CS 596 Functional Programming and Design
Fall Semester, 2014
Doc 6 Some tools
Sep 16, 2014

Some Tools

Leiningen

Project automation & configuration

http://leiningen.org/

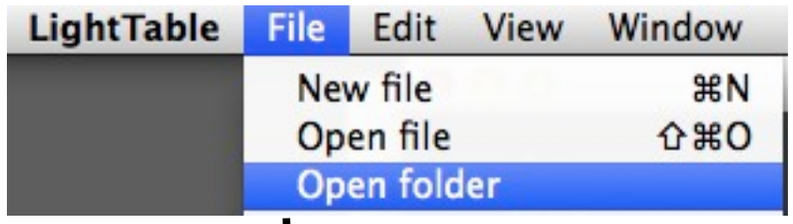
Follow the install instructions

Generating a Project

At command line

```
lein new app lectureexample
```

Open the Project in Light Table




```
lectureexample/
  doc/
  resources/
  src/
    lectureexample/
      core.clj
  test/
    lectureexample/
      core_test.clj
  LICENSE
  project.clj
  README.md
```

core.clj

core.clj

```clojure
(ns lectureexample.core
  (:gen-class))

(defn -main
  "I don't do a whole lot ... yet."
  [& args]
  (println "Hello, World!"))

```

namespaces

each file in src is a namespace

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

Follow Java's conventions

lectureexample.core
- Needs to be in file core.clj
- Inside directory lectureexample

If namespace name has a "-" in it
- Filename uses "_" instead of "-"

New Namespace

lectureexample/src/lectureexample/add_methods.clj

```
(ns lectureexample.add-methods
  (:gen-class))

(defn add-5
  [x]
  (+ x 5))

(defn add-10
  [x]
  (+ x 10))

(defn add-20
  [x]
  (+ x 20))

(defn- add-100
  [x]
  (+ x 100))
```

:require :refer :all

Can access all public methods in other namespace with name of function

```
(ns lectureexample.core
  (:require [lectureexample.add-methods :refer :all])
  (:gen-class))

(add-5 8)      ;;Works fine
(add-10 8)     ;;Works fine
(add-20 8)     ;;Works fine
(add-100 8)  ;;Compile error - private method
```

:require :as

Can access all public methods in other namespace
Name of the function has qualifier

```
(ns lectureexample.core
  (:require [lectureexample.add-methods :as adder])
  (:gen-class))

(adder/add-5 8)        ;;Works fine
(adder/add-10 8)       ;;Works fine
(adder/add-20 8)       ;;Works fine
(adder/add-100 8)      ;;Compile Error
(add-5 10)             ;;Compile Error
```

:require :refer

Can access all public methods listed after :refer

```
(ns lectureexample.core
  (:require [lectureexample.add-methods :refer [add-5 add-10]])
  (:gen-class))
```

```
(add-5 8)        ;;Works fine
(add-10 8)       ;;Works fine
(add-20 8)       ;;Compile Error
(add-100 8)      ;;Compile Error
```

Adding more than one namespace

```
(ns rdr-interface.gui
  (:require [rdr-interface.data :as data]
            [rdr-interface.web :as web]
            [rdr-interface.html :as html]
            [rdr-interface.gui-student :as sgui]
            [rdr-interface.gui-open-group :refer [show-groups]]
            [rdr-interface.gui-input-list :refer [show-input-list-window]]
            [seesaw.dev :as sd]
            [seesaw.table :as table]
            [seesaw.mig :as mig])
            [seesaw.core :refer :all]
            [seesaw.swingx :refer :all])
(:gen-class))
```

REPL Syntax is Different

(use 'clojure.tools.trace)

Not recommended for projects

clojure.tools.trace

https://github.com/clojure/tools.trace

Defines tracing macros/fns to help you see what your code is doing.

Shows input and return values

Several ways to to turn on tracing

Replace defn with deftrace

trace-vars

trace-ns

Using deftrace

```
(ns lectureexample.core
  (:require [clojure.tools.trace :refer :all]))

(deftrace add-10
  [x]
  (+ x 10))

(deftrace add-ten
  "adds 10 to each element in collection"
  [collection result]
  (if (seq collection)
    (let [element (first collection)
          plus-10 (add-10 element)]
      (add-ten (rest collection) (conj result plus-10)))
    result))

(add-ten [1 2 3] [])
```

The Trace

```
core.clj: TRACE t7011: (add-ten [1 2 3] [])
core.clj: TRACE t7012: | (add-10 1)
core.clj: TRACE t7012: | => 11
core.clj: TRACE t7013: | (add-ten (2 3) [11])
core.clj: TRACE t7014: | | (add-10 2
core.clj: TRACE t7014: | | => 12
core.clj: TRACE t7015: | | (add-ten (3) [11 12])
core.clj: TRACE t7016: | | | (add-10 3)
core.clj: TRACE t7016: | | | => 13
core.clj: TRACE t7017: | | | (add-ten () [11 12 13])
core.clj: TRACE t7017: | | | => [11 12 13]
core.clj: TRACE t7015: | | => [11 12 13]
core.clj: TRACE t7013: | => [11 12 13]
core.clj: TRACE t7011: => [11 12 13]
```

Using trace-vars

```
(ns lectureexample.core
  (:require [clojure.tools.trace :refer :all]))

(defn add-10
  [x]
  (+ x 10))

(defn add-ten
  "adds 10 to each element in collection"
  [collection result]
  (if (seq collection)
    (let [element (first collection)
          plus-10 (add-10 element)]
      (add-ten (rest collection) (conj result plus-10)))
    result))

(trace-vars add-ten)
(add-ten [1 2 3] [])
```

Each time you change add-ten you need to evaluate (trace-vars add-ten)

Use (untrace-vars add-ten) to turn off tracing

Using trace-ns

```
(trace-ns 'lectureexample.core)

(untrace-ns 'lectureexample.core)
```

Configuring Project

Add trace to dependencies in project.clj file

Add :require in source code

project.clj

```
(defproject lectureexample "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.6.0"]
                 [org.clojure/tools.trace "0.7.8"]]
  :main ^:skip-aot lectureexample.core
  :target-path "target/%s"
  :profiles {:uberjar {:aot :all}})
```

Source code

```
(ns lectureexample.core
  (:require [clojure.tools.trace :refer :all])
  (:gen-class))


(defn add-10
  [x]
  (+ x 10))
```

```
(defn add-ten
  "adds 10 to each element in collection"
  [collection result]
  (if (seq collection)
    (let [element (first collection)
          plus-10 (add-10 element)]
      (add-ten (rest collection) (conj result plus-10)))
    result))
```

Unit Tests

Leiningen Projects Include Testing

Sets up requirements for tests

Clojure has testing framework

Similar to JUnit

```
lectureexample/
  doc/
  resources/
  src/
    lectureexample/
      core.clj
  test/
    lectureexample/
      core_test.clj
  LICENSE
  project.clj
  README.md
```

Generated test file: core_test.clj

```
(ns lectureexample.core-test
  (:require [clojure.test :refer :all]
            [lectureexample.core :refer :all]))

(deftest test-add-ten
  (testing "FIXME, I fail."
    (is (= 0 1))))
```

deftest - defines the test

testing - optional, label for the output

is - testing method

Some Tests

```
(deftest test-add-ten
  (is (= (add-ten [1 2 3] [] )  [11 12 13]))
  (is (= (add-ten [1] [] )      [11]))
  (is (= (add-ten [] [] )       [] ))
  (is (thrown? clojure.lang.ArityException (add-ten [1 2]))))
```

Inside of **is** place some statement about code that returns true/false

Light Table & Tests

Light Table does not run your tests for you :(

Will see two different ways to run the tests

Semi-Manual

```
(ns lectureexample.core-test
  (:require [clojure.test :refer :all]
            [lectureexample.core :refer :all]))

(defn reload-tests
      [ ]
      (use 'lectureexample.core :reload-all)
      (use 'lectureexample.core-test :reload-all)
      (run-tests 'lectureexample.core-test))

(reload-tests)

(deftest test-add-ten
  (is (= (add-ten [1 2 3] [] )  [11 12 13]))
  (is (= (add-ten [1] [] )      [11]))
  (is (= (add-ten [] [] )       [] ))
  (is (= 1 2))
  (is (thrown? clojure.lang.ArityException (add-ten [1 2]))))
```

:reload-all
reload defintions of your code

run-tests - runs the test

How to Run test Automatically

lein-test-refresh

- Leiningen plug-in
- Runs tests when your source code files change
- Need to run Leiningen command
- Need to configure project

project.clj

```
(defproject lectureexample "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.6.0"]
                 [org.clojure/tools.trace "0.7.8"]]
  :main ^:skip-aot lectureexample.core
  :target-path "target/%s"
  :profiles {:uberjar {:aot :all}}
  :plugins [[com.jakemccrary/lein-test-refresh "0.5.1"]])
```

Starting lein-test-refresh

In terminal/command line

Go to project directory

```
cd lectureexample/
```

Run lein test-refresh

```
lein test-refresh
```

Every time you save a source file test-refresh reload code & runs test

Sample Output

```
Al pro 18->lein test-refresh
**********************************************
*************** Running tests ***************
:reloading (lectureexample.core lectureexample.core-test lectureexample.add-methods)

Testing lectureexample.core-test

Ran 0 tests containing 0 assertions.
0 failures, 0 errors.

Testing lectureexample.core-test

FAIL in (test-add-ten) (core_test.clj:18)
expected: (= 1 2)
  actual: (not (= 1 2))

Ran 1 tests containing 5 assertions.
1 failures, 0 errors.

Failed 1 of 5 assertions
Finished at 19:42:40.035 (run time: 0.263s)
```

are - Shortcut for multiple is

```
(deftest test-add-ten
  (is (= (add-ten [1 2 3] [] )  [11 12 13]))
  (is (= (add-ten [1] [] )      [11]))
  (is (= (add-ten [] [] )       [] ))
  (is (= (add-ten nil [])       []))
```

```
(deftest test-add-ten
  (are [list result] (= (add-ten list []) result)
       [1 2 3]  [11 12 13]
       [1]      [11]
       []       []
       nil      [] ))
```

What to Test

Everything that could possibly break

Test values

- Inside valid range
- Outside valid range
- On the boundary between valid/invalid

GUIs are very hard to test

- Keep GUI layer very thin
- Unit test program behind the GUI, not the GUI

Common Things Programs Handle Incorrectly

Adapted with permission from “A Short Catalog of Test Ideas” by Brian Marick,
http://www.testing.com/writings.html

Strings

Empty String

Collections

Empty Collection
Collection with one element
Collection with duplicate elements
Collections with maximum possible size

Numbers

Zero
The smallest number
Just below the smallest number
The largest number
Just above the largest number

Test for sdsu-nth

```
(deftest sdsu-nth-test
  (testing "sdsu-nth"
    (are [answer list n] (= answer (sdsu-nth list n))
         nil [] 0
         nil [] 1
         1 [1 2 3] 0
         2 [1 2 3] 1
         3 [1 2 3] 2
         nil [1 2 3] 3
         nil [1 2 3] -2
         )))
```

Spyscope - Clojure Magic

Spyscope

Useful debugging tool

Uses reader macros - Lisp magic

Does not work in Light Table, but does work with lein-test-refresh

Three reader macros

#spy/p
#spy/d
#spy/t

Example - The Tests

```
(ns lectureexample.core-test
  (:require [clojure.test :refer :all]
            [lectureexample.core :refer :all]))


(deftest test-add-ten
  (is (= (add-ten [1 2 3] [] )  [11 12 13])))
```

project.clj file

```
(defproject lectureexample "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.6.0"]
                 [org.clojure/tools.trace "0.7.8"]
                 [spyscope "0.1.4"]]
  :main ^:skip-aot lectureexample.core
  :target-path "target/%s"
  :profiles {:uberjar {:aot :all}}
  :plugins [[com.jakemccrary/lein-test-refresh "0.5.1"]])
```

In Light Table - core.clj

```
(ns lectureexample.core
  (:require [clojure.tools.trace :refer :all]
            [spyscope.core :refer :all])
  (:gen-class))

(defn add-10
  [x]
  (+ x 10))
```

But do not evaluate in Light Table

```
(defn add-ten
  "adds 10 to each element in collection"
  [collection result]
  (if #spy/d (seq collection)
    (let [element (first collection)
          plus-10 (add-10 element)]
      (add-ten (rest collection) #spy/d (conj result plus-10)))
    result))
```

Output using lein-test-refresh

```
**********************************************
*************** Running tests ***************
:reloading (lectureexample.core lectureexample.core-test)

Testing lectureexample.core-test
lectureexample.core$add_ten.invoke(core.clj:14) (seq collection) => (1 2 3)
lectureexample.core$add_ten.invoke(core.clj:17) (conj result plus-10) => [11]
lectureexample.core$add_ten.invoke(core.clj:14) (seq collection) => (2 3)
lectureexample.core$add_ten.invoke(core.clj:17) (conj result plus-10) => [11 12]
lectureexample.core$add_ten.invoke(core.clj:14) (seq collection) => (3)
lectureexample.core$add_ten.invoke(core.clj:17) (conj result plus-10) => [11 12 13]
lectureexample.core$add_ten.invoke(core.clj:14) (seq collection) => nil

Ran 1 tests containing 1 assertions.
0 failures, 0 errors.

Passed all tests
Finished at 20:00:28.457 (run time: 0.064s)
```

With tools.trace

```
(ns lectureexample.core
  (:require [clojure.tools.trace :as trace]
            [spyscope.core :refer :all])
  (:gen-class))

(defn add-10
  [x]
  (+ x 10))
```

```
(trace/deftrace add-ten
  "adds 10 to each element in collection"
  [collection result]
  (if #spy/d (seq collection)
    (let [element (first collection)
          plus-10 (add-10 element)]
      (add-ten (rest collection) #spy/d (conj result plus-10)))
    result))
```

Output

```
Testing lectureexample.core-test
TRACE t1939: (add-ten [1 2 3] [])
lectureexample.core$eval1896$f__1430__auto____1897.invoke(core.clj:15) (seq collection) => (1 2 3)
lectureexample.core$eval1896$f__1430__auto____1897.invoke(core.clj:18) (conj result plus-10) => [11]
TRACE t1940: | (add-ten (2 3) [11])
lectureexample.core$eval1896$f__1430__auto____1897.invoke(core.clj:15) (seq collection) => (2 3)
lectureexample.core$eval1896$f__1430__auto____1897.invoke(core.clj:18) (conj result plus-10) => [11 12]
TRACE t1941: | | (add-ten (3) [11 12])
lectureexample.core$eval1896$f__1430__auto____1897.invoke(core.clj:15) (seq collection) => (3)
lectureexample.core$eval1896$f__1430__auto____1897.invoke(core.clj:18) (conj result plus-10) => [11 12 13]
TRACE t1942: | | | (add-ten () [11 12 13])
lectureexample.core$eval1896$f__1430__auto____1897.invoke(core.clj:15) (seq collection) => nil
TRACE t1942: | | | => [11 12 13]
TRACE t1941: | | => [11 12 13]
TRACE t1940: | => [11 12 13]
TRACE t1939: => [11 12 13]
```

Workflows

Learning, Exploring

How does this work?

What does this function really do?

How does this library work?

Evaluate in Light Table

Writing Code/Applications

Use Light Table as editor

Write Unit tests

Use lein-test-refresh

Use trace & spy when trouble

Find more errors when focus on unit tests