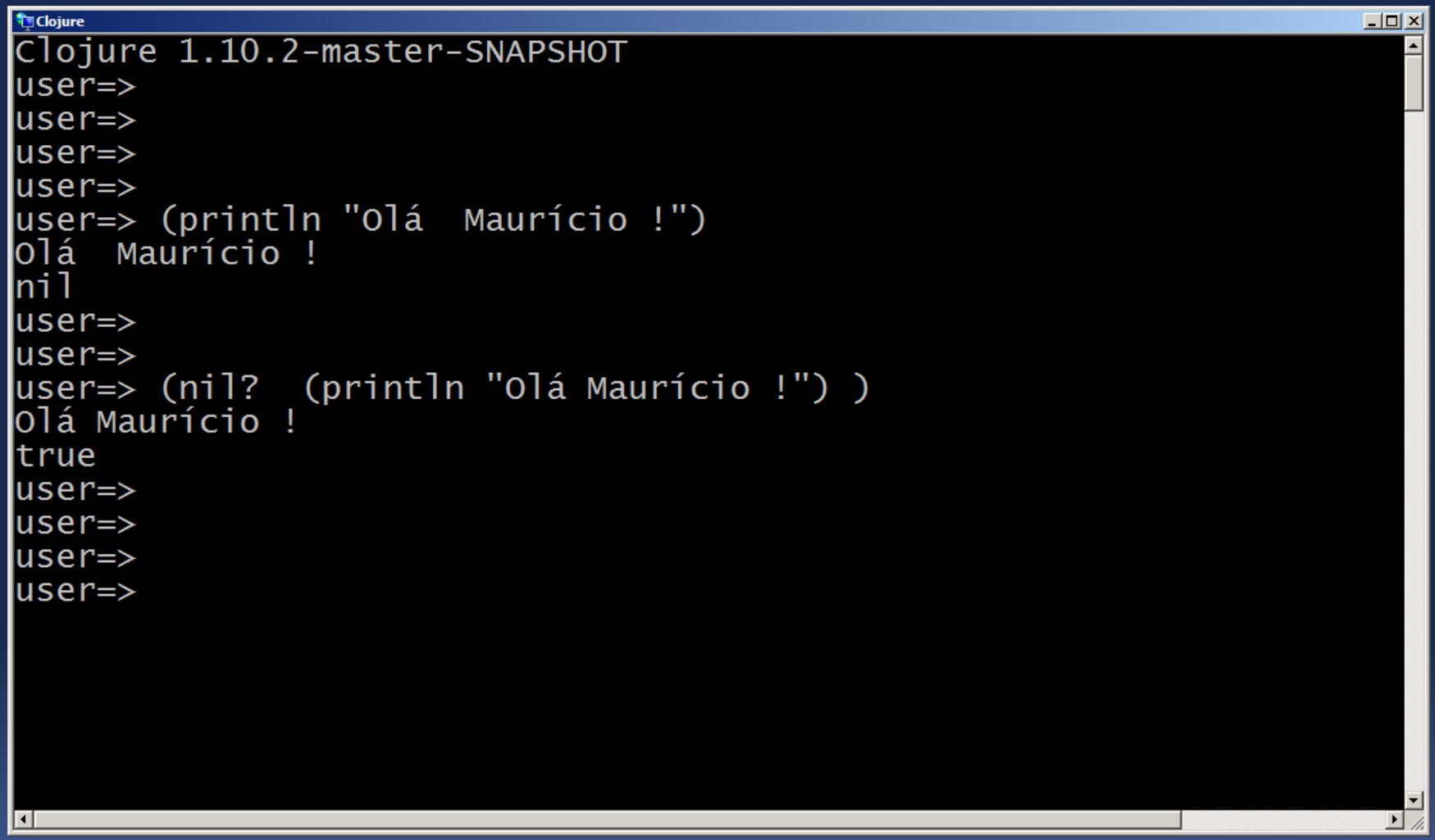


Função nil?

- ✓ A função **nil?** retorna um valor booleano **true** ou **false** e permite checar se o parâmetro é **exatamente nil** e não simplesmente tratado como **veracidade** ou **falsidade**.
- ✓ Assim, **nil?** retorna true apenas quando o parâmetro for verdadeiramente **nil**; Do contrário, sempre retornará **false**.



Exemplo interessante



Macro and

- ✓ A macro **and** retorna a primeira **falsidade** que encontrar (da esquerda para a direita) e **não** avaliará o restante da expressão quando for o caso;
- ✓ Quando todos os valores passados para a macro **and** resultarem em "veracidade", a macro **and** retorna o valor da última expressão.

```
Clojure
user=>
user=>
user=>
user=>
user=> (and "Hello Maurício !!!")
"Hello Maurício !!!"
user=>
user=> (and "Hello" "Maurício")
"Maurício"
user=>
user=> (and false "Hello" "Maurício")
false
user=>
user=> (and "Hello" false "Maurício")
false
user=>
user=> (and nil "Hello" "Maurício")
nil
user=>
user=> (and  (println "Hello Maurício") (println "Você está bem? ") )
Hello Maurício
nil
user=>
user=>
user=>
```

Macro or

- ✓ A macro **or** retorna a primeira **veracidade** que encontrar (da esquerda para a direita) e **não** avaliará o restante da expressão quando for o caso;
- ✓ Quando todos os valores passados para a macro **or** resultarem em "falsidade", a macro **or** retorna false ou nil, dependendo do caso.

```
Clojure
user=>
user=>
user=>
user=> (or "Hello")
"Hello"
user=> (or "Hello" "Maurício")
"Hello"
user=> (or false "Hello")
"Hello"
user=> (or false "Hello" "Maurício")
"Hello"
user=> (or false)
false
user=> (or nil)
nil
user=> (or false nil)
nil
user=> (or false nil '() )
()
user=> (or false nil false nil 0)
0
user=> (or true (println "Hello Maurício !") )
true
user=>
user=>
```

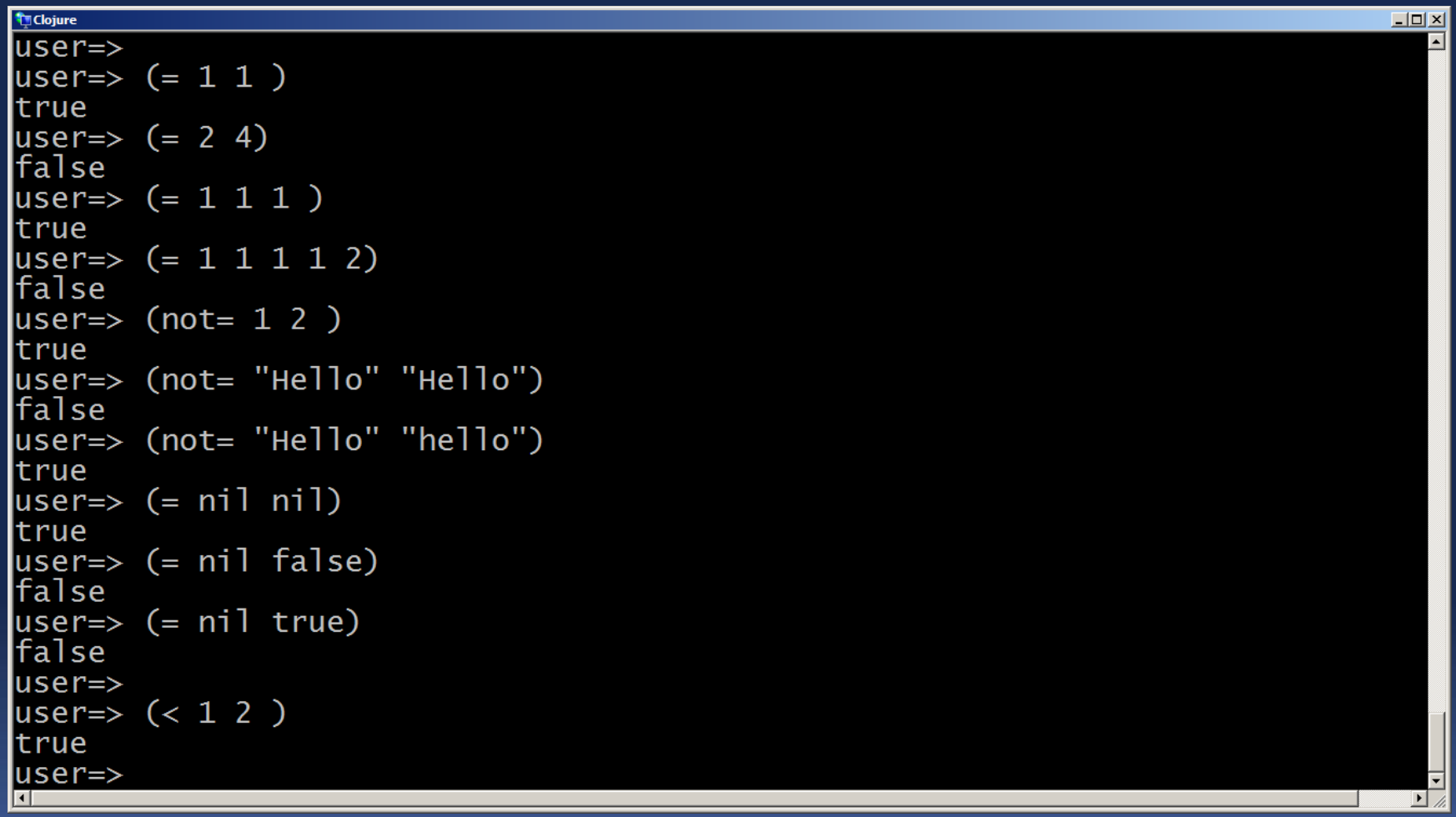
Operações de Equalidade

- ✓ Na maioria das linguagens de programação o símbolo **=** é usado para operações de atribuição. Porém em **Clojure**, bindings de símbolos à valores são feitos pelas special forms **def** e **defn**;

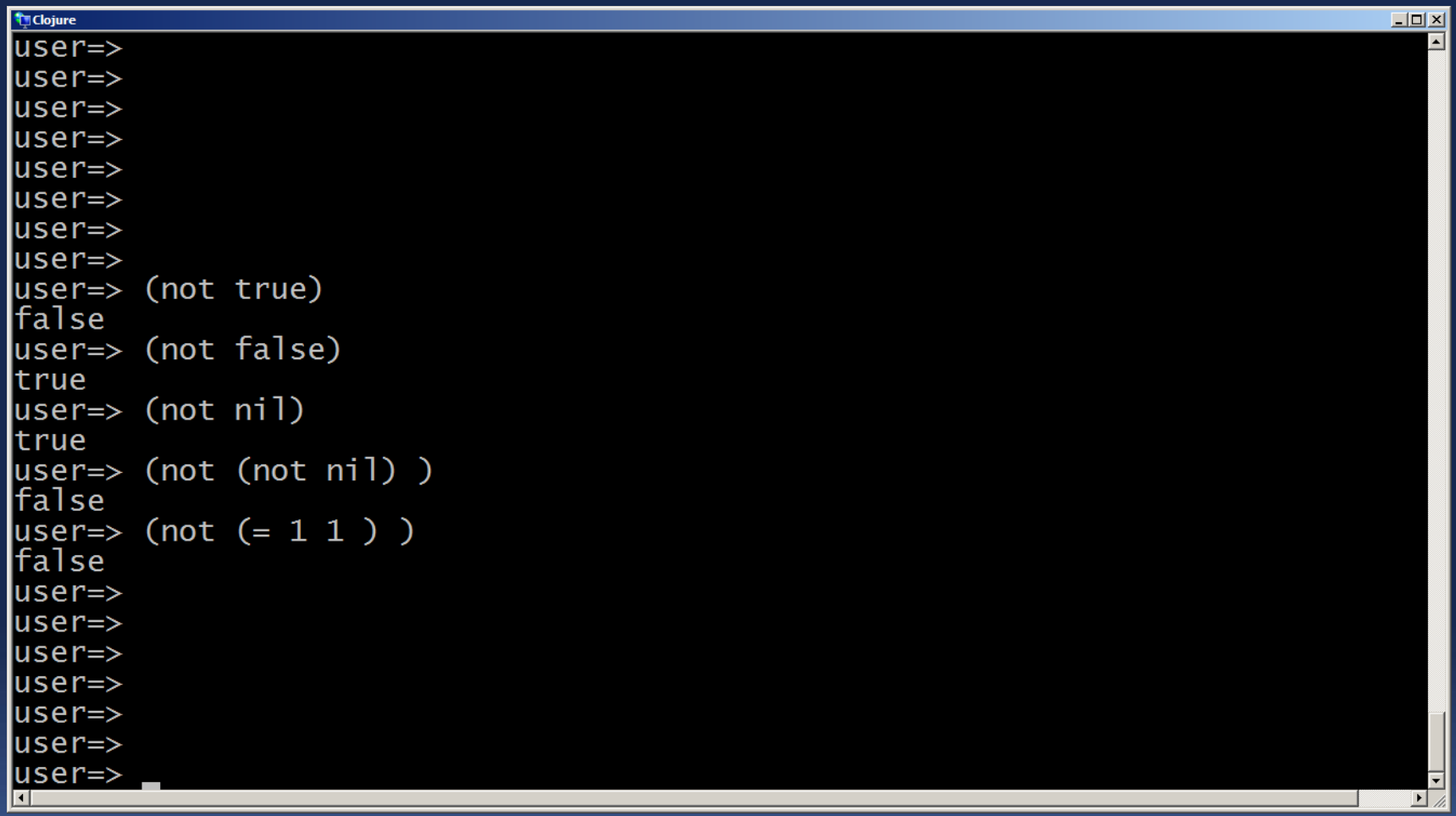
- ✓ Em **Clojure**, o símbolo **=** está associado à uma função para equalidade e retorna **true** se todos os argumentos forem **iguais**.

- ✓ Todos as outras operações de equalidade (**>** , **>=** , **<** , **<=**) são também implementações de funções.

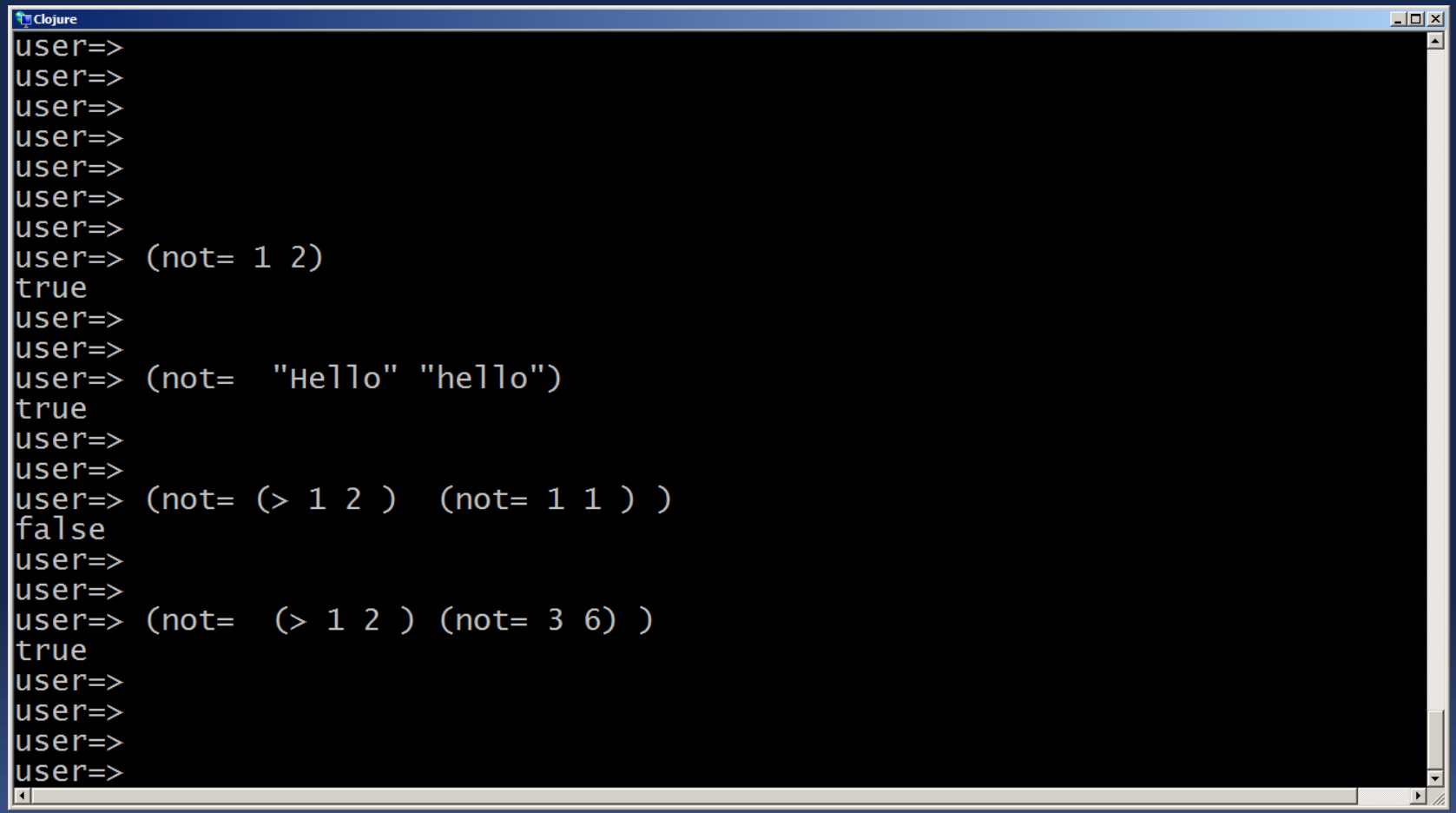
Operações de Equalidade



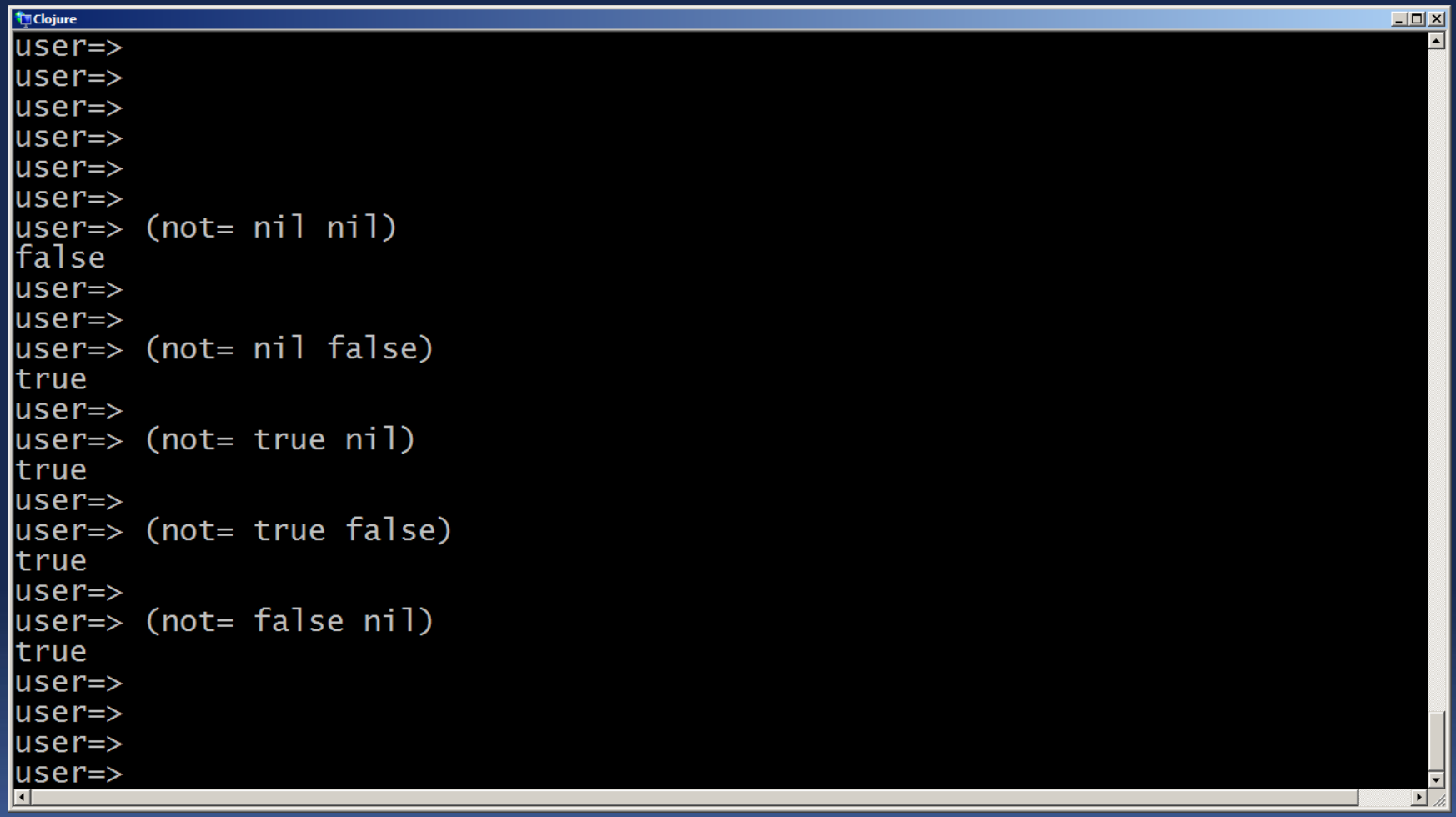
Operações de Equalidade



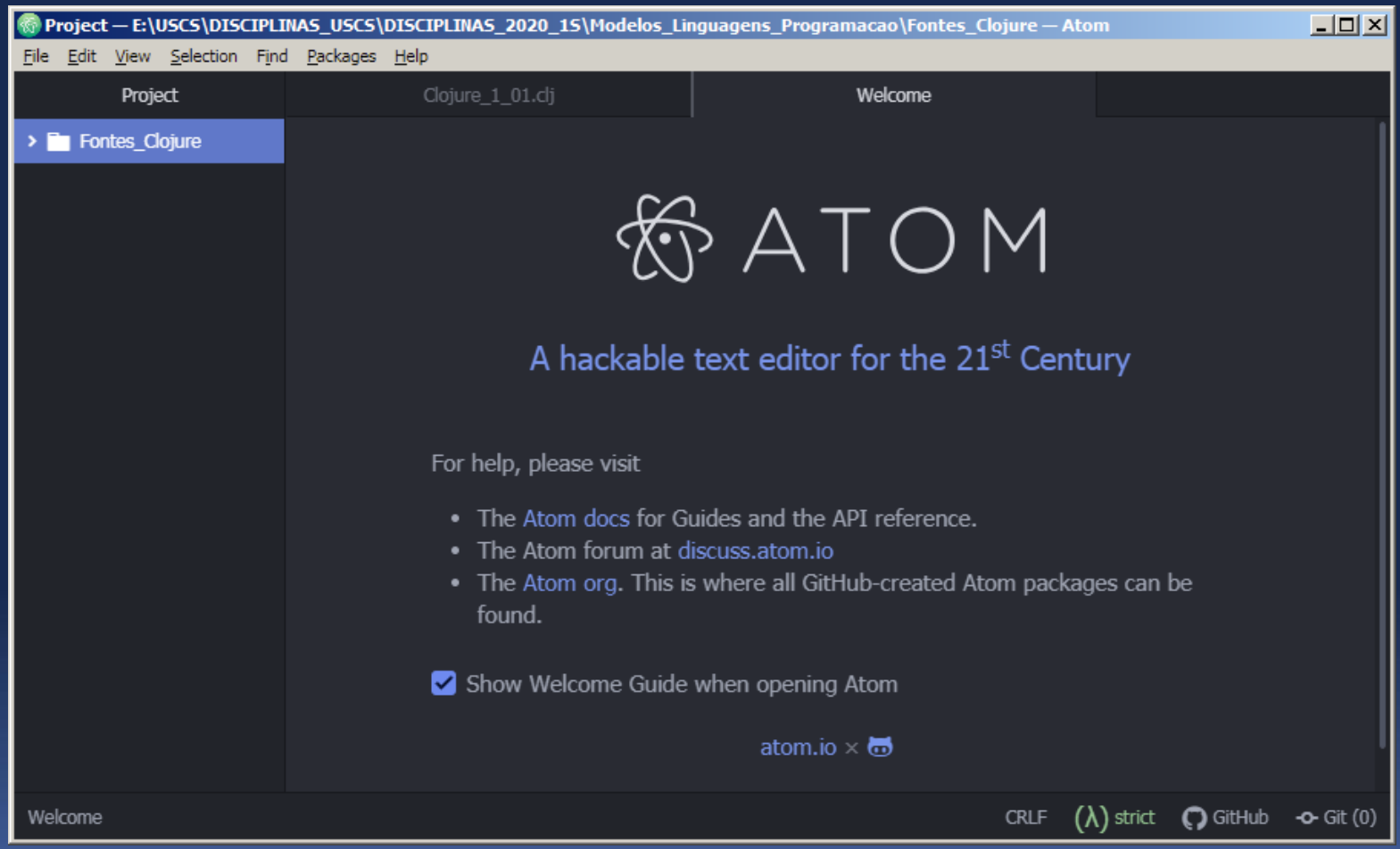
Operações de Equalidade



Operações de Equalidade



Execução de Código Clojure no Atom Package Chlorine

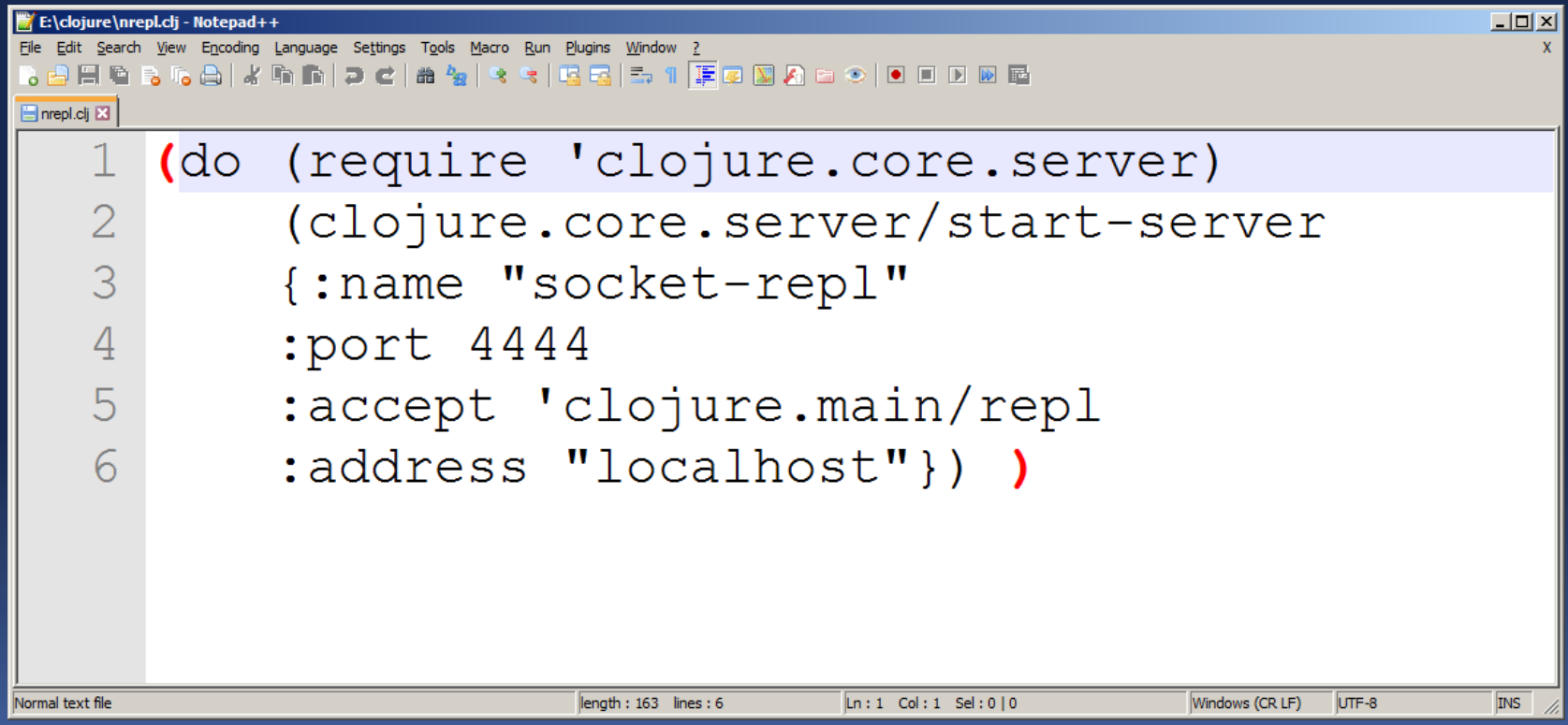


Ativando REPL Remoto

```
(do (require 'clojure.core.server)
(clojure.core.server/start-server
{:name "socket-repl"
:port 4444
:accept 'clojure.main/repl
:address "localhost"}))
```

Salvando código num file .clj

- ✓ Criar arquivo .clj com código
- ✓ Estamos chamando o arquivo de nrepl.clj
- ✓ Será salvo no diretório: E:/clojure

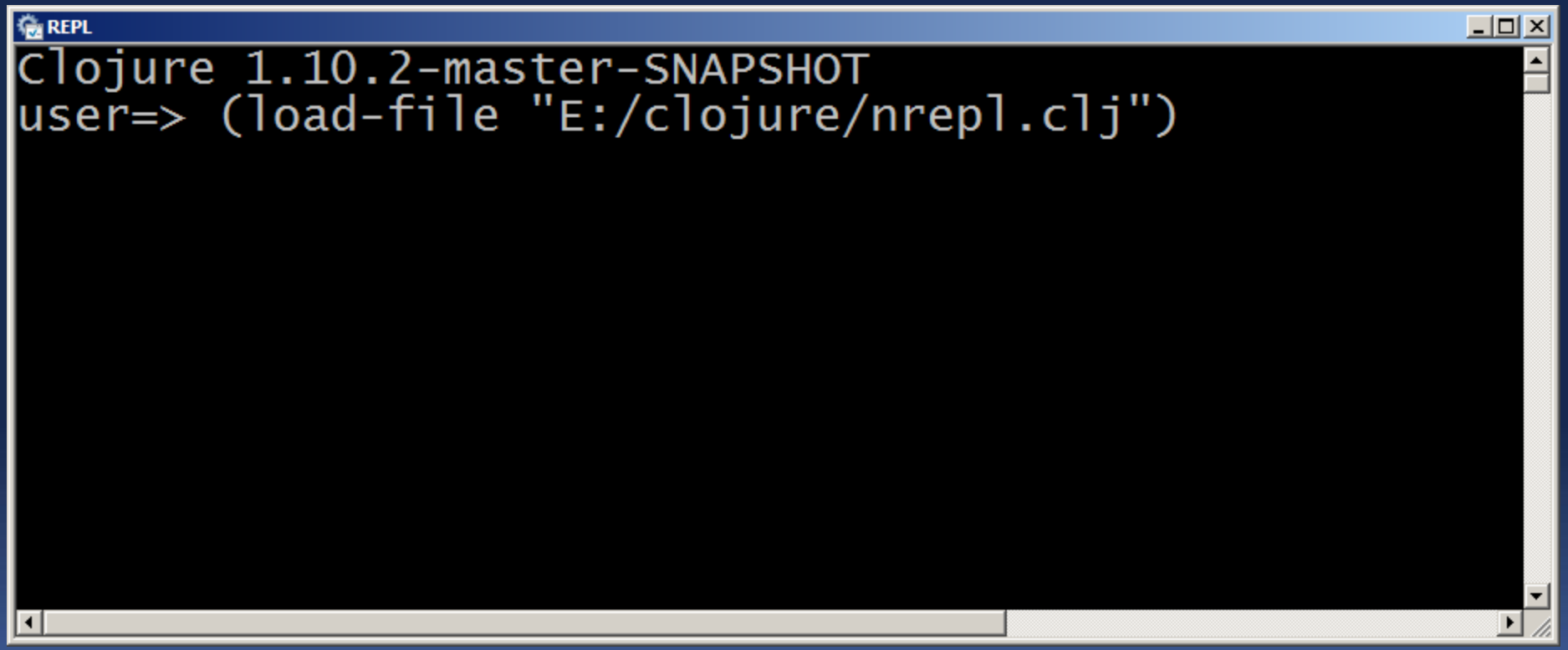


Ativando REPL Remoto
Digitando-se o código na console

```
Clojure
Clojure 1.10.2-master-SNAPSHOT
user=> (do (require 'clojure.core.server)
(clojure.core.server/start-server
{:name "socket-repl"
:port 4444
:accept 'clojure.main/repl
:address "localhost"}))
#object[java.net.ServerSocket 0x43f82e78 "ServerSocket[addr=localhost/127.0.0.1,
localport=4444]"]
user=>
```

Ativando REPL Remoto
Carregando-se arquivo digitado anteriormente

> (load-file “E:/clojure/nrepl.clj”)



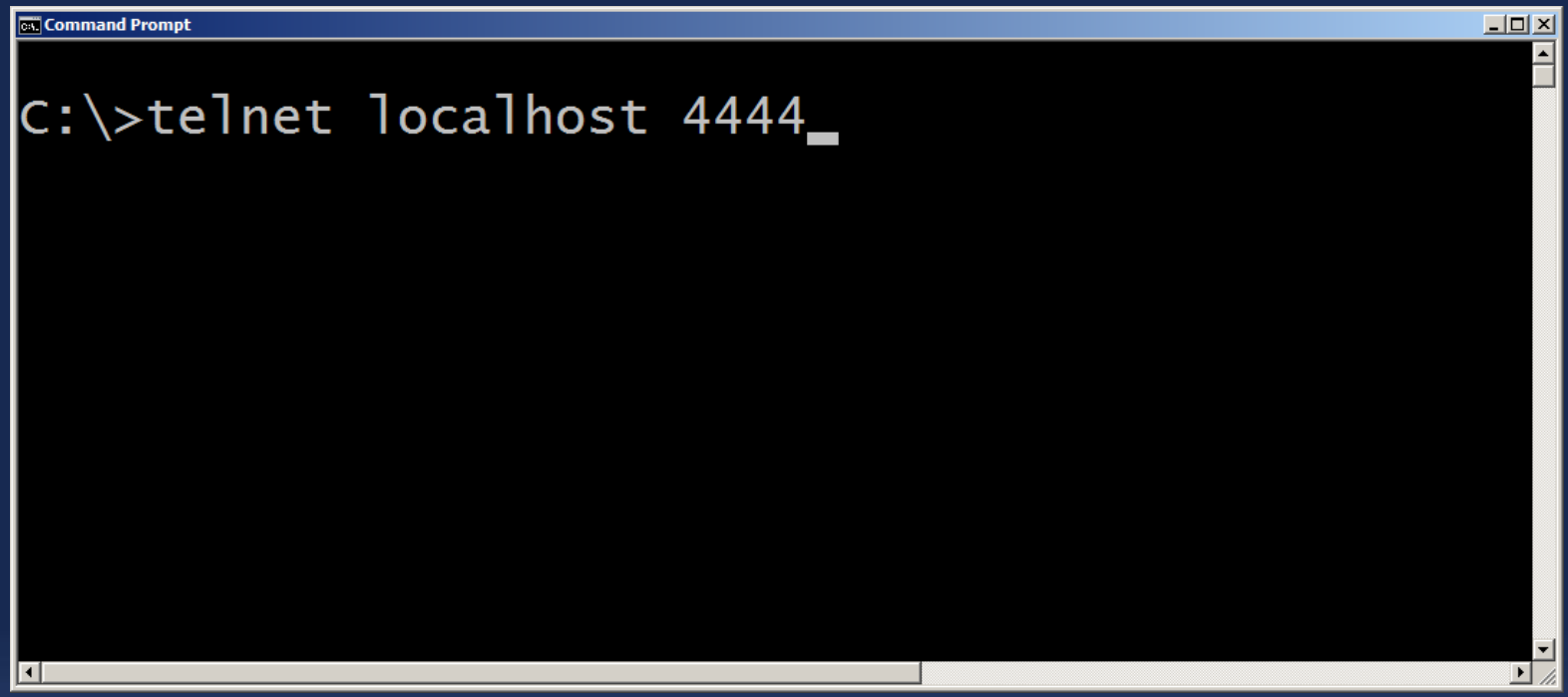
Ativando REPL Remoto
Carregando-se arquivo digitado anteriormente

> (load-file "E:/clojure/nrepl.clj")

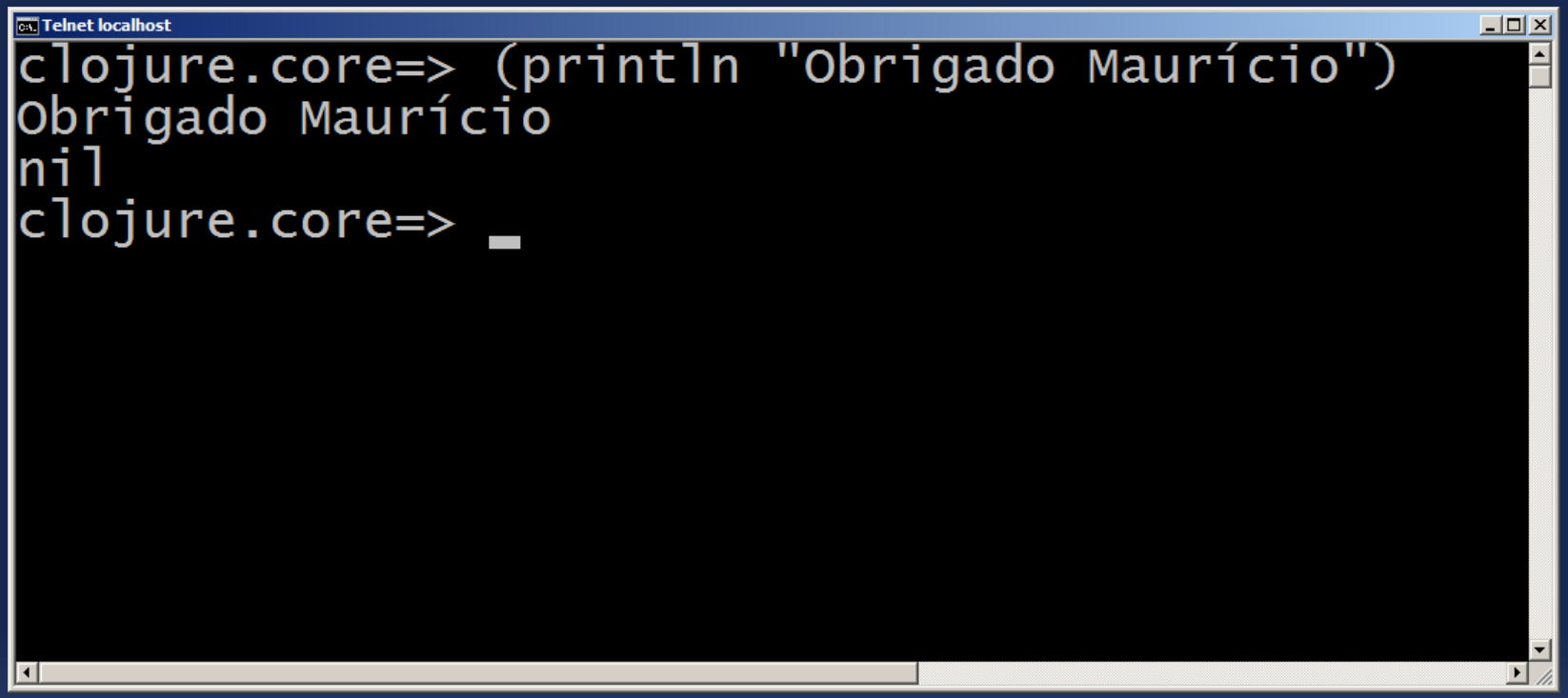
REPL

```
Clojure 1.10.2-master-SNAPSHOT
user=> (load-file "E:/clojure/nrepl.clj")
#object[java.net.ServerSocket 0x12ed9db6 "ServerSocke
localport=4444]"]
user=>
```

Conectando com Telnet na porta 4444

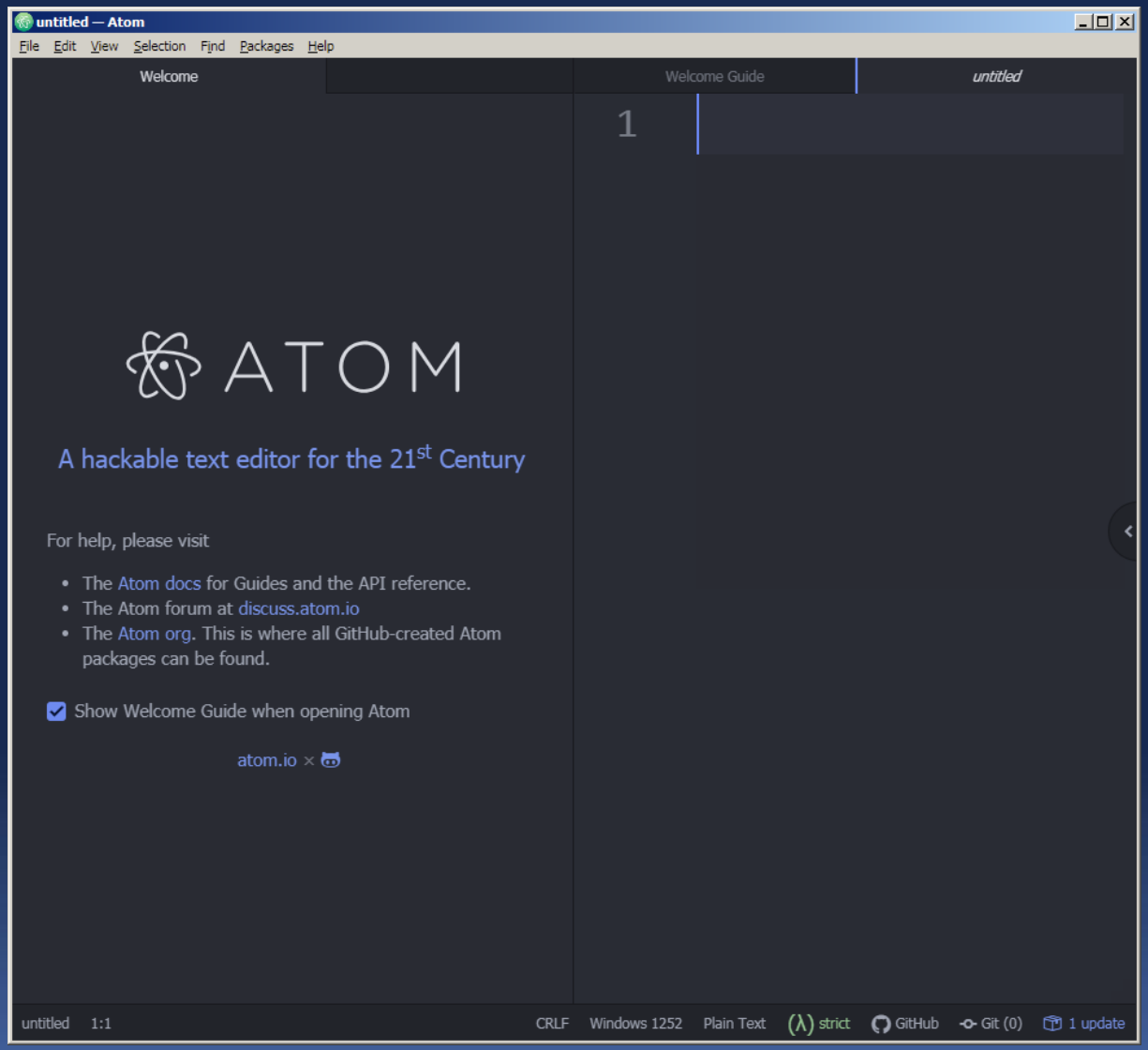


Conectando com Telnet na porta 4444



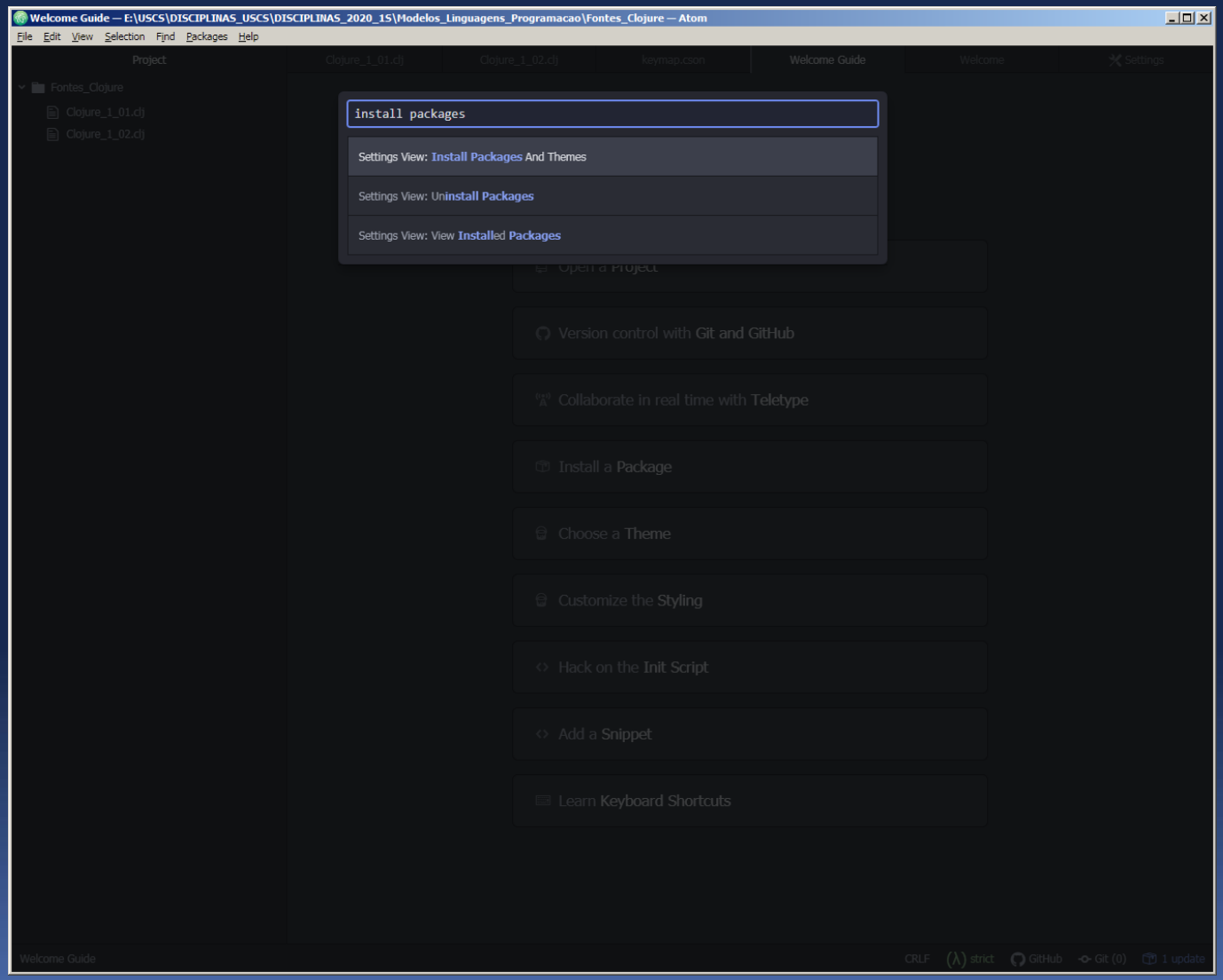
Desenvolvimento Clojure com a IDE Atom

Código Clojure no Atom



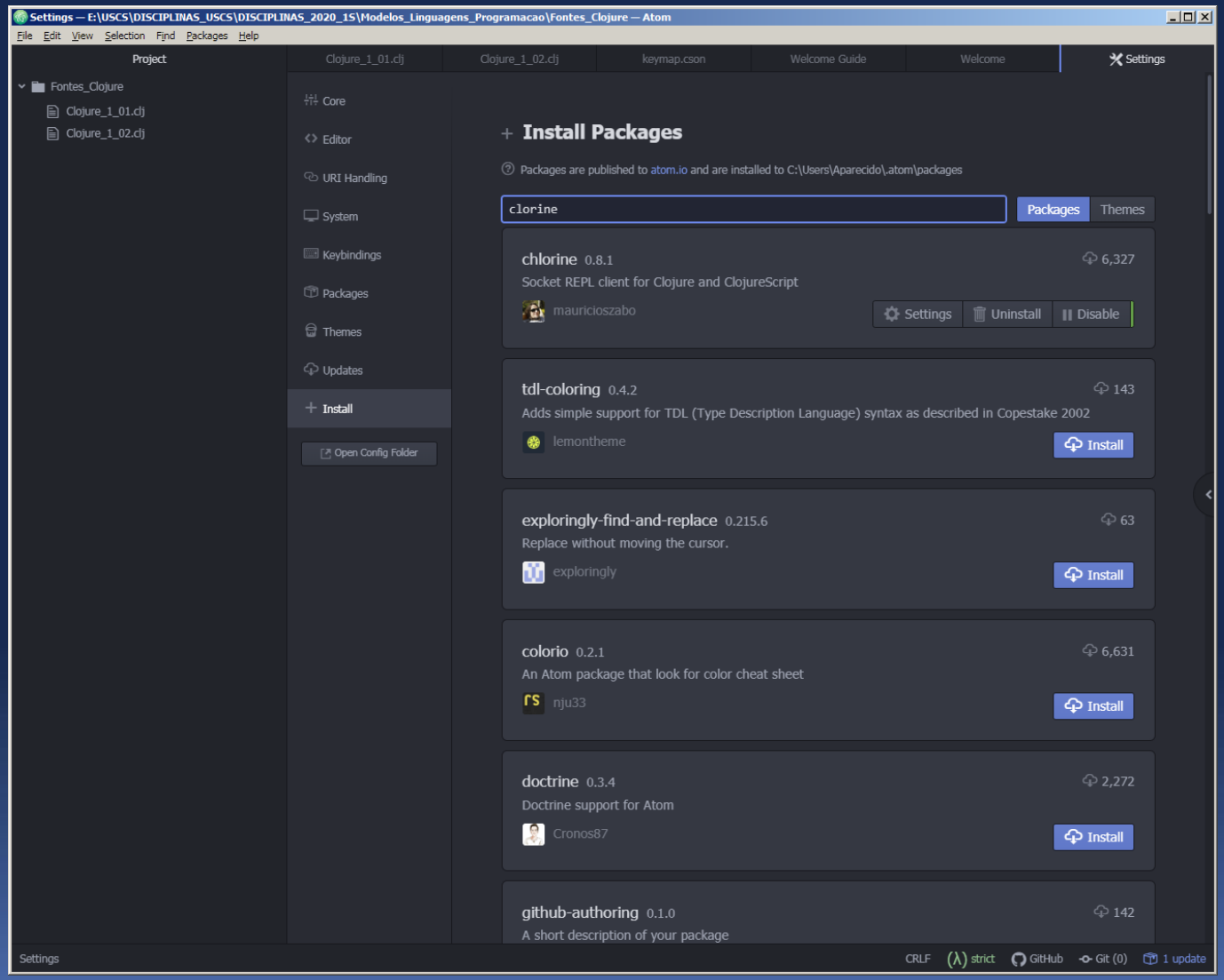
Atom – Package Chlorine

✓ Ctrl + Shift + p > Install Packages and Themes



Atom – Package Chlorine

✓Instalar => package Chlorine



Chlorine – Configuração de Hot Keys

- ✓ Instalar => package **Chlorine**

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

  - ✓ Nesse arquivo podem ser incluídas as configurações:

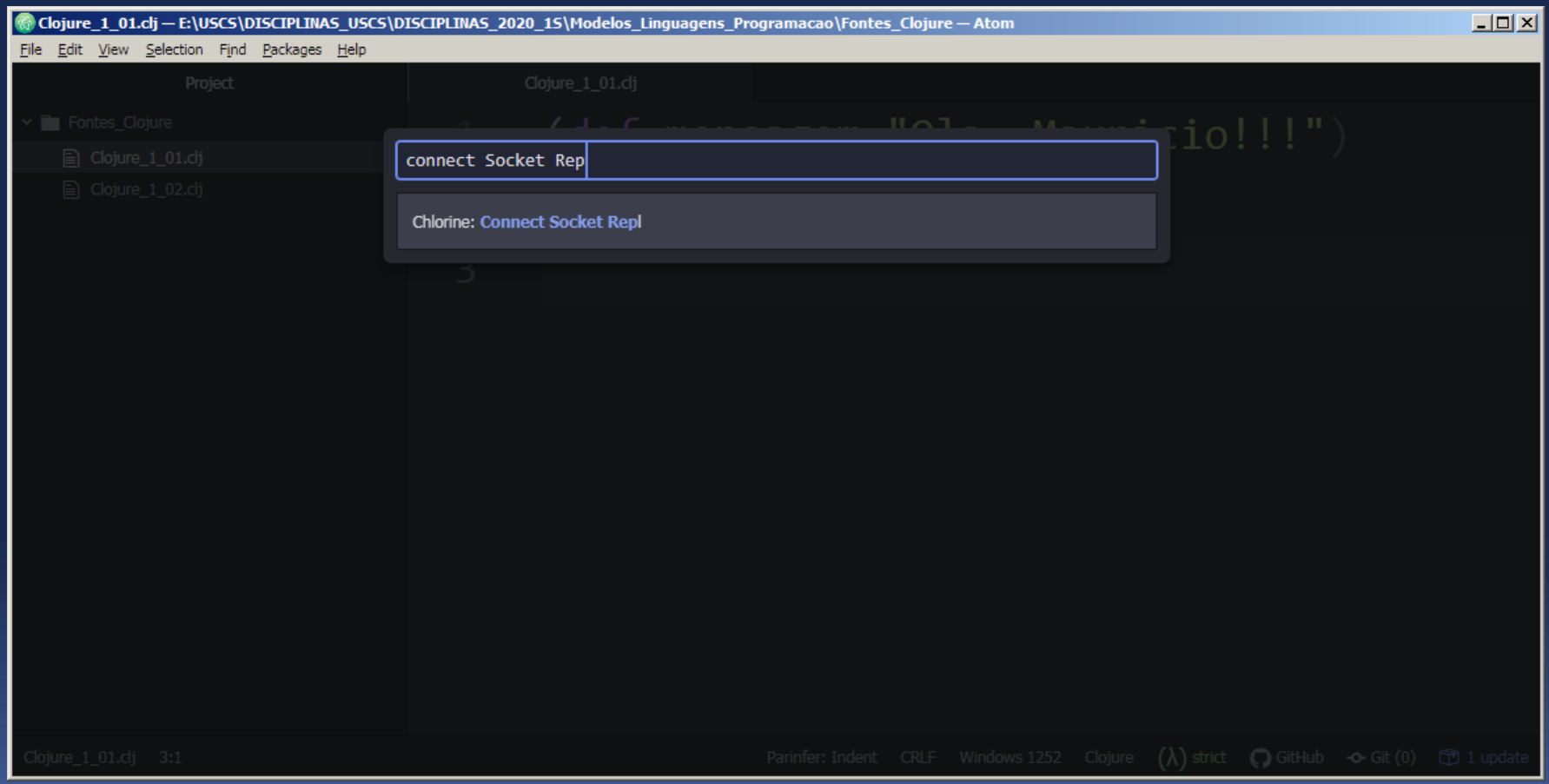
```
'atom-text-editor':
        'ctrl-k':               'chlorine:clear-console'
        'ctrl-shift-I':         'chlorine:load-file'
        'ctrl-shift-l':         'chlorine:clear-inline-results'
        'ctrl-shift-enter':     'chlorine:evaluate-block'
        'ctrl-enter':           'chlorine:evaluate-top-block'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

Chlorine – Configuração de Hot Keys

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

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

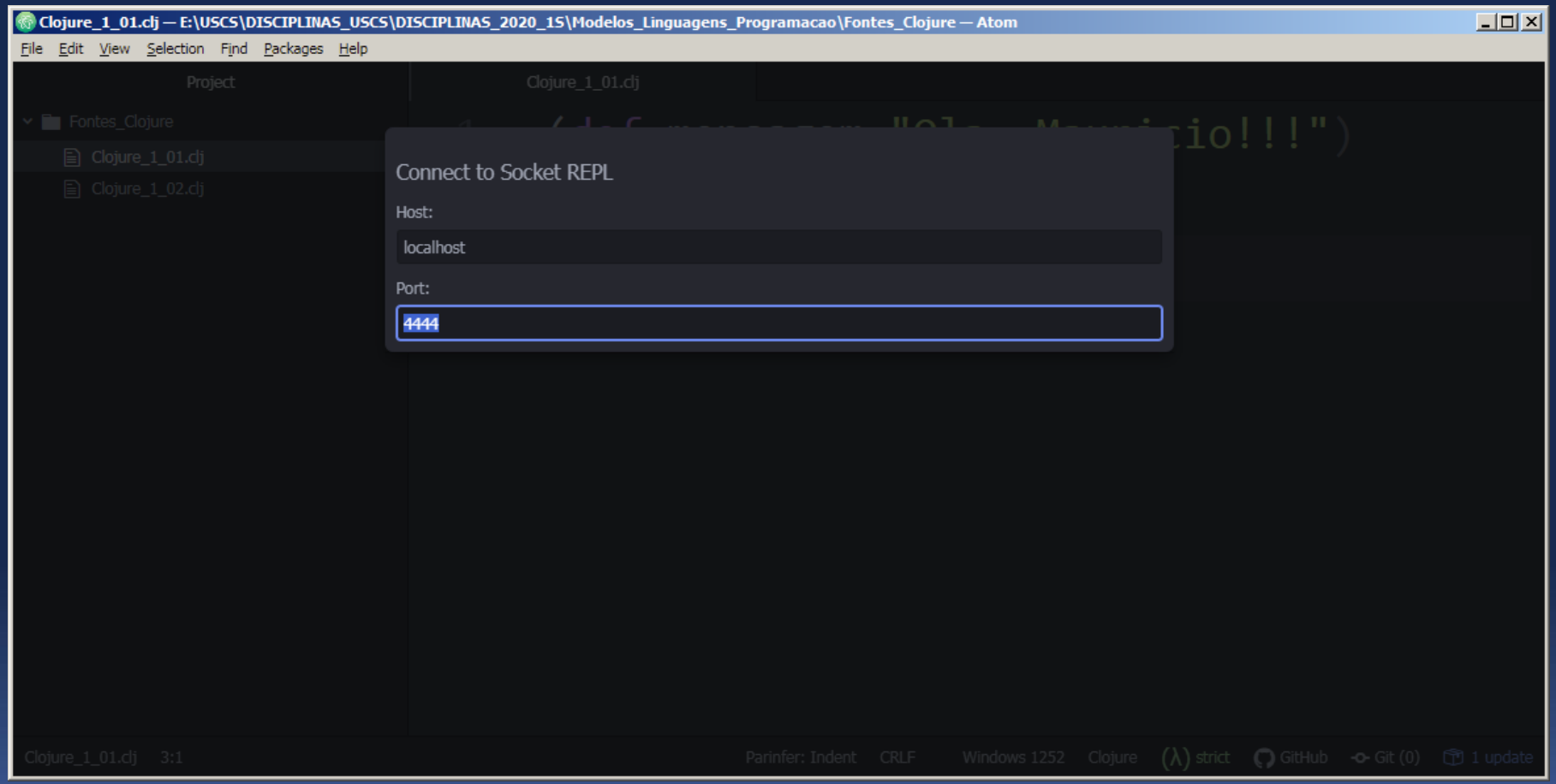
Atom – Package Chlorine

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



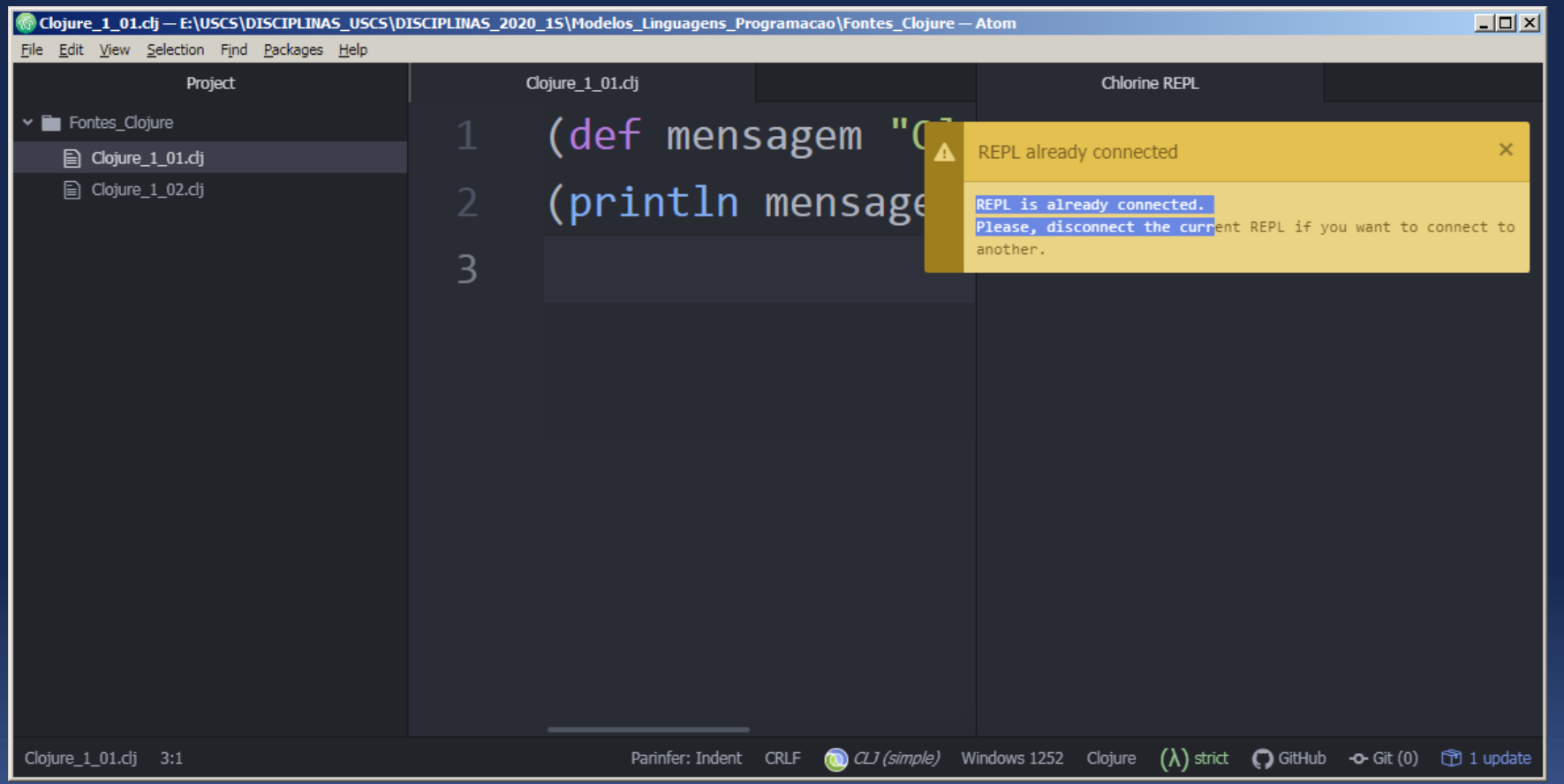
Atom – Package Chlorine

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



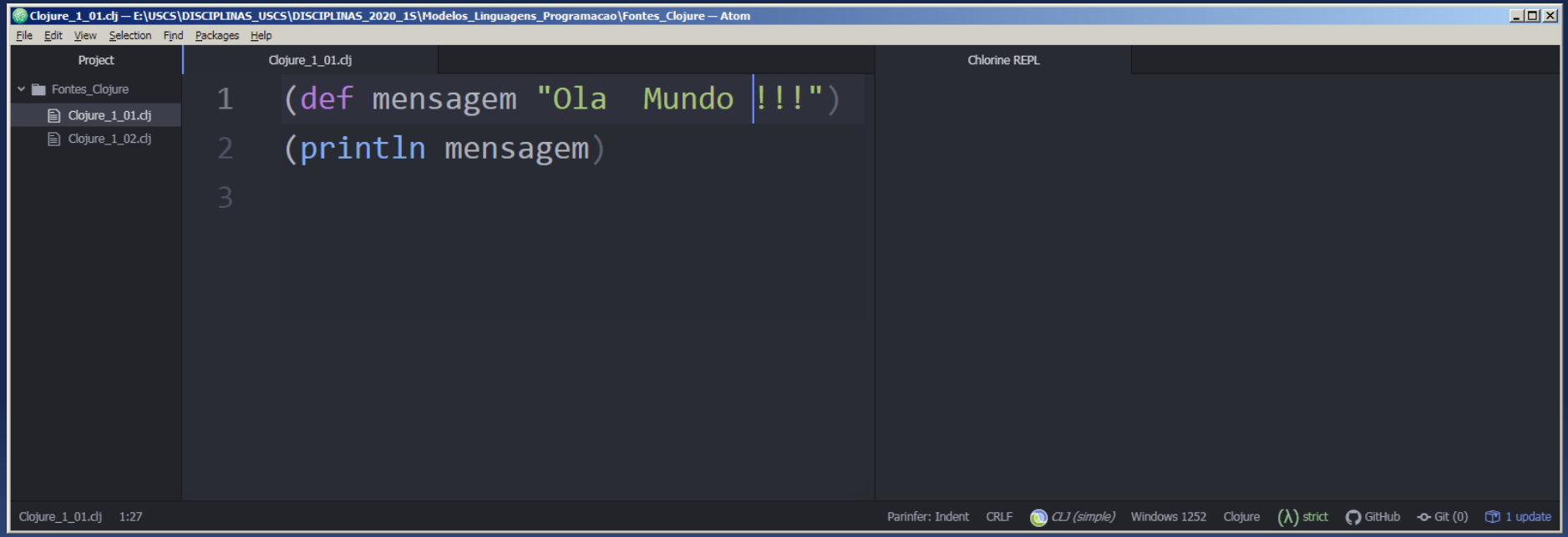
Atom – Package Chlorine

✓REPL connected !!!



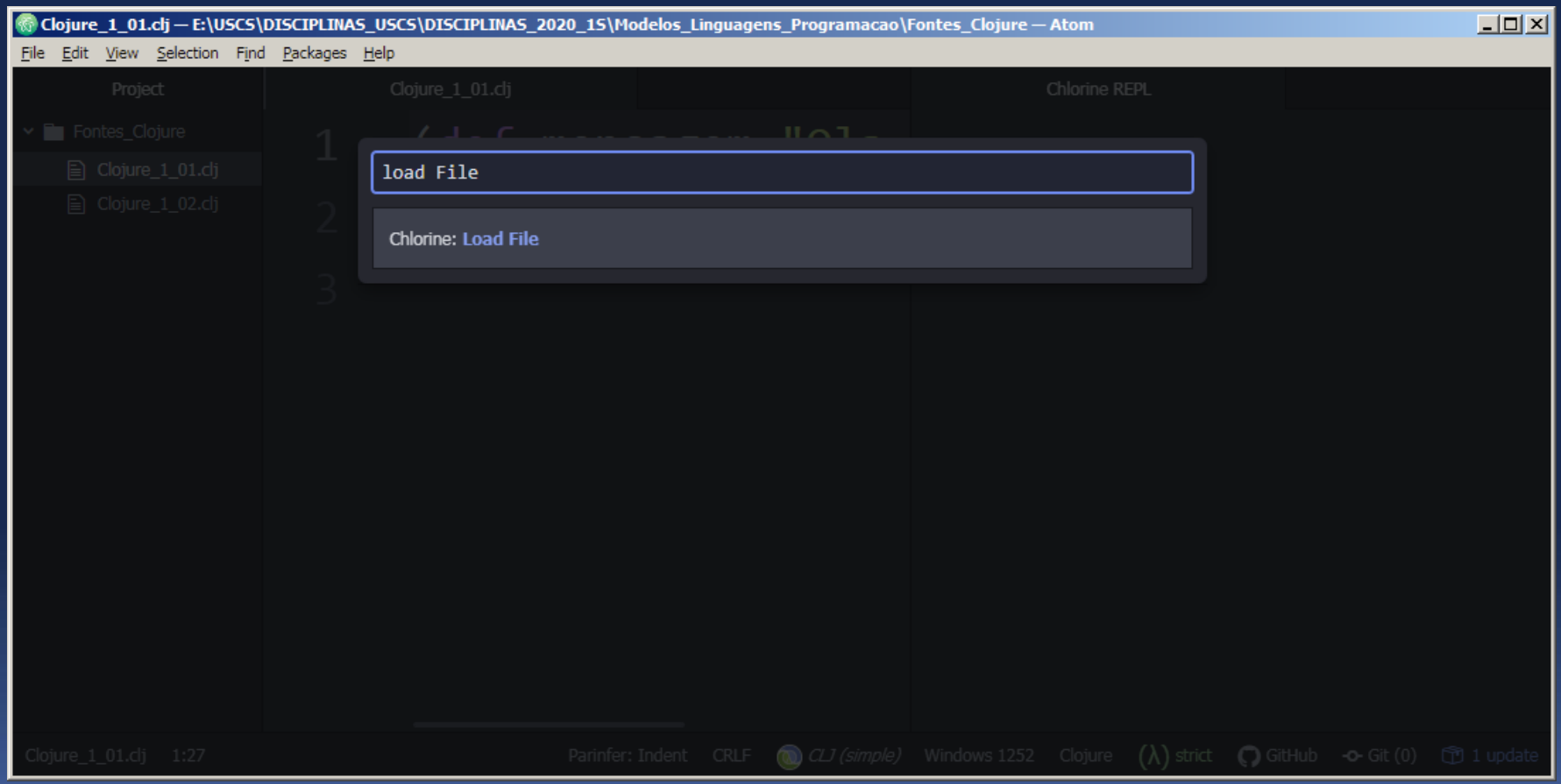
Atom – Package Chlorine

- ✓ Escrever código Clojure (extensão .clj)



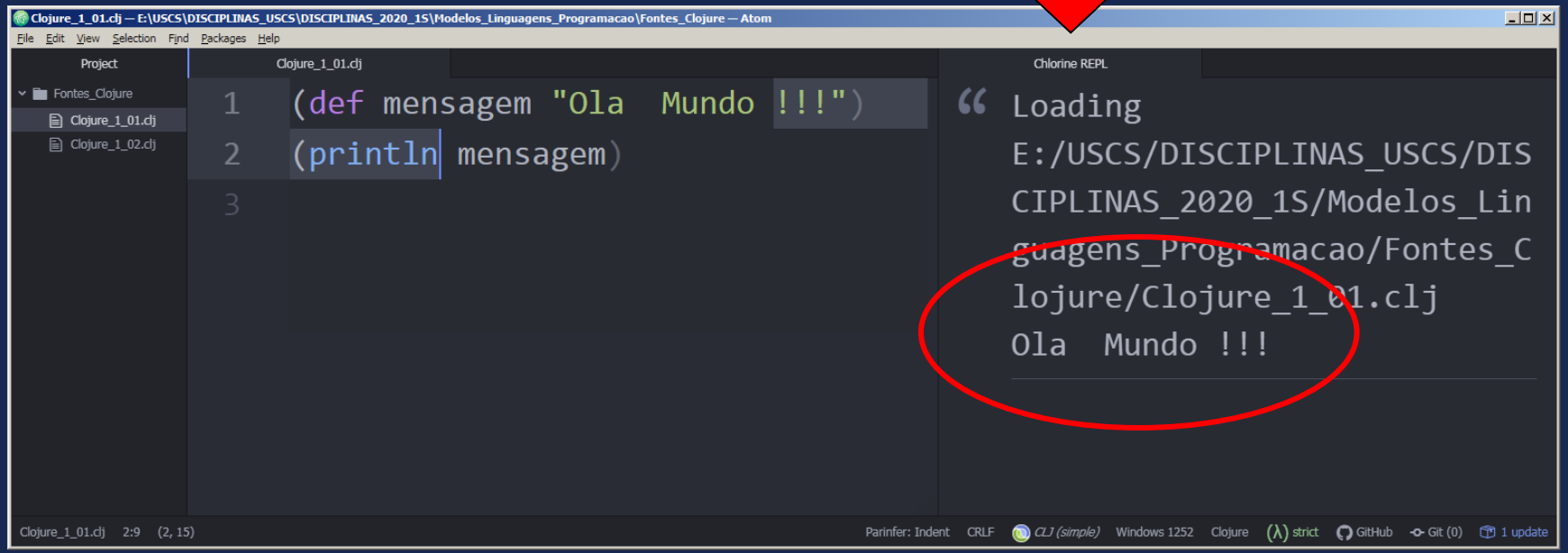
Atom – Package Chlorine

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



Atom – Package Chlorine

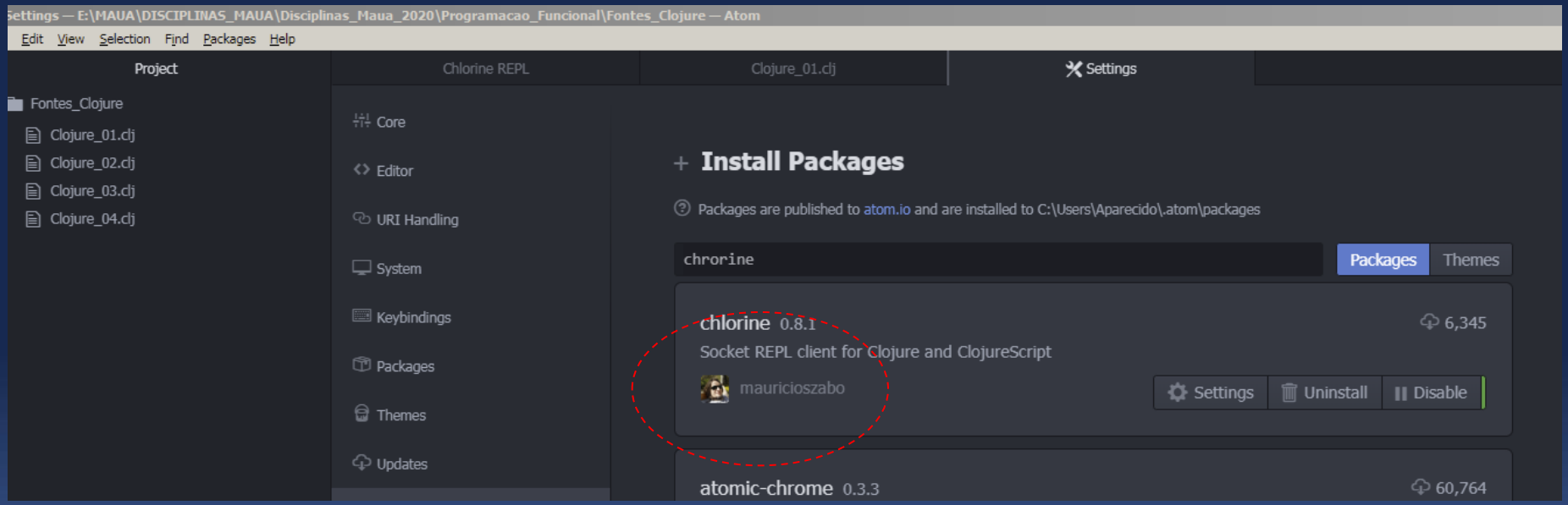
✓ Carga do código Clojure para o REPL



Atom – Package Chlorine + nREPL (Leiningen)

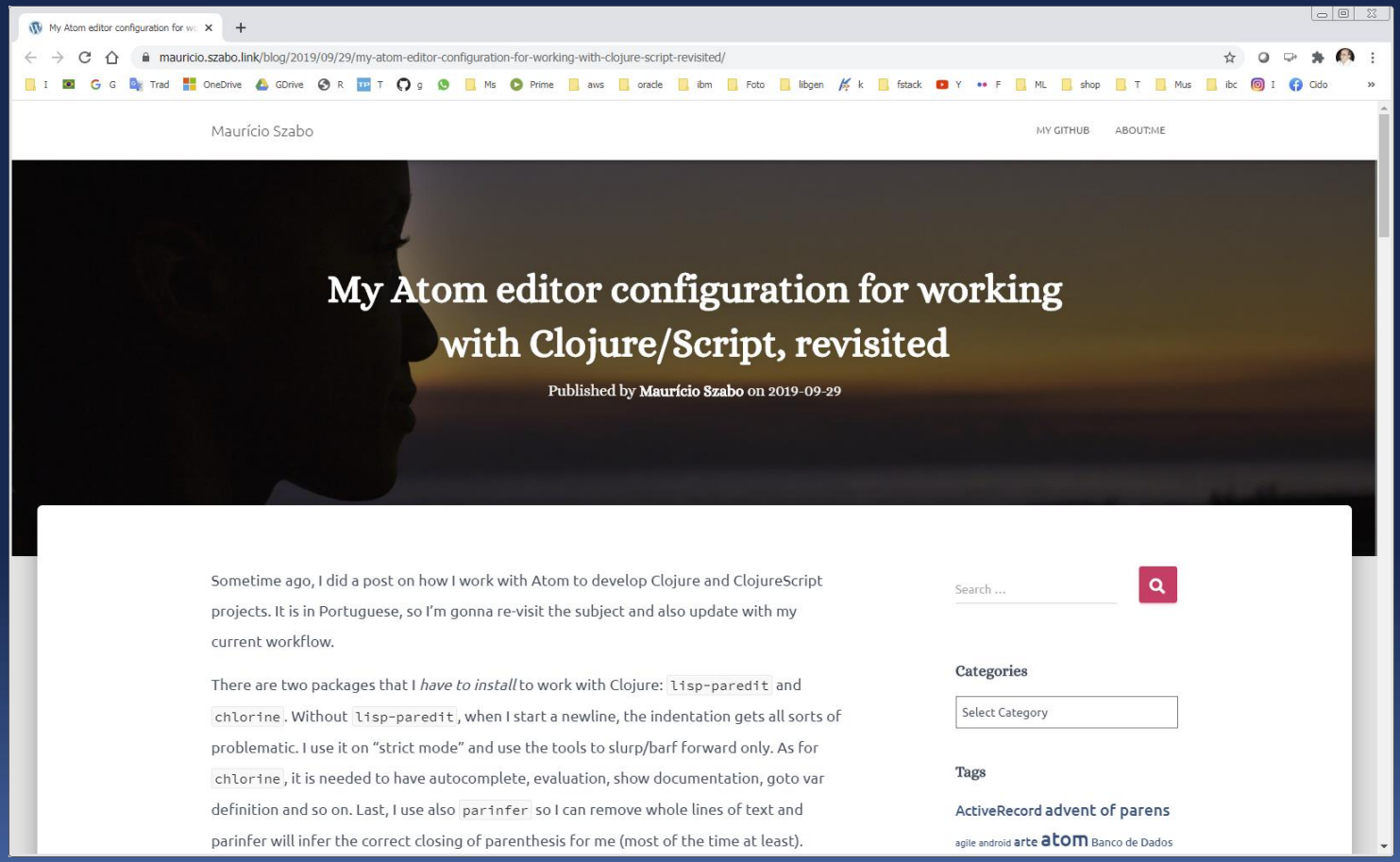
Configuração do Package Chlorine no Atom

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



Configuração do Package Chlorine no Atom

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



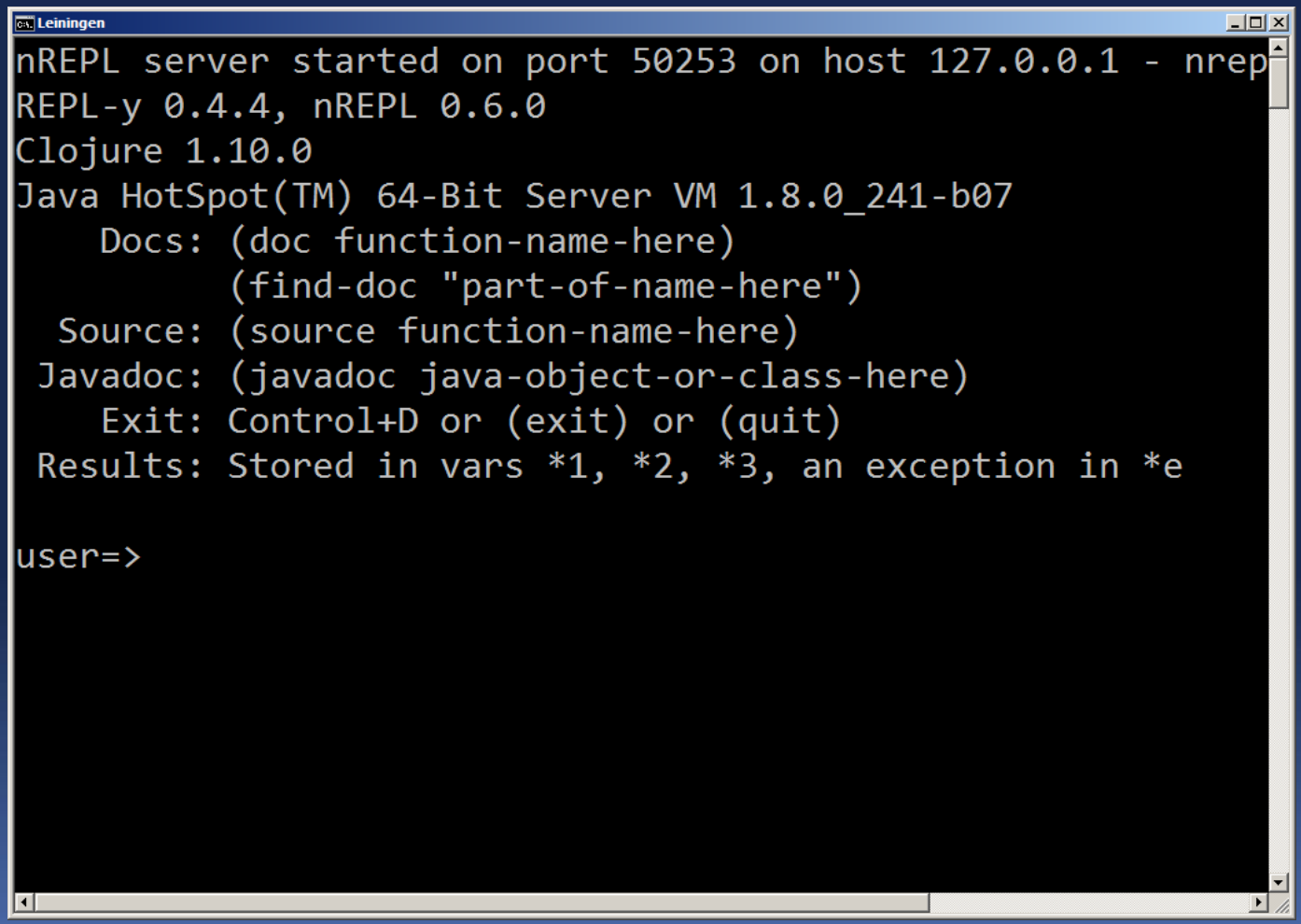
Configuração do Package Chlorine no Atom

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

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

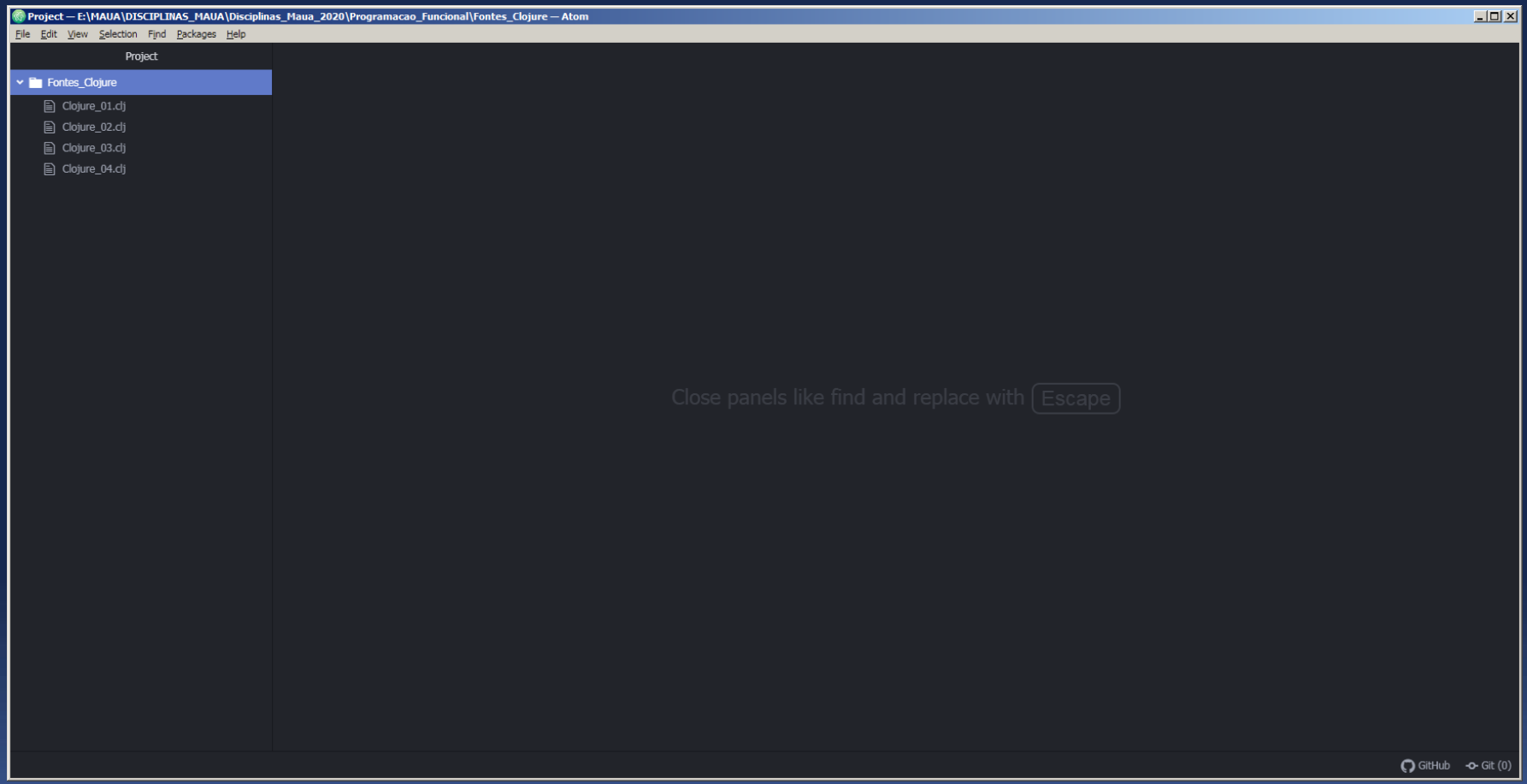
Ativar nREPL - Leiningen

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



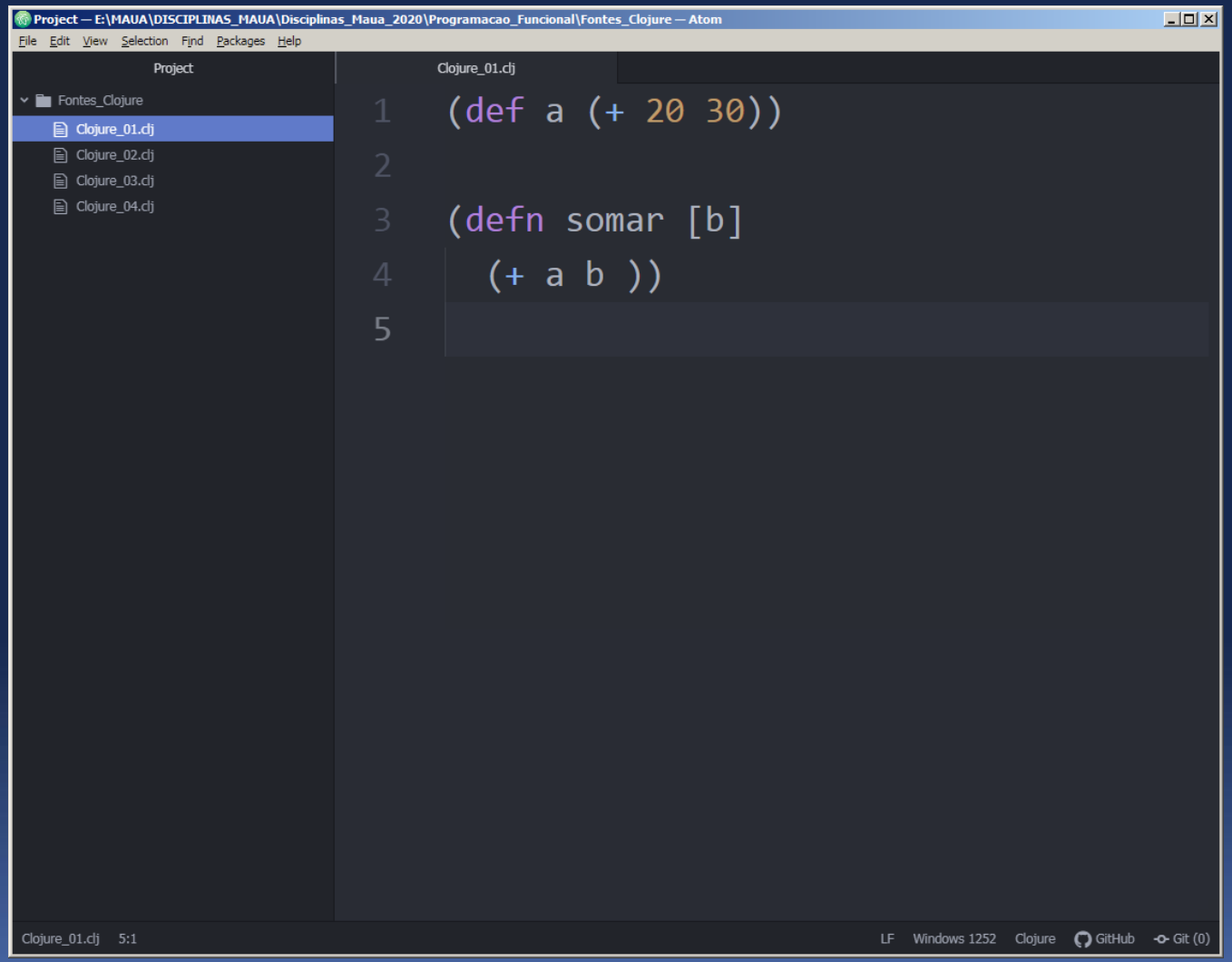
Ativar Atom

- ✓ File > Open Folder



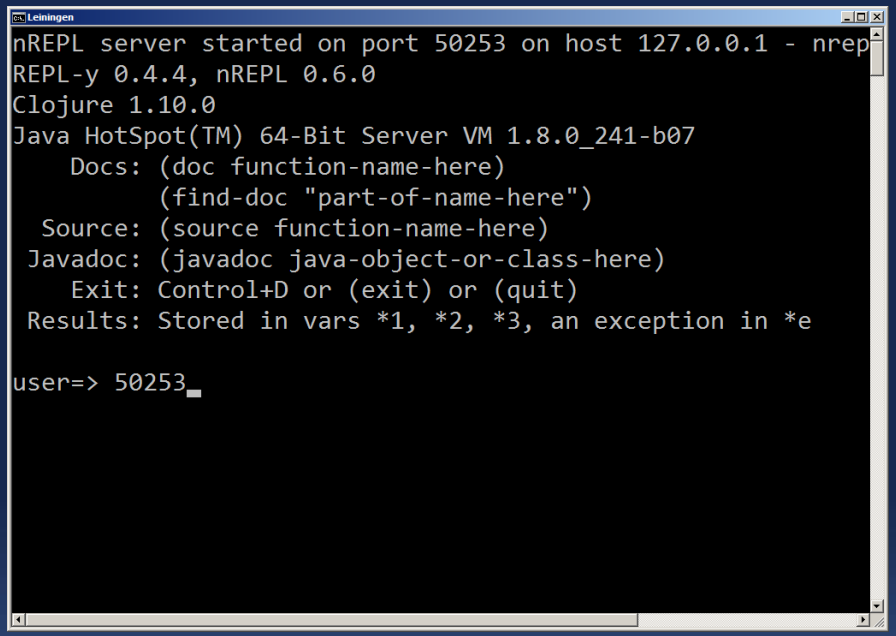
Ativar Atom

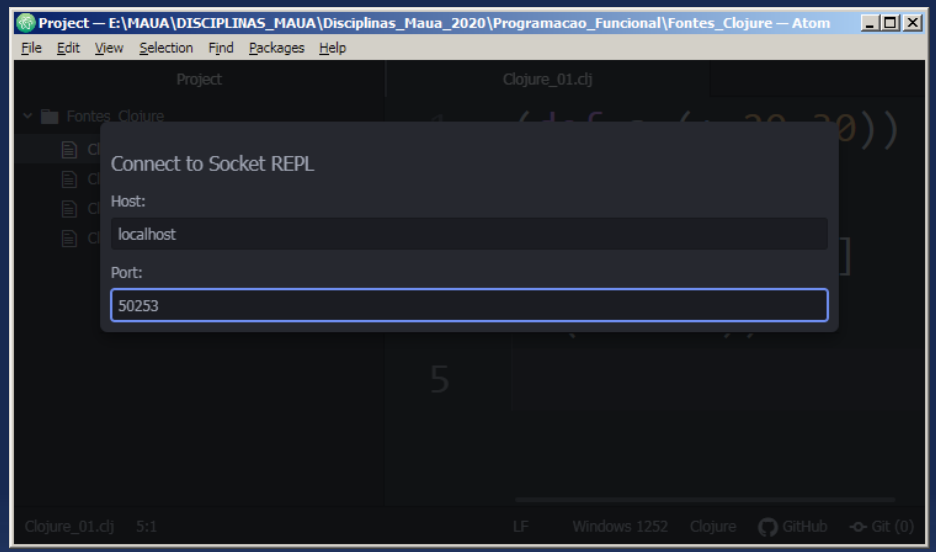
- ✓ Escrevendo código Clojure no Atom



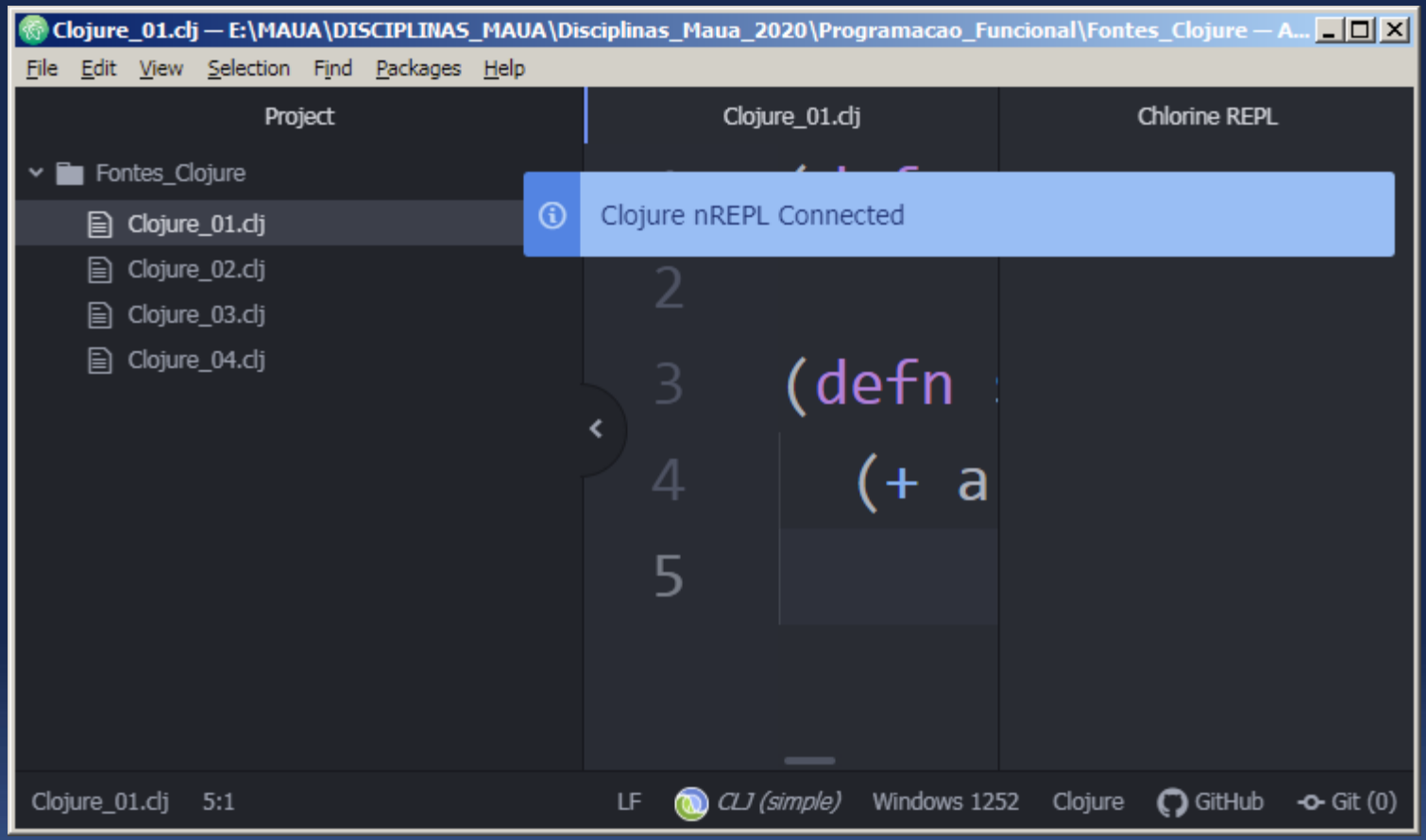
Ativar Atom

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



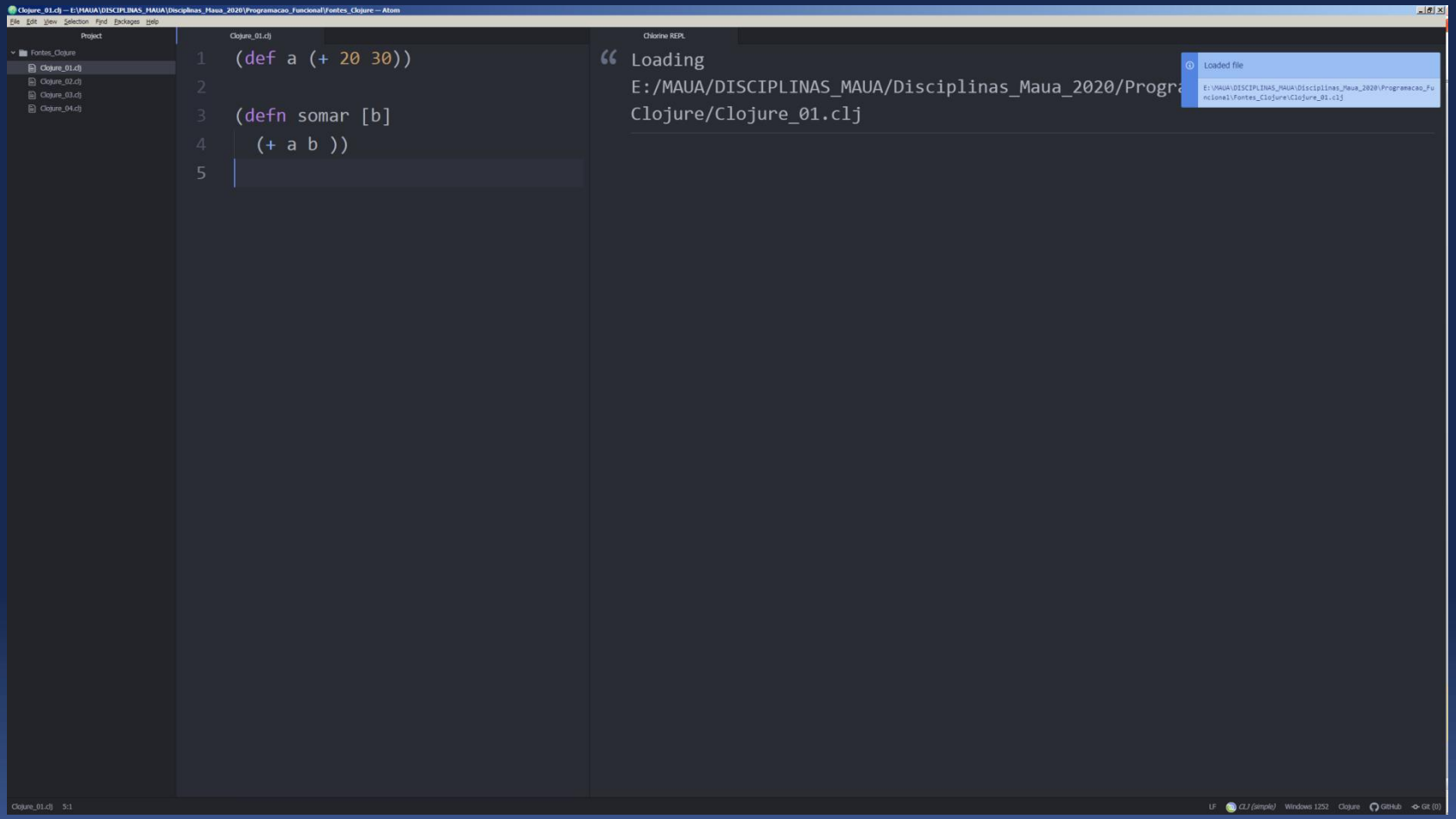


Clojure nREPL Connected



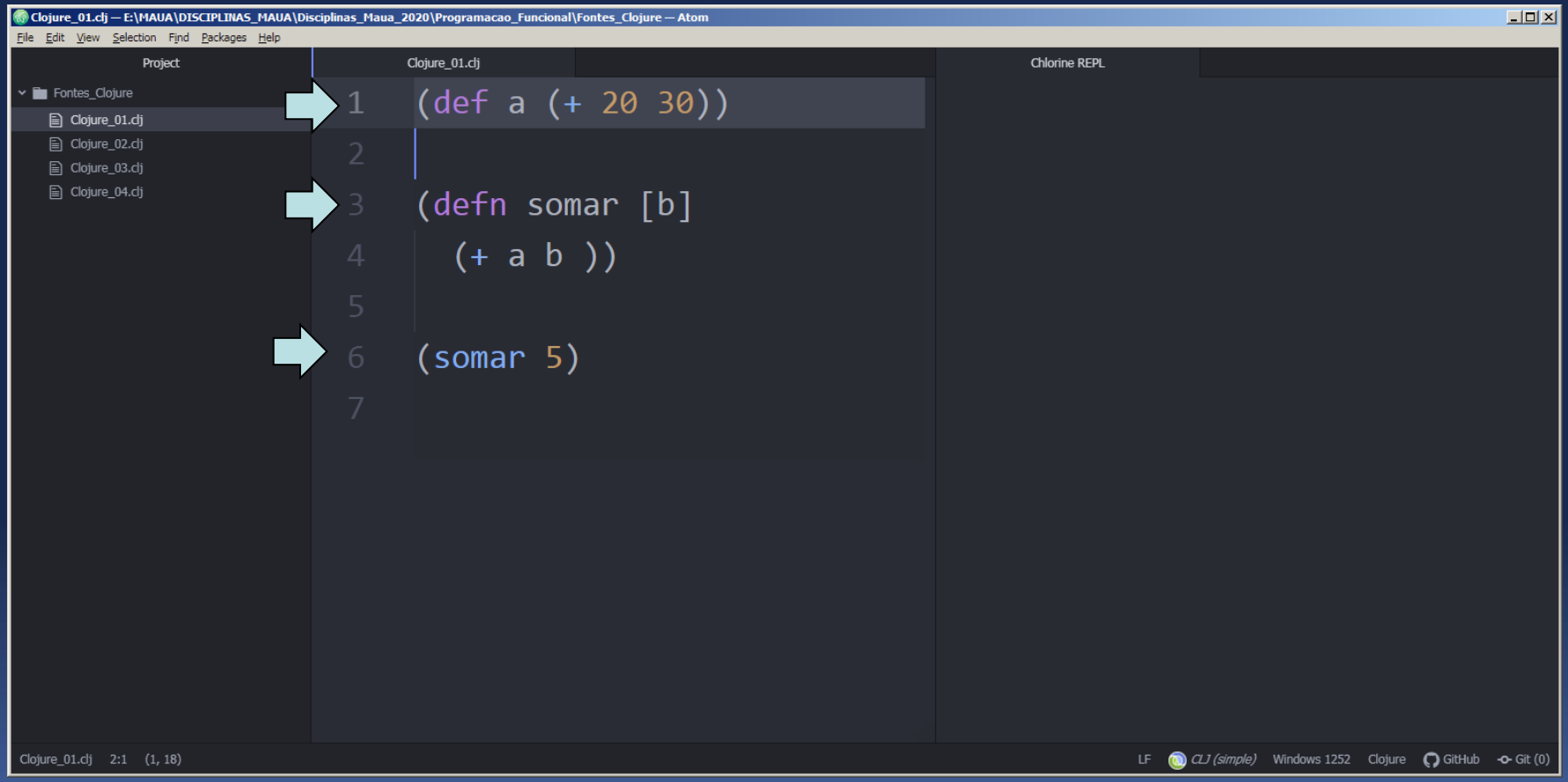
Carregando Código para nREPL

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



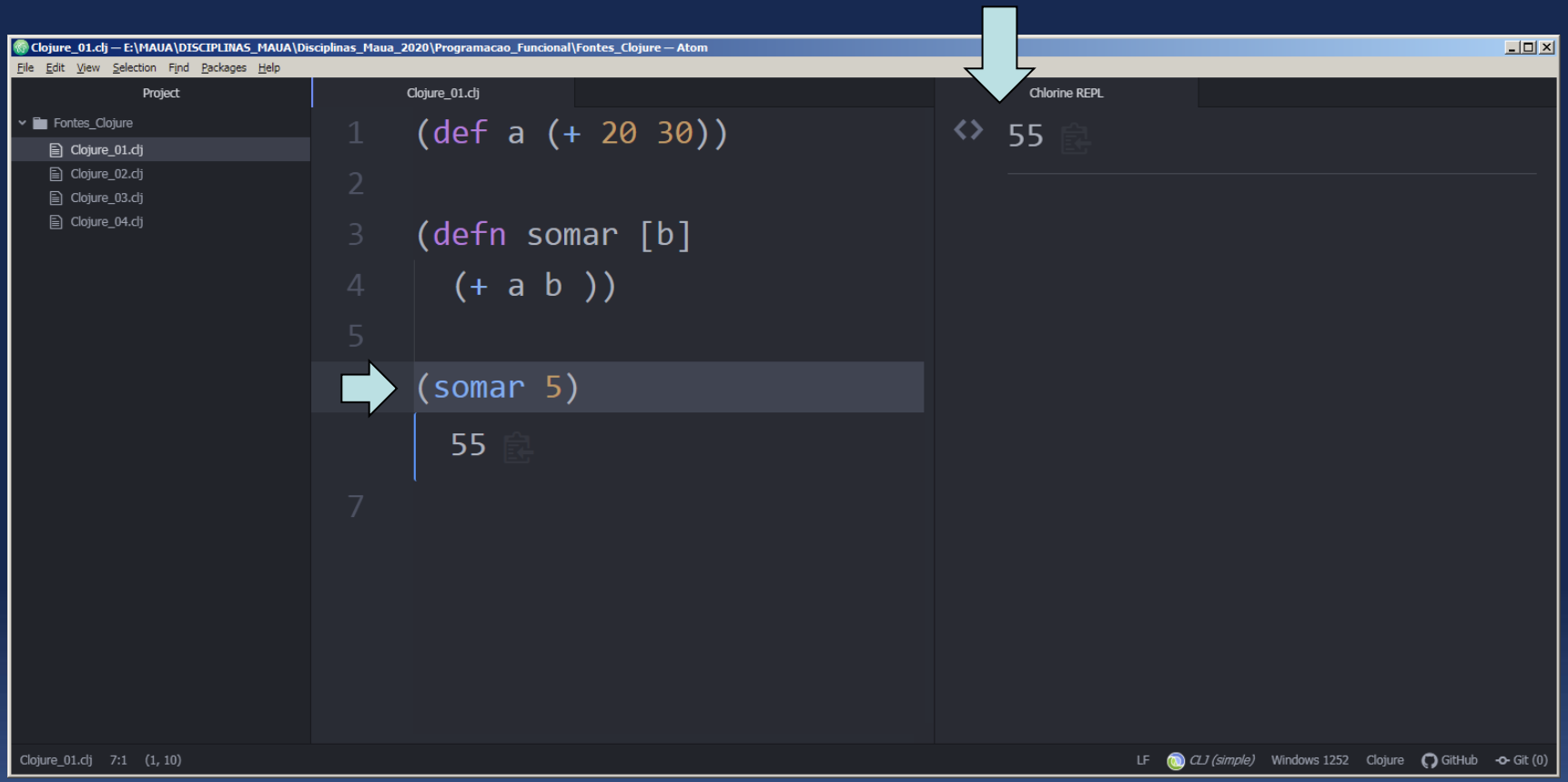
Evaluate "top-blocks"

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



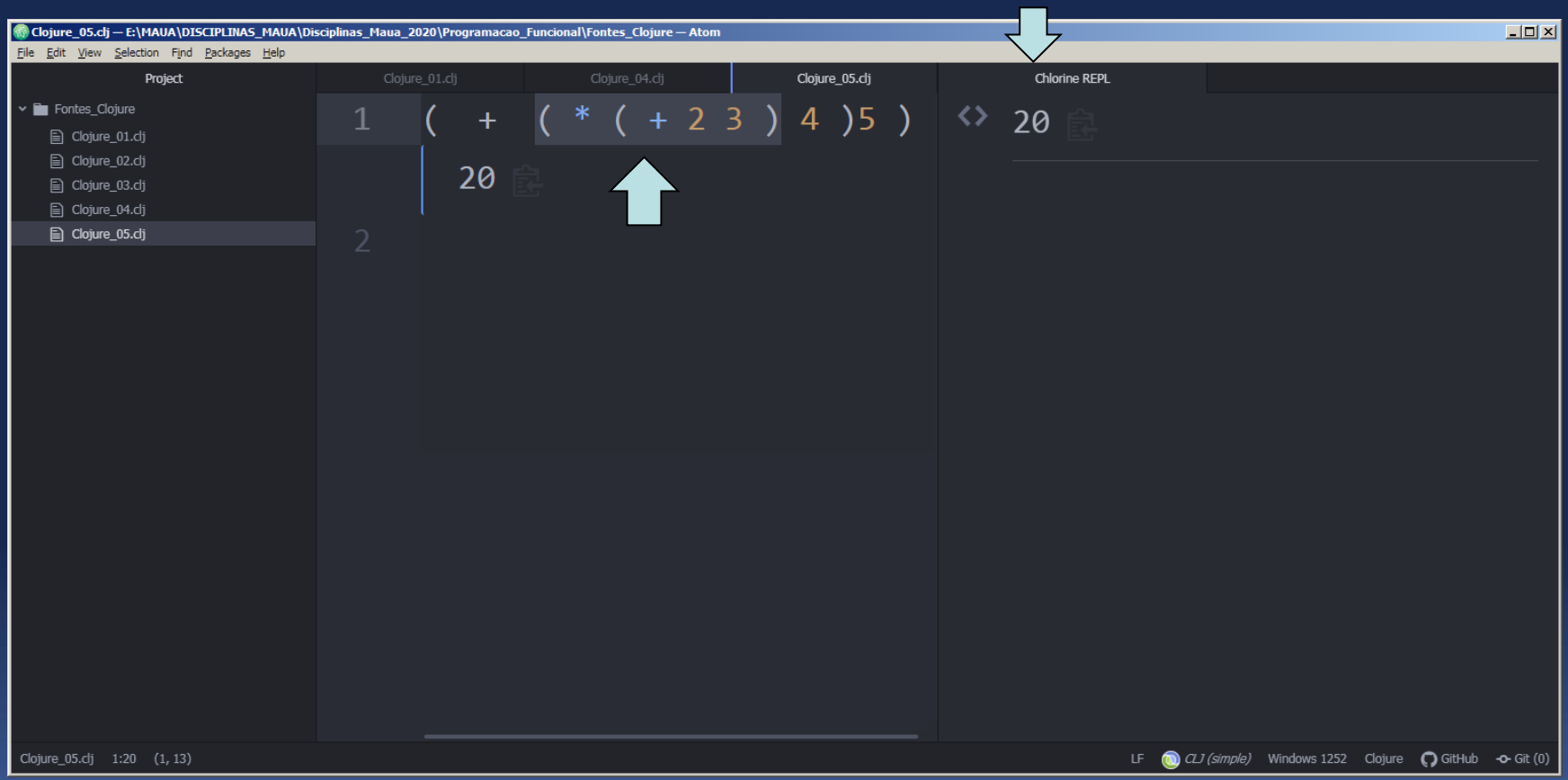
Evaluate "top-blocks"

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



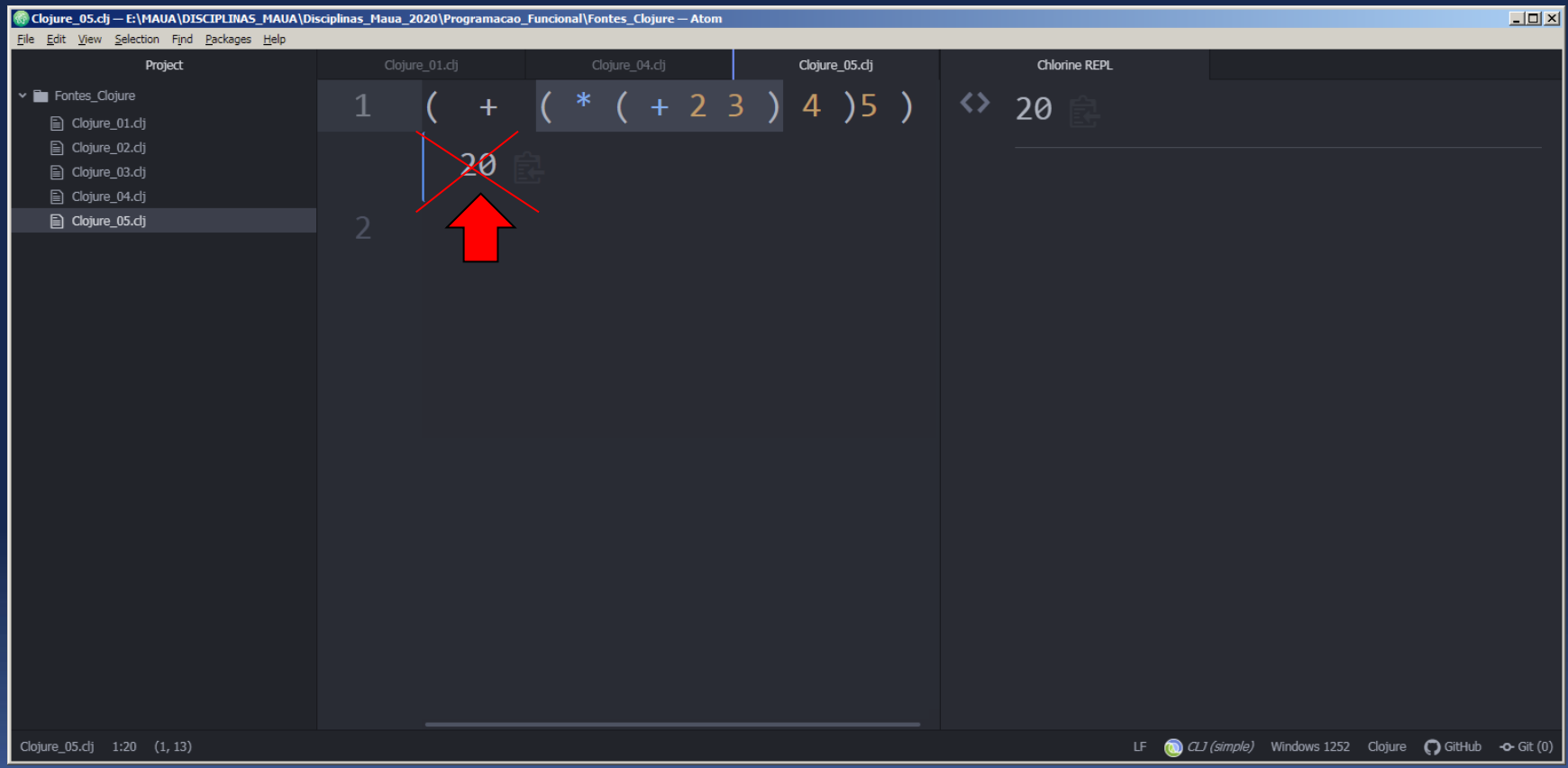
Evaluate "blocks"

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



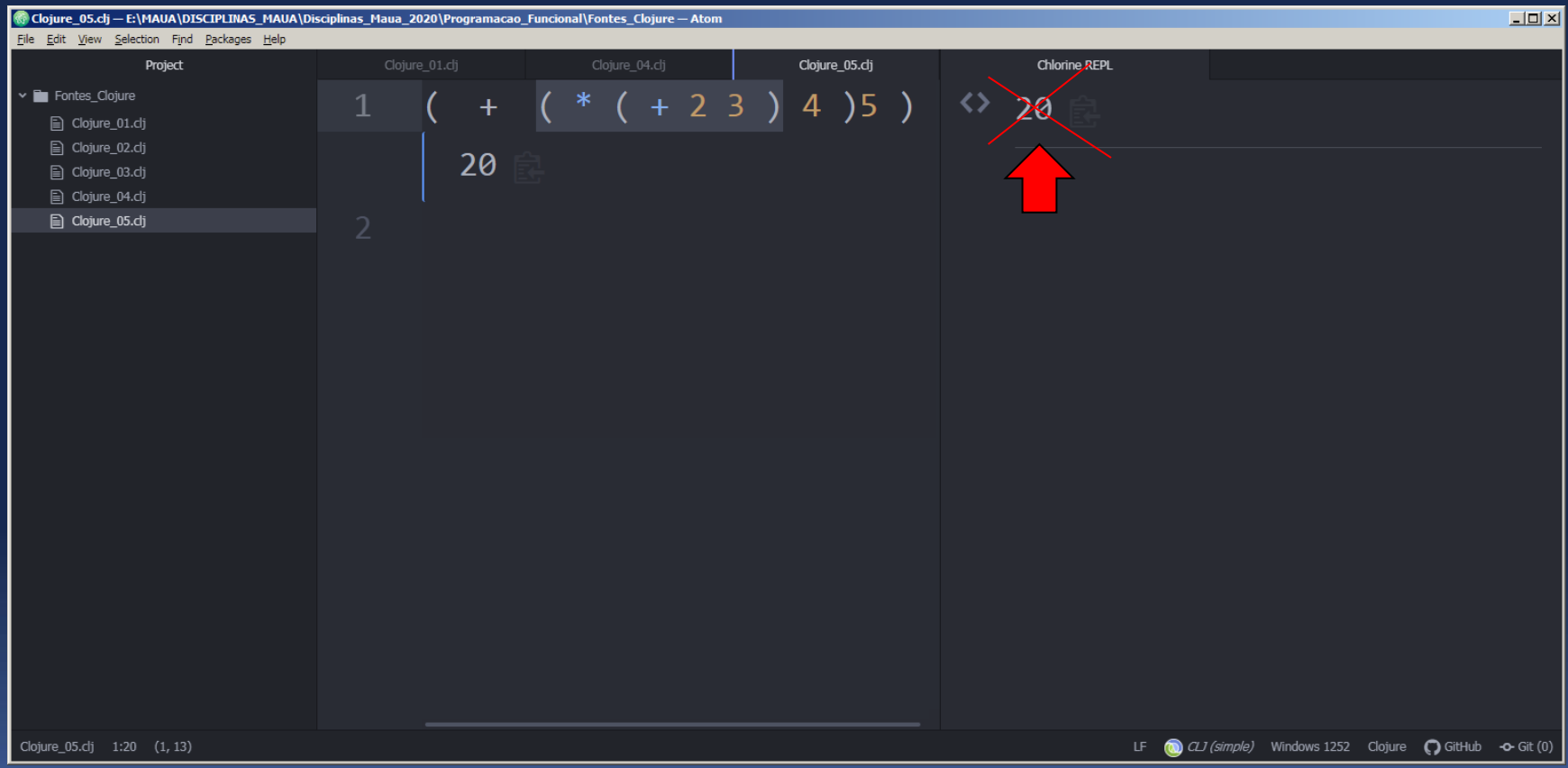
Clear Inline Results

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



Clear Console

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



Clear console

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