Control Flow Basics

Zachary Bennett
Software Engineer

@z_bennett_ zachbennettcodes.com

Control Flow

Control flow is a term used to describe the ability to direct the order in which a computer program executes certain statements, expressions, and/or functions.

Everything is an expression!

Clojure/Java Comparison

Statements

Java and other like-minded languages use control flow statements

Expressions

Clojure uses control flow operators which are just expressions

Control Flow Operator

In Clojure, a control flow operator is a symbol that instructs the compiler to perform a certain, logical operation. These operators are expressions!

Control Flow Expressions

Conditional

Direct flow of control through logical operands

Iterative

Loop over collections to perform side effects or generate output

Exception handling

Direct flow of control by catching and throwing errors

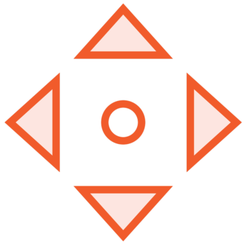
Conditional Expressions

Conditional Expression

A unit of code that evaluates to a boolean value (true or false).

Why Conditional Expressions?

Perform side effects given a certain condition



Execute specific code paths

Return different values in a function

Conditional Expressions

if

The simplest and most effective

if/do

Like “if” but allows for side effects

when

Like “if” but can only execute one code path

case

Executes a code path with a match

cond/else

Allows for chaining conditions

```
;; Conditional Expressions

; if

(if (odd? 3) true false)


; if/do

(if (odd? 3)

  (do (println “3 is odd!”)

      true)

  (do (println “3 is not odd!”)

      false))
```

- Using “if” lets you run a condition and evaluate an “if” or “else” expression

- The ”if”/”do” conditional expression gives you a bigger body to work with – this is for side effects like logging!

```
;; Conditional Expressions
; when
(when (number? input)
      (println “Input is a number!”)
      true)


; case
(case input-num
  1 “The input was 1!”
  2 “The input was 2!”
  3 “The input was 3!”
  “The input was not 1, 2 or 3!”)
```

- **Using “when” is like using “if/do” without the “else” condition.**

- **The “case” conditional executes code depending upon matches to an argument**

```
;; Conditional Expressions

; cond/else


(cond
  (= input 1) ”Input is 1”
  (= input 2) “Input is 2”
  (= input 3) ”Input is 3”
  :else “Input is not 1 or 2 or 3”)
```

- **Using “cond” with the else keyword allows you to chain many “if” conditionals together with a final “else” clause.**

- **The “else” condition is satisfied via the “:else” keyword**

Iterating

Iteration Categories

With Side Effects

Your iteration does not produce an output

Without Side Effects

Your iteration is “pure” and produces an output

Iteration With Side Effects

[1,2,3]

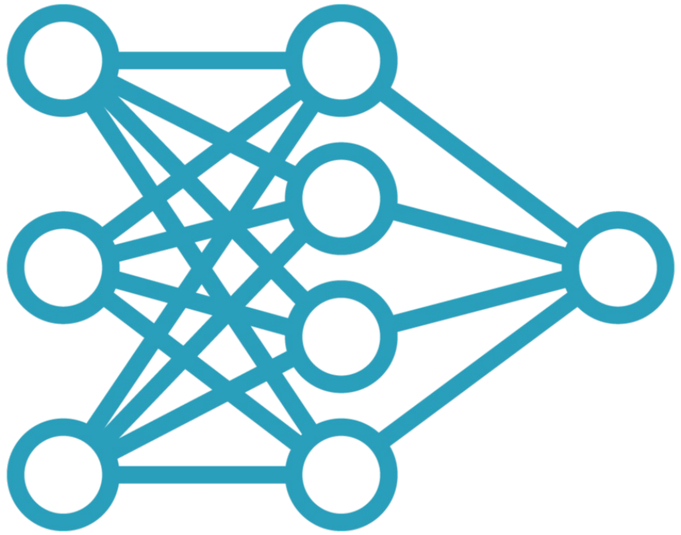
dotimes

Used to run a simple expressions "x" amount of times

doseq

Iterates over a sequence and can support multiple sequences (think nested "for" loops)

Pure Iteration



for (list comprehension)

Iterate over a sequence or multiple sequences and return an output

```
;; Iteration with side effects


(dotimes [n 5]
  (println (str “My number is: “ n)))



(doseq [num-one [1 2 3]
        num-two [4 5 6]]
  (println num-one “ “ num-two))
```

- Simple iteration over a single sequence

- Supports multiple variable bindings

- Works like a nested ”for” loop

```
;; Pure Iteration


(for [num-one [1 2 3]
      num-two [4 5 6]]
  [num-one num-two])
```

- **Iterates over one or multiple sequences**
- **Outputs a final result that can be captured**

Exception Handling

Java Interoperability

```
;; Exception Handling
```

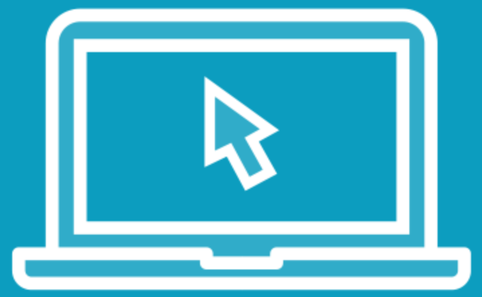
```
; Catching Exceptions
(try
  (myMethod input)
  (catch RuntimeException e ”uh-oh!”)
  (finally (println ”Always runs!")))
```

```
; Throwing Exceptions
(throw (Exception. ”uh-oh!"))
```

- Wrap code that may throw an exception with “try”
- Catch the exception using “catch”
- Use “finally” to clean up

- Use throw to throw Java exceptions

Demo



Control Flow

- Conditional expressions
- Iteration