Introduction to Clojure

Eric Normand
October 11, 2012

Short History of Lisp

“The greatest single programming language ever designed.”

– Alan Kay, 2003

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

– John McCarthy, 1960

Characteristics of Lisp

- Minimal syntax
  - Parentheses
  - Prefix (Polish) notation
- Linked Lists (**LIS**t **P**rocessing)
- Garbage Collection
- Dynamically compiled
- Functional
- Structured
- Homoiconic
- Many more

Two Major Dialects

- Scheme
  - Academic
  - Minimal
  - 50 page specification
- Common Lisp
  - Commercial
  - Standardization committee
  - ~1000 page specification

Enter Clojure

- Rich Hickey, creator
- Released 2007
- Compiles to JVM + ~~CLR~~
- Clojure is a Modern Lisp
  - Fresh start
  - Persistent data structures
  - Literal syntax
  - Concurrency primitives
  - Inspiration from other languages
  - Practical for modern software

Enter Clojure

- Rich Hickey, creator
- Released 2007
- Compiles to JVM + ~~CLR~~
- Clojure is a Modern Lisp
  - Fresh start
  - Persistent data structures
  - Literal syntax
  - Concurrency primitives
  - Inspiration from other languages
  - Pragmatic

Time for Code

Follow along

tryclj.com

Macros

- Macros are functions from code to code
- Macros are called at compile time instead of run time
- Macro calls look like function calls
- Useful for extending syntax
  - conditionals
  - loops
  - with-open

Concurrency

- Software Transactional Memory
- Primitives
  - var
  - atom
  - ref
  - agent
- watchable
  - function called when state changes

STM vs. Locks

Locks	STM
Hard to reason about	Easy to reason about
Deadlock	No deadlock
Blocking reads	Non-blocking reads
Mutual exclusion	Retrying transactions
Low level	High level

ref

- coordination between state
- consistent reads in transactions
  - non-blocking
- atomic transactions
  - retry if something else has modified
  - can be nested
- no io inside of transactions, please
- modification
  - alter
  - ref-set
  - commute

atom

- uncoordinated, synchronous state
- thread-safe (no race conditions)
- modification
  - swap!

agent

- uncoordinated, asynchronous state
  - modification runs in another thread
- thread-safe
- modification
  - send
  - send-off

Where to next?

- 4clojure: http://4clojure.com
- Clojure Koans: https://github.com/functional-koans/clojure-koans
- leiningen: https://github.com/technomancy/leiningen
- Concurrency talk: http://blip.tv/clojure/clojure-concurrency-819147