

Introduction to ClojureScript and Functional Programming



200ok GmbH

Alain M. Lafon, 24.07.2018

alain@200ok.ch

Outline

Introduction

Complexity, Functional Programming and Clojure

Live Demo

Resources

Questions

Introduction

Alain M. Lafon



CV

- CEO 200ok GmbH
- CEO QuickShift GmbH
- Lecturer at ZHAW
- Zen Monk, runs the Lambda Zen Temple

Contact

alain@200ok.ch

- I'll cover a lot of ground in this talk
- However, this is a lightning talk

-
- Please ask questions at any time, when something is unclear or you're simply curious

Any LISPers in the room?

```

/* Don't assume argc/argv's give up in advance. */
if (argc == "skiptr + 1)
    return 0;

arg = argv["skiptr+1"];
if (arg == NULL)
    return 0;
if (strcmp (arg, str) == 0)
    if (valptr != NULL)
    {
        *valptr = argv["skiptr+2"];
        *skiptr += 2;
    }
    else
        *skiptr += 1;
    return 1;

arglen = (valptr != NULL && (p = index (arg, "=")) != NULL
    ? p - arg : strlen (arg));
if (last == 0 && arglen < strlen 1) strcmp (arg, last, arglen) != 0)
    return 0;
else if (valptr == NULL)
    {
        *skiptr += 1;
        return 1;
    }
    else if (p != NULL)
    {
        *skiptr = p+1;
        *skiptr += 1;
        return 1;
    }
    else if (argv["skiptr+2"] != NULL)
    {
        *valptr = argv["skiptr+2"];
        *skiptr += 2;
        return 1;
    }
    else
    {
        return 0;
    }
}

first DOES LEA HALLOC

/* malloc can be invoked even before main (e.g. by the dynamic
linker), so the dumped malloc state must be restored as early as
possible using this special hook. */

static void
malloc_initialize_hook ()
{
    #ifndef USE_CRT_DLL
    extern char **environ;
    #endif

    if (initialized)

```



```

"beginning-of-defun".

If variable 'end-of-defun-function' is non-nil, its value
is called as a function to find the defun's end."
(interactive "p")
(or (not (eq this-command "end-of-defun"))
    (or last-command "end-of-defun")
    (and transient-mark-mode mark-active)
    (point-max))
(or (and arg1 & arg2) (setq arg 1))
(if end-of-defun-function
    (or (or arg 0)
        (define (< arg)
            (downcase end-of-defun-function)))
    ;; Better not call beginning-of-defun-function
    ;; directly, as some A&T's use default.
    (beginning-of-defun (< arg)))
<let (<first 1)
    (while (and (< arg 0) (< (point) (point-max)))
        (let (upper (point)))
            (while (point)
                (let (last-char)
                    (end-of-line 1)
                    (beginning-of-defun raw 1)))
                (let (last) (forward-char -1))
                    (beginning-of-defun raw -1))
                (setq upper nil)
                (forward-char 1)
                (if (looking-at "[^a-zA-Z]*")
                    (forward-char 1))
                (or (point) point)))
    (setq arg (< arg))
    (while (< arg 0)
        (let (upper (point))
            (beginning-of-defun raw 1)
            (forward-char 1)
            (forward-line 1)
            (if (< (point) point)

```

Free as in Freedom

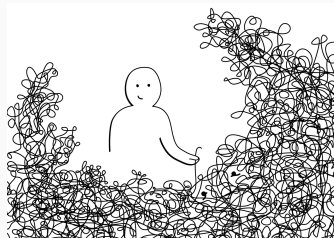
gnu.org

- Did some of you start using LISP because of Clojure?

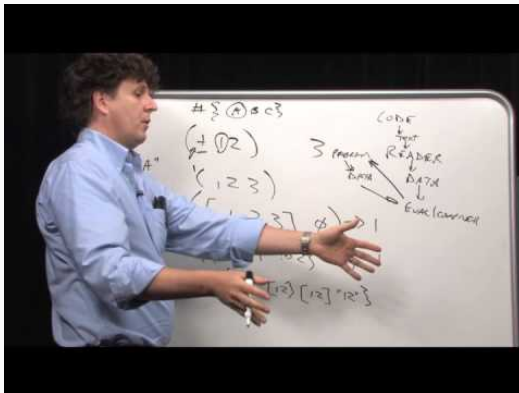


Complexity, Functional Programming and Clojure

- The complexity of software is growing at an exponential rate
- The biggest problem is the growing complexity of dynamic state which makes it hard to reason about a system



"State - You're doing it wrong!"



Rich Hickey, creator of the Clojure

Quote Source: <http://clojure-log.n01se.net/date/2008-06-26.html#11:10a>

State is hard. And on top, we often use **OOP** and **mutable state** to make things worse. It is very easy to add incidental complexity to your application. Just look at this very normal code which probably doesn't look complex at all:

```
var x = 1;  
var y = x + 1;  
y == 2 == true;  
x = 2;  
y == 3 == false;
```

Due to **mutable state**, `y` was not updated after `x` has been changed. This makes reasoning about the current state and logic of the application hard.

The UI layer is most predictable when it is described as a pure function of the application's state.

Some of the benefits of Clojure/ClojureScript are:

- It's a Lisp! (Code as Data)
- Functional language with immutable persistent data structures
- Almost no syntax
- Amazing tooling, Editor integration, performance
- Use the same language on back-end and front-end
- Uses Google Closure to optimize your code and included libraries
- Uncle Bob believes it to be the last programming language

Syntax example (counting clicks) written in Reagent which builds on top of ReactJS.

```
(ns example
  (:require [reagent.core :as r]))
(defonce click-count (r/atom 0))

(defn counting-component []
  [:div
    "Button has been clicked" @click-count " times"
    [:input {:type "button" :value "Click me!"
             :on-click #(swap! click-count inc)}}]])
```

Live Demo

Live Coding - what can go wrong?



Resources

Slides and minimal example app

Will be uploaded later to <https://200ok.ch>



Video

- Simple made Easy (Rich Hickey): <https://www.infoq.com/presentations/Simple-Made-Easy>
- Figwheel Demo (Bruce Hauman):
<https://www.youtube.com/watch?v=KZjFVdU8VLI>

To read

- Clojure for the Brave and True:
<http://www.braveclojure.com/>
- <https://200ok.ch/category/clojure.html>
- 200ok.ch/atom.xml

In case you're searching for a job, talk or write to me!

Questions
