Clojure

CRASH COURSE

Getting Started

Install Leiningen

https://leiningen.org/#install (or install ‘lein’ from your package manager)

Clone git repo

git clone https://github.com/danielytics/sherlock-and-array

Edit Code

sherlock-and-array/src/sherlock_and_array/core.clj

Run Tests

lein test

lein test-refresh

The basics

Basic Types

123

123.4

true

“This is a string”

:keyword

[“This” :is “a vector” 123]

The basics

Function calls

(println “Hello, World”)

(+ 123 456)

(inc 4)

(first [3 2 1])

(str “lots” “of” “arguments” “here”)

The basics

Functions

```
(defn hello [arg]
  (println “Hello,” arg))
```

```
(hello “World!”)
```

```
(fn [arg] (println “Hello,” arg))
```

```
#(println “Hello,” %)
```

The basics

Conditionals

```
(if condition
  (println “Condition was true”)
  (println “Condition was false”))
```

```
(cond
  (= x 1) “One”
  (= x 2) “Two”
  :else “Many”)
```

The basics

Let Bindings

```
(let [my-identifier “hello”
      another-identifier (str “Hello,” “World”)]
  (println my-identifier)
  (println another-identifier))
```

The basics

Destructuring

```
(defn generate-data [] [1 2 3])

(let [[first second third] (generate-data)]
  (println first)
  (println second)
  (println third))

(let [[first & rest] (generate-data)]
  (println “This is the first value:” first)
  (println “These are all of the other values:” rest))

(defn foo [first second & rest] …)
```

Sequences

Creating vectors

[1 2 3]

(vector 1 2 3)

(range 10)

Basic Oerations

(count [1 2 3])

(conj [1 2 3] 4)

Sequences

Accessing

(first [1 2 3])

(second [1 2 3])

(rest [1 2 3])

(last [1 2 3])

Sequences

Slicing & Dicing

```
(take 3 [1 2 3 4 5 6 7])

(drop 3 [1 2 3 4 5 6 7])

(split-at 3 [1 2 3 4 5 6 7])

(subvec [1 2 3 4 5 6 7 8 9] 2 4)
```

Sequences

Processing

```
(map inc [1 2 3 4 5])
```

```
(map (fn [x] (* 2 x)) [1 2 3 4 5])
```

```
(map #(* 2 %) [1 2 3 4 5])
```

```
(for [element [1 2 3 4 5]]
  (* element 2))
```

Sequences

Processing

```
(reduce + [1 2 3 4 5])
```

```
(reduce
  (fn [accum next-val]
    (+ accum next-val))
  [1 2 3 4 5])
```

```
(reduce
  (fn [accum next-val]
    (+ accum next-val))
  0
  [1 2 3 4 5])
```

Sequences

Processing

(every? true? [true true true false true])

(not-every? true? [true true true false true])

(some true? [false false false true false])

Learning Resources

TUTORIAL
- https://clojure.org/guides/learn/syntax

REFERENCE
- https://clojuredocs.org

CHEATSHEET
- https://clojuredocs.org/quickref

ONLINE BOOK
- https://www.braveclojure.com/