
CLOJURE: A *Common Sense* LANGUAGE

A PREPRINT

Suresh Krishna Devendran

Department of Computer Science

University of Arizona

Tucson, AZ 85719

sureshdevendran@email.arizona.edu

Zach Keane

Department of Computer Science

University of Arizona

Tucson, AZ 85719

ztkane@email.arizona.edu

March 26, 2019

ABSTRACT

Clojure is a relatively new functional programming language that works with the established JVM platform. It offers many of the same control structures we're used to: such as switch/case statements, loops, try/catch blocks, and if-statements. The data structures are also standard: nil corresponds to null, Numbers, Strings, Characters, Keywords, Symbols, and Collections.

1 Introduction

Clojure is a functional programming language that uses the Lisp dialect, matching closely with other functional languages such as Scheme. An advantage it has over other programming languages is that it works with the established JVM platform, which is used in many industries and is very portable. Clojure is not only powerful, but easy to use, offering features such as dynamic data structures for new programmers.

2 History

Clojure is a language designed by an individual named Rich Hickey, that was created in 2007. Its current status is alive, with the most recent update being applied on December 7th, 2018. Clojure was created in order to provide a lisp, a family of languages using the Scheme/Haskell parentheses format, that's also compatible with the Java Virtual Machine. Clojure also is designed with concurrency in mind, and with all of these desires from Rich Hickey, he couldn't find a language that suited him. Therefore, Clojure was born.

3 Control Structures

Control Structures in Clojure; if statements, case switches, loops and exceptions.

3.1 Conditional Statements

Basic layout for a standard if-statement will have the format of (if (condition) trueCondition falseCondition), as shown below. Another format is the if-elif-else format as seen in the "Subprograms" example below, where we define (empty? x), (== (mod (first x) 2) 0), and finally :else to be our conditions. The argument after these conditions will be what is returned.

```
(defn isThisZero? [number]
  (if (= number 0) "yes" "no"))
```

3.2 Case Switches

Cases are handled in the following format. Based on what is passed in to `x`, we will either return "One case!" if we give 1, "Two cases!" if we give 2, or "I can't hold that many cases!" if we pass in anything else. This case statement is identical to a switch statement in other languages, but will run only once.

```
(defn myCase [x]
  (case x
    1 "One case!"
    2 "Two cases!"
    "I can't hold that many cases!"))
```

3.3 Loops

There are many, many ways to accomplish looping in Clojure, but this is the most basic example. "dotimes" is a built-in function that takes in an iterator, `i`, and a range, in this case hard-coded as 10. If we run the program, we will print the integers 0 through 9, one per line.

```
(dotimes [i 10]
  (println i))
```

3.4 Exceptions

Try/catch blocks are very basic, and look like how they work in Java. The code below shows us trying to divide by 0, and it being caught and printed using the Java syntax. Finally blocks are included, and will always run.

```
(try
  (/ 2 0)
  (catch ArithmeticException e
    (.getMessage e))
  (finally
    (println "this will always print")))
```

4 Data Types

4.1 Integers

- Decimal Integers (Short, Long, Int) - These are used to represent whole numbers. For example, 345.
- Octal Numbers - These are used to represent numbers in octal representation. For example, 015.
- Hexadecimal Numbers - These are used to represent numbers in representation. For example, 0xff.
- Radix Numbers - These are used to represent numbers in radix representation. For example, 2r1111 where the radix is an integer between 2 and 36, inclusive.

4.2 Floating Point

- The default is used to represent 32-bit floating point numbers. For example, 13.56.
- There is also the scientific representation. For example, 1.24e-13.

4.3 Char

A single character literal. Characters are defined with the backslash symbol. For example, `\a`.

4.4 Boolean

A boolean value, true or false.

4.5 String

Text literals which are represented as a chain of characters "Hello"

4.6 Nil

To represent NULL in clojure.

4.7 Atom

Atoms provide a way to manage shared, synchronous, independent state. They are reference type like refs and vars.

5 Subprograms

5.1 Checking for even

```
(defn hasEven [x]
  (cond
    (empty? x) false
    (== (mod (first x) 2) 0) true
    :else (hasEven (rest x))
  )
)
```

5.2 Return a list with the Odd digit values

```
(defn retOdd [x]
  (cond
    (empty? x) []
    (empty? (rest x)) x
    :else (cons (first x) (retOdd (rest (rest x))))
  )
)
```

5.3 Return the list of consecutive sums

```
(defn consSumHelp [a x]
  (cond
    (empty? x) []
    :else (cons (+ a (first x)) (consSumHelp (+ a (first x)) (rest x)))
  )
)

(defn consSum [x]
  (consSumHelp 0 x))
```

6 Summary

Clojure is a really fun language. It works with the well established JVM platform. Clojure was primarily designed with concurrency in mind. Clojure is a functional programming language that uses the Lisp dialect. Hence it is used in many platforms and it is very portable. The workflow using REPL is phenomenal, hence Clojure is a very competitive and compelling language to program in.

References

- [1] Clojure-programming. (n.d.). Retrieved from <https://www.safaribooksonline.com/library/view/clojure-programming/9781449310387/ch01.html>.
- [2] Rationale. (n.d.). Retrieved from <https://clojure.org/about/rationale>
- [3] Clojure Core API Reference. (n.d.). Retrieved from <https://clojure.github.io/clojure/>
- [4] R/Clojure. (n.d.). Retrieved from <https://www.reddit.com/r/Clojure/>

- [5] Phillips, M. (2015, July 14). Walmart Runs Clojure at Scale. Retrieved from <http://blog.cognitect.com/blog/2015/6/30/walmart-runs-clojure-at-scale>