

## 1 Introduction and Learning Objectives

1. Ensure that you can start CLOJURE.
2. Use git operations to retrieve your MP.
3. Use midje to write tests.
4. Have fun playing with CLOJURE.

## 2 Getting Started

### 2.1 Get a command line environment

The first thing you need to do is to have a working development environment. There are a few ways you can accomplish that:

- Use the provided Vagrant box.
- Install `lein` on the EWS.
- Use your own computer running OS X or Linux
- Use of the editors that can connect to CLOJURE.

For these instructions, we are going to work with `lein`. If you have a different setup, feel free to ignore this.

If you are not familiar with Linux, please go through the Linux Command-line Guide, which will be posted on the course web site.

### 2.2 Get CLOJURE

Once you have a command line environment running, you need to be sure that CLOJURE is installed. If you are unsure if you have it or not, type

```
lein repl
```

and you should see something like this:

```
vagrant% lein repl
nREPL server started on port 58594 on host 127.0.0.1 - nrepl://127.0.0.1:58594
REPL-y 0.3.5, nREPL 0.2.6
Clojure 1.6.0
OpenJDK 64-Bit Server VM 1.7.0_71-b14
  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 numbers may be different. Hit control-D to exit.

## 2.3 Clone your repository

The linux account guide contains a chapter on Git. You will want to look at that, or the tutorial that is on bitbucket. In your repository you will find the directory `initial`.

In the `initial` directory you will see the following folders:

- `handout` — this contains the file `handout.tex`, which is the  $\text{\LaTeX}$  source for the current document.
- `src/initial` — contains the file `core.clj`, which contains the running code for the lab.
- `test/initial` — contains the file `t_test.clj`, which contains the test cases for the lab.

## 3 Given Code

### 3.1 The initial Namespace

At the beginning of `core.clj`, you will see these lines:

```
1 (ns initial.core)
```

This creates a namespace for your code. We will talk about that more later, but for now you can ignore it.

You need to write four functions. Since this is just practice for getting everything working, we'll have some fun and write politically active arithmetic functions.

- `plus` — This just adds integers together. We wrote this one for you, actually, so it's even easier than it sounds. This is the apathetic citizen who can't be bothered to vote.
- `socialist-plus` — People with more numbers to add should pay their fair share to help out people with fewer numbers. The `socialist-plus` function adds one to sums with fewer than two numbers, and subtracts one for each number after the first 2. See the given code for examples.
- `capitalist-plus` — People with more numbers are better than people with fewer numbers, and mathematical policy should reflect that. After all, it takes numbers to make numbers. The `capitalist-plus` function subtracts one to sums with fewer than two numbers, and adds one for each number after the first 2. See the given code for examples.
- `communist-plus` — Everyone should be equal, no matter how many numbers they have. The state will give you the numbers you need. All sums come out to 10.
- `political-extremist-plus` — Political extremists don't do anything in moderation. They multiply instead of adding. You get to decide which political group would use this function. :-)

### 3.2 Running the Code

If you want to run the code from the command line using `lein`, you can do that as follows:

```
% lein repl
user=> (require 'initial.core)
nil
user=> (use 'initial.core)
nil
user=> (socialist-plus 10 10 10 10)
38
user=> (plus 10 20 30)
60
```

Note that the % and user=> strings are prompts. Do not try typing them in.

### 3.3 Test Cases

The file `test/initial/t_core.clj` contains testing code. It uses a testing framework called `midje`. You will not write any test cases for this MP, but in the future you will, so do take a look to get an idea how they are written.

To run the test cases, run `lein midje`.

## 4 Your Work

Your job now is to write the functions and pass the test cases. There will be an autograder that will start up in a few days. All MPs are due one week from when the grading script is started, rounded up to the next midnight.