Prerequisites

The Clojure build tool

- Leiningen = the Clojure package manager and project jumpstarter
- Java

Install Leiningen

- See: http://leiningen.org

Test it

```
lein self-install
lein

-> Leiningen is a tool for working with Clojure projects.
...
```

An IDE

Many choices:

- Emacs
- IntelliJ with LaClojure or Cursive plugin
- Eclipse
- Lighttable (young but probably soon amazing)
- Sublime
- VI

Most popular in the Clojure world: Emacs and IntelliJ with Cursive plugin.

Emacs is cool but hard to learn. Focus on one thing at a time: Clojure!!

For this workshop, take **Sublime** or the Cursive plugin for IntelliJ.

Sublime

- Install sublime
- And its package manager: https://sublime.wbond.net, https://sublime.wbond.net/docs/usage
- Install the REPL integration: http://sublimerepl.readthedocs.org/en/latest/
    - Ctrl + Shift + P: Install Package
    - Define the path to leiningen

Shortcuts:

- Ctrl+ F12 cs Start REPL
- Ctrl+, b Send the current “block” to REPL. Currently Clojure-only.
- Ctrl+, s Send the selection to REPL
- Ctrl+, f Send the current file to REPL
- Ctrl+, l Send the current line to REPL

Lab - Exploring the REPL

Start a REPL

```
lein repl
```

Clojure has brackets around the expression and the operator comes first:

5 + 2 in Clojure:

```
(+ 5 2)
```

Try a couple of math operations in the REPL.

- println is a function to print
- str concatenates strings
- Clojure strings are like Java: “Hey dude!”

Try more stuff

Lab - Test driven development

Clojure has its own test library *clojure.test*, but there are other libraries like

- https://github.com/slagyr/speclj
- https://github.com/marick/Midje

In this lab we will use *Midje*. Steps:

- Create a new project
- Execute tests in REPL
- Do a coding DOJO

Prepare the project

Create an empty project

```
lein new stringcalc
```

This creates an empty project.

We have to add the dependencies to the midje testing library. Modify the *project.clj* in the root of the project.

```
(defproject stringcalc "0.1.0-SNAPSHOT"
  :dependencies [[org.clojure/clojure "1.6.0"]]
  :profiles {
    :dev {
        :dependencies [[midje "1.6.3"]]
        :plugins [[lein-midje "3.1.3"]]}})
```

Start a REPL

```
lein repl
```

Import the midje.repl lib

```
(require 'midje.repl)
(midje.repl/autotest)
```

Midje starts executing the tests. You should see something like

```
FAIL in (a-test) (core_test.clj:7)
FIXME, I fail.
...
```

Remove the dummy files.

```
rm stringcalc/src/stringcalc/core.clj
rm stringcalc/test/stringcalc/core_test_.clj
```

Coding Dojo

For a Dojo, we can keep tests and code in a single file.

Create a new file *string_calc_test.clj* in *test/foo*

```
(ns stringcalc.stringcalc-test
 (:require [midje.sweet :refer :all] ))
```

```
(defn calc[input] )

(fact empty-string
    (calc "") => 0)
```

The moment you safe the file, you should see the test failure in the REPL.

```
FAIL in (empty-string) (stringcalc_test.clj:12)
empty string returns 0
expected: 0
  actual: nil
    diff: - 0
```

Follow the Kata at: http://osherove.com/tdd-kata-1/

Some pointers

If you want to use the clojure tests instead of midje

- Run tests on file change: https://github.com/jakemcc/lein-test-refresh
- More readable output: https://github.com/pjstadig/humane-test-output

Lab - Rest services

Compojure is a small routing library. It uses ring, which is the Clojure abstraction for HTTP requests.

Start a new template project:

```
lein new compojure hello-world
```

Start a local server

```
lein ring server
```

If not already open in your browser, go to http://localhost:3000/

The routes are defined in the *handlers.clj* in *src/hello-world*

As we build a JSON REST api and not a website, please change the ring handler in *handlers.clj* to:

```
(def app
  (wrap-defaults app-routes api-defaults))
```

Destructure request parameters

```
(GET "/myage" {{:keys [name age]} :params}
    (str name " age is " age))

; Test: http://localhost:3000/myage?name=abc&age=5
```

Destructure path parameter

```
(GET "/myname/:name" [name]
    (str "My name is " name))

; Test: http://localhost:3000/myage?name=abc&age=5
```

JSON response

Add the **dependency [ring/ring-json "0.3.1"]** to the *project.clj* and restart *lein ring server*

Change your handler.

```
(ns hello-world.handler
  (:require [compojure.core :refer :all]
            [compojure.handler :as handler]
            [compojure.route :as route]
            [ring.util.response :refer [response]]
        [ring.middleware.json :refer [wrap-json-response wrap-json-body]]
            ))

(defroutes app-routes
  (GET "/" [] (response {:message "Hello World"}))

  (def app
    (->
        (wrap-defaults app-routes api-defaults)
        (wrap-json-body {:keywords? true})
        (wrap-json-response)))
```

Exercises:

- Add a POST and a DELETE service

Hint: JSON params are stored in request under {:body {…}}

Pointers

- https://github.com/weavejester/compojure
- Very cool stuff: JSON schemas to validate input and REST API documentation for free
    - https://github.com/metosin/compojure-api
    - lein new compojure-api my-api

Lab - Packaging for Jetty / Tomcat

The *lein ring* plugin has tasks to build war files with all required libraries.

```
lein ring uberwar
```

You can find the WAR in *target*. Copy it into the Jenkins or Tomcat deploy directory.

Lab - Database

There are different libs to abstract from SQL in Clojure. We will have a look at Korma

http://sqlkorma.com

Add the following dependencies to the *project.clj* to access PostgreSQL

```
[korma "0.3.2"]
[org.postgresql/postgresql "9.3-1101-jdbc4"]
```

If you want to use MySQL use its driver accordingly: [mysql/mysql-connector-java “5.1.32”]

In a bash, create PostgreSQL database and tables:

```
createdb kitestore
psql --username=postgres kitestore -c "create table kites(name text, type text, size int);"
```

In a REPL:

```
(require 'korma.db)
(korma.db/defdb prod (korma.db/postgres {:db "kitestore"
                      :user "postgres"
                      :password "p"}))


(require '[korma.core :refer :all])
(defentity kites)
```

```
(select kites
  (fields :name :type)
  (where {:size [> 12]}))

(insert kites (values {:name "Core" :type "Delta" :size 14}))
```

Korma provides a very good documentation: http://sqlkorma.com/docs

If you prefer Mongo DB have a look at congomongo.

Books and resources

- The Joy of Clojure, 2nd Edition (2014)
- Programming Clojure, 2nd edition (2012)
- http://clojurekoans.com (online exercises)
- https://www.4clojure.com (local exercises)
- http://clojars.org/ - the community repository of most projects
- http://tryclj.com - online REPL