FROM PYTHON TO CLOJURE

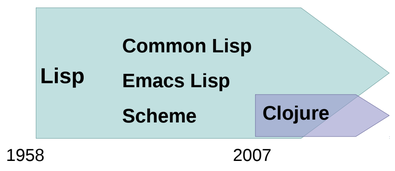
A newbie's tale by Eleonore Mayola

Eléonore Mayola, PhD - @EleonoreMayola
Junior data scientist @MastodonC, Co-organiser @PyLadiesLondon

The Fellowship of the Lisp

Have you heard about Lisp? Its name derives from "LISt Processing" and it's one of the earliest programming language.

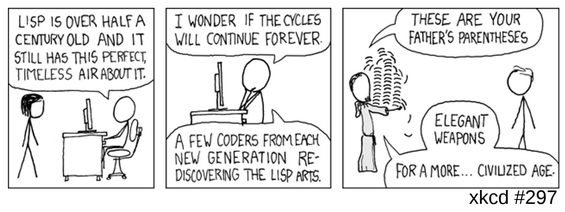
Clojure is a Lisp



Fast, powerful and expressive

Code-as-data approach

Macro system



xkcd #297

Clojure runs on the JVM

Clojure is hosted on the Java Virtual Machine (JVM). It relies on the JVM for core features and can use Java libraries.

Clojure programs are compiled to Java bytecode by the Clojure compiler and executed by the JVM

Basic syntax

To Lisp' nested lists (S-expressions) it adds vectors, maps and sets

ordered
- **List:** `("Lisp" "Clojure" "Python")`
- **Vector:** `[1958 2007 1991]`

un-ordered
- **Map:** `{:lisp 1958, :clojure 2007, :python 1991}`
- **Set:** `#{"Lisp" "Python" 1958 "Clojure" 2007 1991}`

(function operand1 operand2)

```
user> (defn is-Lisp? [language]
        (if (contains? #{"common-lisp" "emacs-lisp"
                         "scheme" "clojure"}
                       language)
          true
          false))
#'user/is-Lisp?
user> (is-Lisp? "clojure")
true
user> (is-Lisp? "python")
false
```

~$ Tools

User=> (install Leiningen[1])

Standard tool to create/run a Clojure project + nREPL (Read-eval-print loop)

```
~$ lein repl
user=> lein new app clj-app
user=> lein run
user=> lein test
lein uberjar java -jar target/uberjar/clj-app-0.1.0.standalone.jar
```

User=> (Clojure-friendly IDE)

Emacs, Light Table, Eclipse, Vim, Cursive, NightCode...

User=> (smartparens/Paredit)

Keeps parentheses balanced for your sanity!

xkcd #859

The Two Paradigms

What's all this fuss about functional programming? Where should you start?

Imperative vs functional

The imperative paradigm describes computation in terms of statements that change a program state.

Example: Python

The functional paradigm treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

Example: Clojure

Clojure is functional

To avoid mutable state Clojure provides immutable data structures, first-class functions and emphasizes recursive iteration over side-effect based looping.

Immutable lists, vectors, sets and maps

→ Easy way to avoid mutating state
→ Adding to or removing from the a collection means creating a copy with the changes

```
user> (conj ["repl" "paredit"] "lein")
["repl" "paredit" "lein"]
```

First-class functions

→ anonymous functions: fn
→ associated to a variable: def, defn

```
user> (defn Bonjour
        [friend]
        (println "Bonjour " friend))
#'user/Bonjour
user>
user> (Bonjour "Python")
Bonjour  Python
nil
```

Looping without side-effects

→ map, reduce, loop/recur, doseq

```
user> (def my-map {:name "Clojure"
                   :family "Lisp"})
#'user/my-map
user> (when (not (empty? my-map))
        (map (fn [m] (val m))
             my-map))
("Lisp" "Clojure")
```

~$ Learning Clojure

User=> (Books)

- **Clojure for the brave and true**, D Higginbotham
- **Practical Clojure**, L VanderHart and S Sierra
- **Clojure programming**, C Emerick, B Carper and C Grand
- **The Joy of Clojure**, M Fogus and C Houser

User=> (First steps)

4clojure[2], Clojure/ClojureScript Koans [3, 4], Clojure katas [5]

User=> (Community)

Coding dojos, Meetups and conferences are beginners friendly. Also ClojureBridge[6]!

The Return of the Python

Can we use functional programming features while writing Python?

Python functional style

Tuple: immutable data structure

```
In [1]: montreal = (45.50866990, -73.55399250)
```

List / Dict comprehension:

```
In [1]: [i for i in range(5) if i % 2 == 0]
Out[1]: [0, 2, 4]
```

Anonymous function lambda associated with filter(), map(), reduce()

```
In [2]: values = [1.2, 7.4, 3.9, 6.5]

In [3]: print filter(lambda x: x < 5, values)
[1.2, 3.9]

In [4]: print map(lambda x: x * 2, values)
[2.4, 14.8, 7.8, 13.0]

In [5]: print reduce(lambda x, y: x + y, values)
19.0
```

Iterators and **generators**

```
In [1]: values = iter([1.2, 7.4, 3.9, 6.5])

In [2]: print values
<listiterator object at 0x7f2f4238e950>

In [3]: values.next()
Out[3]: 1.2

In [4]: values.next()
Out[4]: 7.4
```

Modules and **functions**

→ Functional programming forces to **break down a problem into small bits**.
→ Using **small functions** that each perform **one task**.
→ This also makes it **easier to write and read** programs
→ It is possible in Python to write such small functions inside of modules when needed

~$ Code examples [7]

Clojure:

```
(def data
  (str/split-lines (slurp "data.csv")))

(def headers
  (str/split (first data) #","))

(defn list-data []
  (mapv #(zipmap headers
                 (str/split % #","))
        (rest data)))
```

Python:

More classic

```
def process_csv(filename):
    '''Open the file and build a list
    containing dictionnaries of data.'''
    list_data = []

    with open(filename, 'rb') as csvfile:
        content = csv.reader(csvfile)
        headers = content.next()
        data = content

        for row in data:
            dict_data = {}
            for i in range(len(headers)):
                dict_data[headers[i]] = row[i]
            list_data.append(dict_data)
    return list_data
```

More functional

```
def process_csv(filename):
    with open(filename, 'rb') as csvfile:
        content = csv.reader(csvfile)
        headers = content.next()
        data = [row for row in content]
        list_data = [dict(zip(headers, row)) for row in data]
    return list_data
```

References:
[1] leiningen.org
[2] 4clojure.com
[3] clojurekoans.com
[4] clojurescriptkoans.com
[5] github.com/gigasquid/wonderland-clojure-katas
[6] clojurebridge.org
[7] github.com/Eleonore9/from-python-to-clojure

Other resources:
Clojure docs: clojuredocs.org | More about Java: http://www.flyingmachinestudios.com/programming/how-clojure-babies-are-made-the-java-cycle/
About Leiningen: http://www.flyingmachinestudios.com/programming/how-clojure-babies-are-made-what-leiningen-is/
Imperative vs functional: https://joshldavis.com/2013/09/30/difference-between-imperative-and-functional-part-1/ | Python to Clojure: http://zachcp.org/blog/2015/python-to-clojure/