Beginner Clojure Code
Masters Exam Website

Implemented by four undergraduate students from Brazil

No Clojure experience

No Functional Programming experience

Only one had any web programming experience
Long Sequences of composed functions

Used recursion rather than higher order functions
  map/reduce/filter

Towards the end started using higher order functions
## Issue: Displaying Dates

### Exams list

<table>
<thead>
<tr>
<th>Exam ID</th>
<th>Enrolled</th>
<th>Title</th>
<th>Exam Date</th>
<th>Exam Hours</th>
<th>Exam Location</th>
<th>Term</th>
<th>Register period</th>
<th>Pass Grade</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>32</td>
<td>Database Management Systems (DMS)</td>
<td>Thursday, January 14, 2016</td>
<td>4:30pm - 6:30pm</td>
<td>GMCS 333</td>
<td>Spring 2016</td>
<td>11/11/2015 to 11/30/2015</td>
<td></td>
<td>Edit, Delete, Grades</td>
</tr>
<tr>
<td>5</td>
<td>47</td>
<td>Programming Languages (PL)</td>
<td>Wednesday, January 13, 2016</td>
<td>4:30pm - 6:30pm</td>
<td>GMCS 333</td>
<td>Spring 2016</td>
<td>11/11/2015 to 11/30/2015</td>
<td></td>
<td>Edit, Delete, Grades</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>Operating Systems &amp; Architecture (OSA)</td>
<td>Tuesday, January 12, 2016</td>
<td>4:30pm - 6:30pm</td>
<td>GMCS 333</td>
<td>Spring 2015</td>
<td>11/11/2015 to 11/30/2015</td>
<td></td>
<td>Edit, Delete, Grades</td>
</tr>
</tbody>
</table>
(defn convert-date-to-calendar-format
  "Get a whole vector of maps and convert a date type (yyyy-mm-dd) to
  Calendar type (Day, Month Day, Year)"

  ([vector-of-maps key] (convert-date-to-calendar-format vector-of-maps key []))

  ([vector-of-maps key result]
    (if (empty? vector-of-maps)
      result
      (convert-date-to-calendar-format
        (rest vector-of-maps)
        key
        (conj
          result
          (assoc (first vector-of-maps)
            (keyword key)
            (f/unparse calendar-formatter (c/from-sql-date
              ((keyword key) (first vector-of-maps)))))\))))))

  All the work is done in last argument of the last argument of the last argument of the recursion
(defn date->string
  "Convert a java.sql.Date object to string Full date format
  ie Thursday, January 14, 2016"
[sql-date]
(->> sql-date
  c/from-sql-date
  (f/unparse calendar-formatter)))
(defn convert-date-to-calendar-format
  "Get a whole vector of maps and convert a date type (yyyy-mm-dd) to
  Calendar type (Day, Month Day, Year)"
  ([vector-of-maps key] (convert-date-to-calendar-format vector-of-maps key []))

  ([vector-of-maps key result]
   (if (empty? vector-of-maps)
     result
     (convert-date-to-calendar-format
      (rest vector-of-maps)
      key
      (conj
       result
       (assoc (first vector-of-maps) (keyword key) (date->string
        ((keyword key) (first vector-of-maps)))))))))))
Using Higher Order Functions

(defn date->string
  "Convert a java.sql.Date object to string - Thursday, January 14, 2016"
  [sql-date]
  (->> sql-date
       c/from-sql-date
       (f/unparse calendar-formatter)))

(defn convert-date-to-calendar-format
  "Convert value at key from java.sql.Date to string - Full date format"
  [vector-of-maps key]
  (mapv #(update % key date->string) vector-of-maps))
Issue: Entering Student Requests in Database

Students can sign up for 1-3 exams
(defn insert-request!
  "Manipulate request to be sent to database"
  [current-id request-map]
  (when (string? (:exam_id request-map))
    (let [new-exam-id (Integer. (:exam_id request-map))]
      (insert-request-to-db! (assoc request-map :exam_id new-exam-id
                                :request_date (l/local-now)
                                :student_redid current-id))))

(defn insert-exam-request!
  "Add exam request to student in database
   current-id Int student Red Id requesting exam
   request-map {::exam_id IntOrString, _} Id of exam to add"
  [current-id request-map]
  (let [exam-id-int (Integer. (:exam_id request-map))]
    (insert-request-to-db! {::exam_id exam-id-int
                            :request_date (l/local-now)
                            :student_redid current-id})))
(defn insert-multiple-requests!
  "Insert multiple exam requests"
  [current-id request-map]
  (if (vector? (:exam_id request-map))
      (let [request (core/from-map-of-vector-to-vector-of-maps-request request-map)]
        (loop [req request]
          (when (not-empty req)
            (insert-exam-request! current-id (first req))
            (recur (rest req)))))
      (insert-exam-request! current-id request-map)))
(defn from-map-of-vector-to-vector-of-maps-request
  "Change the structure from map of vectors to vector of maps"
  [map-of-vectors]
  (vec (for [x (range (count (:exam_id map-of-vectors)))]
    {:exam_id (nth (:exam_id map-of-vectors) x)})))

This is only used in one function - insert-multiple-requests!
(defn insert-multiple-requests!
  "Insert multiple exam requests"
  [current-id request-map]
  (if (vector? (:exam_id request-map))
    (let [request (core/from-map-of-vector-to-vector-of-maps-request request-map)]
      (loop [req request]
        (when (not-empty req)
          (insert-exam-request! current-id (first req))
          (recur (rest req))))
      (insert-exam-request! current-id request-map)))

(defn insert-multiple-requests!
  "Insert multiple exam requests
  request-map {:exam_id IntOrString} or {:exam_id [IntOrStrings]}"
  [current-id request-map]
  (if (vector? (:exam_id request-map))
    (doseq [exam-id (:exam_id request-map)]
      (insert-exam-request! current-id {:exam_id exam-id}))
    (insert-exam-request! current-id request-map)))
Simpler code

Improved function names

More information about arguments
Step in Processing Students Request

Get data from web page

Validate data
   1-3 exams
   No exams that meet at same time

If errors display them to user

Convert data into format needed by database

Enter data into database

All done as one thing
(defn request-exam
  "Requests an exam"
  [request exams]
  (let [current-id {:redid (Integer. (:identity request))}]
    registered-exams (student/get-active-registered-exams current-id)
    registered-exams-after-calendar (core/convert-date-to-calendar-format registered-exams :exam_date)
    exam (exam/filter-exams-for-registration (exam/get-available-exams current-id))
    exam-after-calendar (core/convert-date-to-calendar-format exam :exam_date)
    exam-after-slash (core/convert-dash-to-slash-format exam-after-calendar :register_start :register_end)]
  (if (too-many-exams (:exam_id exams) registered-exams-after-slash)
    (layout/render "students/exam-request.html" {:exams exam-after-slash :registered-exams registered-exams-after-slash :request request :error "You can only register for three exams"})
  (if (exam/verify-conflict-exam-requests exams)
    (layout/render "students/exam-request.html" {:exams exam-after-slash :registered-exams registered-exams-after-slash :request request :error "You cannot register for exams given at the same time"})
  (let [current-id (Integer. (:identity request))]
    (try
      (exam/insert-multiple-requests! current-id exams)
      (response/redirect "/masters/students/request-exam")
      (catch Exception e
        (timbre/error e
          (response/redirect "/masters/students/request-exam")
        ))))))
How to make sure it works

Display web page

Enter data

See what happens

Debugging behind web server using web browser sucks

The structure of the program makes it hard to debug/maintain/extend
Make Independent

Get data from web page

Validate data

Convert data into format needed by database

Enter data into database

You can test at least the last two in
REPL
Unit tests

Testing database is work
Seperate converting data
From adding to database
The End of Dynamic Languages

By Elben Shira

Nov 22, 2015

http://elbenshira.com/blog/the-end-of-dynamic-languages/
Used Clojure in the past

Working in Scala (work) and Haskell (side project)

Spent a week doing Ruby & Clojure

Uncertainty

What are the arguments to the functions

(defn convert-date-to-calendar-format [vector-of-maps key]
AppsFlyer

Mobile Analytics Company
Based in San Francisco
2 Billion events per day
Traffic double in 3 months
Grew from 6 to 50 people past year

Technologies used
Redis, Kafka, Couchbase, CouchDB, Neo4j
ElasticSearch, RabbitMQ, Consul, Docker, Mesos
MongpDB, Riemann, Hadoop, Secor, Cascalog, AWS
AppsFlyer - Python Based

Started code base in Python

After two years python could not handle the traffic

Problems caused by
  String manipulations
  Python memory management
Their options

Rewrite parts in C & wrap in Python

Rewrite in programming language more suitable for data processing

Wanted to try Functional Programming
Scala vs. OCaml vs. Haskell vs. Clojure

Scala
   Functional & Object Oriented
   They wanted pure Functional

OCaml
   Smaller community
   Only one thread runs at a time even on multicore

Haskell
   Monads made us cringe in fear

Clojure
   Runs on JVM
   Access to mutable state if needed
   Now have 10 Clojure engineers
Monads

What are they?

Why do they make engineers cringe in fear?
Monoids & Monads
Monoid

Binary Function
Two parameters

Parameters and returned value have same type

Identity value

Associatively

Integer +

2 + 1

2 + 0

(2+3) + 4 = 2 + (3 + 4)
Monoid

Binary Function
  Two parameters

Parameters and returned value - same type

Identity value

Associatively

Java String concat

“hi”.concat(“ Mom”);

“hi”.concat(“”)

“hi”.concat(“Mom”.concat(“!”))

“hi”.concat(“Mom”).concat(“!”)
Monoid

Binary Function
   Two parameters

Parameters and returned value - same type

Identity value

Associatively

Sets union

“hi”.concat(“ Mom”);

“hi”.concat(“”)

“hi”.concat(“Mom”.concat(“!”))
“hi”.concat(“Mom”).concat(“!”)
Monoid

Associative binary function $F : X^X \rightarrow X$
that has an identity
Haskell

class Monoid m where
    mempty :: m
    mappend :: m -> m -> m
    mconcat :: [m] -> m
    mconcat = foldr mappend mempty
Monad - Some Motivation

Exceptions
    Interrupt program flow

(filter foo [a b c d e f g h])
Swift - optionals

let possibleNumber = "123"
let convertedNumber = possibleNumber.toInt()

if (convertedNumber)
    print( convertedNumber! )
Pyramid Of Doom

let b = foo(a)
if b
    let c = bar(b)
    if c
        let d = fooBar(c)
        if d
            let e = barFoo(e)
            if e
                return e!
            return "No e"
        return "No d"
    return "No c"
return "No b"
(-> some-collection
   foo
   bar
   fooBar
   barFoo)

What if one of the functions (foo, etc) returns an optional?

All the rest of the functions need handle them
Haskell Monad

Contains a context & four functions

return
  return :: a -> m a
  Takes a value and wraps in a monad

bind
  (>>=) :: m a -> (a -> m b) -> m b
  Take a
  monad
  function that requires a regular value and returns a monad
  Applies the function to the monad
Haskell Monad

Contains a context & four functions

>>(>>) :: m a -> m b -> m b
First argument is ignored

Error
What are Monads used for?

In Haskell all functions are pure

Monad contexts can have side effects

All I/O in Haskell is done in monads

Monads allow you to compose computational steps together
Monads in Clojure

let
for
->
->->
Monads Tutorial For Clojure Programmers

http://onclojure.com/2009/03/05/a-monad-tutorial-for-clojure-programmers-part-1/