CS 596 Functional Programming and Design
Fall Semester, 2014
Doc 2 Clojure Introduction
Aug 28, 2014

Copyright ©, All rights reserved. 2014 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/openpub/) license defines the copyright on this document.
Simplicity Matters
Simplicity Matters

Rich Hickey

What does he think of OO programming?

What is the Elephant?

What is the difference between Simple and Easy?

Good design involves what?

Benefits verses Tradeoffs

Examples of complecting order with lists
The Future Programming Manifesto

http://alarmingdevelopment.org/?p=893

Inessential complexity is the root of all evil

Complexity is cumulative cognitive load

Simplicity first; performance last
Simplicity is Hard work
What is Functional Programming
## Elements of Functional Programming

<table>
<thead>
<tr>
<th>Pure Functions</th>
<th>Currying</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Class Functions</td>
<td>Memoization</td>
</tr>
<tr>
<td>Higher-Order Functions</td>
<td>Destructuring</td>
</tr>
<tr>
<td>Immutability</td>
<td>Collection Pipelines</td>
</tr>
<tr>
<td>Lazy Evaluation</td>
<td>List Compressions</td>
</tr>
<tr>
<td>Recursion</td>
<td></td>
</tr>
</tbody>
</table>
Lazy Evaluation

Operations & functions evaluated
  When used
  Not when called

Why important

Simplifies logic
Recursion

function factorial(n)
    if n = 1 return 1
    return n * factorial(n-1)

Tail recursion/Tail Call Optimization

When last statement is just the recursion
Compiler can convert recursion into loop

Why important

Powerful tool
Currying

function add(int x, int y) {
    return x + y;
}

addTen = add(10);

addTen(3) //returns 13
Memoization

Cache value of functions

Why important

memoize(factorial)

Performance

factorial(1000) // 1000 recursive calls
factorial(1001) // 1 recursive call
Collection Pipelines

String[] words = {"a", "ab", "abc", "abcd", "bat"};
List<String> wordList = Arrays.asList(words);
List<String> longWords;
longWords = wordList.stream()
    .filter( s -> s.length() > 2)
    .filter( s -> s.charAt(0) == 'a')
    .map( s -> s.toUpperCase())
    .collect(Collectors.toList());
Clojure
Clojure

Developed by Rich Hickey

Started 2007

Variant of Lisp

Functional programming language

Dynamic typing

Interactive development - REPL

Tight Java Integration

Active development community
Variants

Clojure  
Java

ClojureScript  
JavaScript

ClojureCLR  
.NET

Base language the same

Few changes due differences between Java/JavaScript/.NET
Development Environment

Command Line

Eclipse
Counterclockwise plugin
https://code.google.com/p/counterclockwise/

IntelliJ
Cursive plugin
https://cursiveclojure.com

Emacs
CIDER

Vim
Fireplace

Light Table
Clojure/Web IDE
https://nightcode.info

Lein�이
Night Code
Light Table

http://www.lighttable.com

Recommended IDE for the course
Lots of Irritating Superfluous Parenthesis-LISP

Actually not more that Java's

But only () and they build up
  (+ 5 (- 2 (/ 4 (* 2 (inc (read-string "123"))))))

Use editor that is parenthesis aware

Useful forms
  let
  ->
Resources

Clojure Home Page

http://clojure.org

Google Clojure Groups

https://groups.google.com/forum/#!forum/clojure

Clojure Cookbook

Safari Books On-line
http://proquest.safaribooksonline.com.libproxy.sdsu.edu/
Clojure

Clojure 1.3-1.6 Cheat Sheet (v13)

Download PDF version, Download other versions with tooltips

Documentation

clojure.repl/

```
doc find-doc apropos source pst javadoc (foo.bar/ is namespace for later syms)
```

Primitives

Numbers

<table>
<thead>
<tr>
<th>Literals</th>
<th>Long: 7, hex 0xff, oct 017, base 2 2r1011, base 36 36rCRAZY BigInt: 7N Ratio: -22/7 Double: 2.78 -1.2e-5 BigDecimal: 4.2M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+ - * / quot rem mod inc dec max min</td>
</tr>
<tr>
<td>Compare</td>
<td>= == not= &lt; &gt; &lt;= &gt;= compare</td>
</tr>
<tr>
<td>Bitwise</td>
<td>bit-{and, or, xor, not, flip, set, shift-right, shift-left, and-not, clear, test} (1.6) unsigned-bit-shift-right</td>
</tr>
<tr>
<td>Cast</td>
<td>byte short int long float double bigdec bigint num rationalize</td>
</tr>
</tbody>
</table>

Transients (clojure.org/transients)

Create transient persistent!

Change conjl popl associ dissocl disj! Note:
always use return value for later changes, never original!

Misc

Compare = == identical? not= not compare clojure.data/diff

Test true? false? instance? nil? (1.6) some?

Sequences

Creating a Lazy Seq

From collection seq vals keys rseq subseq rsubseq

From producer fn lazy-seq repeatedly iterate
Equality

We shall contemplate truth by testing reality, via equality

(= true)
Intro to Strings

Difficulty: Elementary
Topics:

Clojure strings are Java strings. This means that you can use any of the Java string methods on Clojure strings.

```clojure
(= ___ (.toUpperCase "hello world"))
```

Code which fills in the blank:
Try Clojure

http://tryclj.com

Welcome to Clojure! You can see a Clojure interpreter above - we call it a REPL. Type `next` in the REPL to begin.

©2011-2012 Anthony Grimes and numerous contributors.
Hello World in Clojure

(print "Hello World")

"Hello, from Clojure"
Elements of Clojure Code

symbols
keywords
literals
lists
vectors
maps
sets

functions
macros
special forms (functions)
REPL

Read-Eval-Print Loop
Light Table - front end to Clojure REPL

Executable code (program) in repl

"hi there"

42

[1 2 3]

(+1 2)
Compiling Clojure

Compiles to Java byte code
  Java Classes & methods

Packaged into jar file
  One function marked as entry point

Clojure can call Java code

Java can call Clojure code

Can create stand alone applications
Chain of functions calling functions

(defn factorial
  [n]
  (if (= n 1)
    (biginteger 1)
    (* n (factorial (- n 1)))))

(factorial 20)
Clojure Function Calls

foo(1, "cat")

C function call

(foo 1 "cat")

Clojure function call

Function Name

Arguments
## Some Basic Operations

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+ 1 2)</td>
<td>3</td>
</tr>
<tr>
<td>(+ 1 2 4 6)</td>
<td>13</td>
</tr>
<tr>
<td>(= &quot;cat&quot; &quot;dog&quot;)</td>
<td>false</td>
</tr>
<tr>
<td>(= 1 1 )</td>
<td>true</td>
</tr>
<tr>
<td>(= 1 1 2)</td>
<td>true</td>
</tr>
<tr>
<td>(even? 8)</td>
<td>true</td>
</tr>
<tr>
<td>(/ 10 2)</td>
<td>5</td>
</tr>
<tr>
<td>(/ 10 2 3)</td>
<td>5/3</td>
</tr>
<tr>
<td>(bit-shift-left 4 1)</td>
<td>8</td>
</tr>
</tbody>
</table>

Thursday, August 28, 14
No built-in operators

Just functions

(if (> x y)
  "cat"
  "dog")

true value

false value
Assignment

No built-in operators

Just functions

(def a 10)

(def b (+ a 12))

(def a 20)

Called a binding which is sort of like assignment
No Precedence

\[ a - b \times c + d \]

\[ (- a (+ (* b c) d)) \]

Clojure expressions read inside out

Will see several ways to change this
Data Structures
Basic Data Elements

symbols
keywords
literals
lists
vectors
maps
sets
Symbols

Can reference another value

(def foo 12)
(defn bar [n] (inc n))

When evaluated returns the value

foo 12
bar fn

When quoted & evaluated returns it self

'foo foo
'bar bar
Symbols

Can start with any non-numeric character

Can contain alphanumerical characters and ! $ % & * - + = _ | < > ?

(def !$%&*-+=_|<>? "wtf")

Unicode is supported

(def నాలుగు 12) (def రెండు 12) (def చందనిరంతం 12)
(def బార్రా 12) (def కండిచండి 12)
(def బార్రా 12) (def ఘనప్ప 12)
Keywords

Like symbols but evaluates to itself

Literal syntax starts with a colon

:foobar
:2
:? :ThisIsALongKeyWordWhichShowsThatTheCanBeLong

Colon is part of literal syntax, but not the name of the keyword

(= :cat (keyword "cat")) true
(= :cat (keyword ":cat")) false
Literals - Strings & Characters

"A String"

"Another string"

(class "cat")

java.lang.String

\c
\u00ff
unicode
\o64
octal
Whitespace Characters

\space
\newline
\formfeed
\return
\backspace
tab

So what is \n ?
\n verses "\n"

\n
"\n"

Character n newline in string

(str "a" \n "b" 5)  "anb5"

(str "a" \newline "b")  "a
b"

(str "a" "\n" "b")  "a
b"
### Numbers

<table>
<thead>
<tr>
<th>Long</th>
<th>Double</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>2.11</td>
<td>3/5</td>
</tr>
<tr>
<td>0xfe</td>
<td>1.3e-4</td>
<td></td>
</tr>
<tr>
<td>2r111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5r123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36xCRAZY</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BigInt</th>
<th>BigDecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>12N</td>
<td>4.2M</td>
</tr>
</tbody>
</table>

(factorial 100N)

```
93326215443944152681699238856266700490715
96826438162146859296389521759999322991560
89414639761565182862536979208272237582511
852109168640000000000000000000000N
```
Cast/Convert

byte
short
int
long
float
double
bigdec
bigint
num
rationalize
biginteger

(long 12.8) 12
(rationalize 0.25) 1/4
(read-string "12.6") 12.6
(str 12.3) "12.3"
Collections

Immutable
Heterogeneous
Persistent

Vectors
Sets
Maps
Lists
Queues
Vectors

Expandable, indexed list

Fast insert at end

Expensive insert in front

Fast indexed loopup

[4 "cat" \c]

[4, "cat", \c]

[ ]
Vectors

(vector 8 4 2) [8 4 2]
(nth [:a :b :c] 2) :c
(get ["a" "b" "c"] 2) "c"
(["a" "b" "c"] 2) "c"
(nth [:a :b :c] 2 "rat") :c
(nth [:a :b :c] 4 "rat") "rat"
(.indexOf ["a" "b" "c"] "b") 1
(peek ["a" "b" "c"]) "c"
(pop ["a" "b" "c"]) ["a" "b"]
(conj [1 2 3] 4) [1 2 3 4]
(assoc [1 2 3] 0 9) [9 2 3]
## Accessing Elements - 3 ways

<table>
<thead>
<tr>
<th>Condition</th>
<th>nth</th>
<th>get</th>
<th>Vector as function</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil vector</td>
<td>Returns nil</td>
<td>Returns nil</td>
<td>Exception</td>
</tr>
<tr>
<td>Index out of range</td>
<td>Exception or &quot;Not found&quot; arg</td>
<td>Returns nil</td>
<td>Exception</td>
</tr>
<tr>
<td>Not found arg</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Immutability & Persistence

(def a [1 2 3])
(def b (conj a 4))
(def c (assoc b 0 8))

\[
\begin{array}{c}
\text{a} \Rightarrow [1 \ 2 \ 3] \\
\text{b} \Rightarrow [1 \ 2 \ 3 \ 4] \\
\text{c} \Rightarrow [8 \ 2 \ 3 \ 4]
\end{array}
\]

Java

int[] d = {1, 2, 3};
d[0] = 8;

\[
\begin{array}{c}
\text{d} \Rightarrow \{8, 2, 3\}
\end{array}
\]
Vectors implemented a B-Trees

\[
\begin{array}{c}
3 & 6 & 7 & 7 \\
5 & 8 & 8 & 8 \\
7 & 7 & 7 \\
\end{array}
\]
Vectors implemented a B-Trees