Clojure

The Revenge of Data

by
Vjeran Marcinko
Kapsch CarrierCom

Data Processing is what we do

- Most programs receive, transform, search, and send data
- Data is raw, immutable information
- In its essence, data is simple thing
- But lots of languages, primarily OO ones, add extra stuff to data (making data active and mutable, adding functions to it etc...)

This addition of extras to data, was this good idea?

Clojure

- Developed by Rich Hickey in 2007
- It’s a LISP
  (yours truly since 1958)
- Runs on JVM and JavaScript engines
- Dynamic, functional language

Clojure Data Literals

- There are data literals to define basic data types:
  - String (”John”), Long (123), Null (nil), Boolean (true), Keyword (:name), Symbol (+money)...
- But also for most common collections:
  - Vector: [1 2 nil ”John” :name :age nil]
  - List: (1 2 nil ”John” :name :age nil)
  - Set: #{2 3 4 5}
  - Map: {:name ”John”, :age 15, :active false}
- Great for copy-pasting, logging, debugging, sampling ...

Data-Function separation

- Unlike OO, it doesn’t add functions to data
- Function calling syntax:
  - Most languages: someFunction(2, ”John”)
  - LISP/Clojure: (someFunction 2 ”John”)
- Example:

  (activate-user {:name ”John”, :active false})

REPL (1)

Remember this?

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
```

REPL (2)

- R(ead)-E(val)-P(rint)-L(oop)
- Interactive development environment
- Quickly testing functions with sample data

- (+ 2 3)
  => 5
- (str ”Hello ” ”World”)
  => ”Hello World”
- (concat [1 2] [3 4 5])
  => (1 2 3 4 5)

REPL (3)

All LISPs have it, and frequently praised for bringing developers in state of “flow”

Data Immutability

- All data structures are immutable, functions cannot modify them, they only produce new data

```
(def user1 {:name “John”, :active false})
(def user2 (activate-user user1))
user1
=> {:name “John”, :active false}
user2
=> {:name “John”, :active true}
```

These generic data structures (lists, maps, sets…) are used everywhere…

Configuration files

```
{:tomcat-port 8080
 :db {:jdbc-url ”jdbc:h2://testdb”
     :username ”myuser”
     :password ”mypass”}
 :mail {:protocol ”smtp”
       :host ”10.234.10.11”
       :port 25}}
```

HTML templates

- Using vectors for HTML tags:

```
[:html
  [:body
    [:div {:id "email" :class "selected starred"} "…"]
```

- Natural mixing of code and HTML:

```
[:ul
  (filter is-user-active users
    [:li (:name user)]))]
```

SQL

- Map as SQL command:
```
{
:select ["e.name" "e.age" "c.name"]
:from   [["employee" "e"]
         ["company" "c"]]
:where  [...]
}
```
- Great when you need to construct SQL programmaticaly (eg. complex search forms)

Metadata

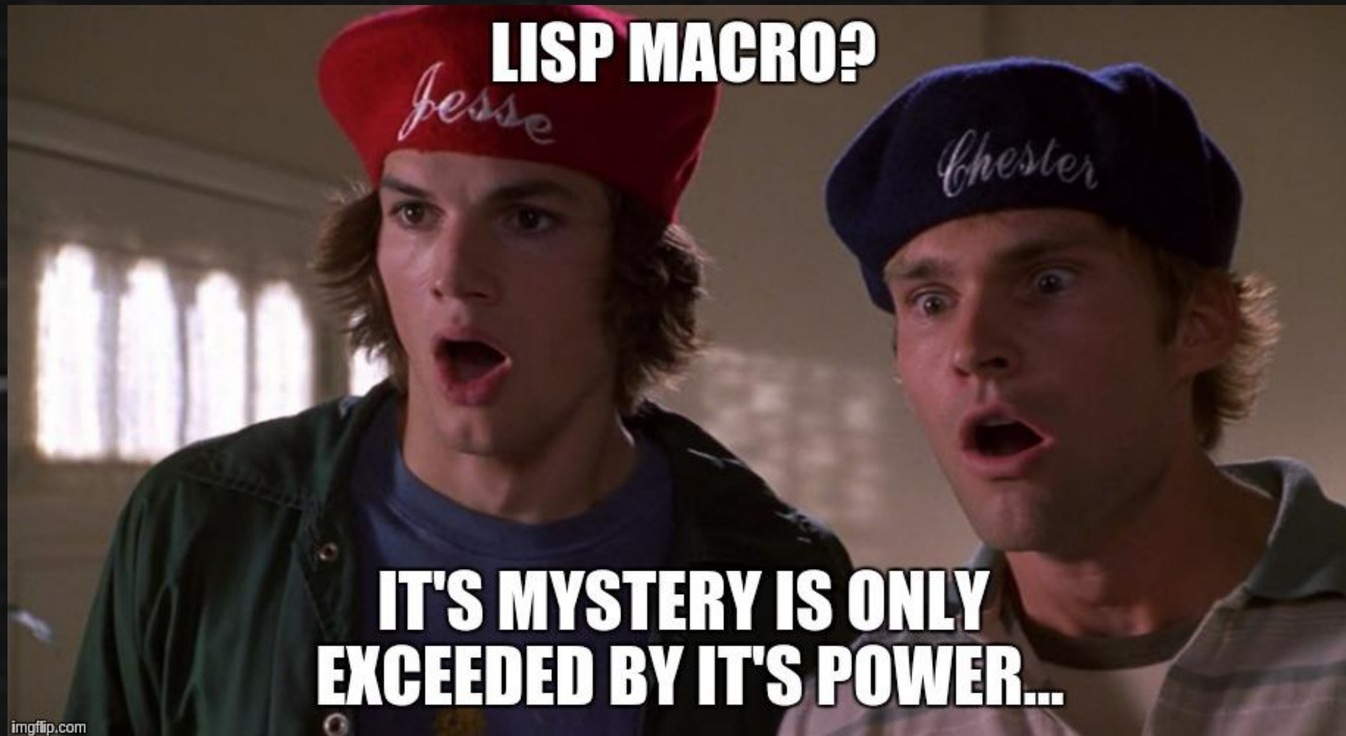
- Metadata – data about data
- In Java, we have annotations – special API
- In Clojure, we ”attach” a map to a data:

```
(def my-person
  (with-meta
    {:name ”John” :age 32} ;; data
    {:auth-role :role/admin}) ;; meta
```

Code is data (1)

- Most popular feature of all LISPs – ”homoiconicity” (code is data, data is code)
- Code is represented by nested lists → LIS(T)P(rocessing)
- Example:
  (+ 2 3)
  - It is code
  - It is data - a list of 3 elements: symbol +, number 2 and number 3

Code is data (2)



- This ”code is data” feature is used by ”macros” – functions called during compile time that take code and produce new code

Code is data (3)

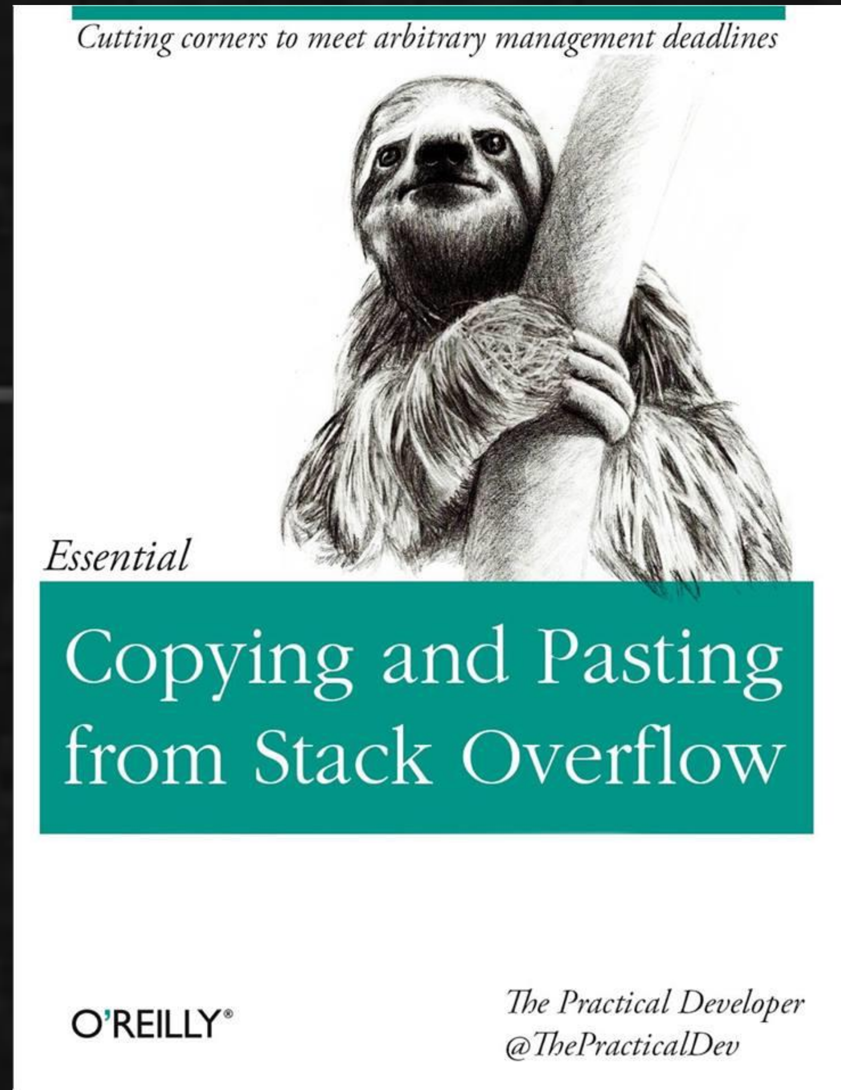
- Example - I want a new syntax where function calls are defined backwards, like:
  (2 3 +)
  => 5
- (defmacro reversed-call [& original]
  (reverse original)) ;; creating code
- (reversed-call 2 3 +)
  => 5
- Code (2 3 +) is first transformed to new code (+ 3 2) by ”reversed-call” macro and then evaluated
- Macros enable LISPs to develop new languages (DSLs)

General vs specific data

- Each class is specific data type that has to provide it's API (methods) for rest of the world to use it
- By using general data structures we reuse:
  - all the existing functions that work with them
  - knowledge about their behaviour – eg. equality, printout, data literals ...

Clojure popularity?

- Not mainstream, but sufficiently widespread to practice SDD
- SDD (StackOverflow-Driven Development)



Shameless quote...

“A lot of the best programmers and the most productive programmers I know are writing everything in Clojure and swearing by it, and then just producing ridiculously sophisticated things in a very short time. And that programmer productivity matters. “

- Adrian Cockroft
(of Netflix / Microservices fame)

Game Over