```
(% 10 3) # this evaluates to 1

(** 2 5) # this evaluates to 32

# LOGICAL OPERATORS
    # truthy? => evaluates whether a statement is true or false
    # and
    # or
    # not

# COMPARISON OPERATORS
    # id => stricter complete equality check for type and value, the equivalent of identical object identity in Common Lisp, (keywords and symbols with the same name evaluate to true, vectors maps and sets with the same name evaluate to true only if they point to the same reference)
    # = => looser complete equality check for structural equality in type and value
    # <= < >= > all operate as expected for numeric comparisons, evaluate to a boolean

(id true true) # this evaluates to true
(id true false) # this evaluates to false
(id 5 "5") # this evaluates to false
(id :test :test) # this evaluates to true
(id 'sym 'sym') # this evaluates to true
(id '() '()) # this evaluates to false
(id [] []) # this evaluates to false
(id {} {}) # this evaluates to false

(= true true) # this evaluates to true
(= true false) # this evaluates to false
(= 5 "5") # this evaluates to false
(= 5 5) # this evaluates to true
(= 5 5.0) # this evaluates to false
(= :test :test) # this evaluates to true
(= 'sym 'sym') # this evaluates to true
(= '() '()) # this evaluates to true
(= [] []) # this evaluates to true
(= {} {}) # this evaluates to true
```

Control structures

```
# ---------- CONTROL STRUCTURE ----------

# CONDITIONAL CHECKS
    # (if test then else)

(if true 10) # this evaluates to 10 since true so truth form is evaluated
(if false 10) # this evaluates to nil since the else form is evaluated and here there is no else form
(if true (print 1) (print 2)) # this prints 1 but not 2 since (print 2) is the else form
(if nil (print 1) (print 2)) # this prints 2 since nil evaluates to false which hits the else form
(if [] (print 1) (print 2)) # this prints 2 since an empty list evaluates to nil which hits the else form

# CASE
    # (case test & pairs)
    # case => provides a terse syntax for powerful pair-based pattern-matching with match-case like statements

(case (+ 7 5)
  3 :small
  12 :big) # this evaluates to :big

(case (+ 7 5)
  3 :small
  15 :big) # this evaluates to nil

(case (+ 7 5)) # this evalutes to nil

# COND
    # (cond & pairs)
    # cond => used as an alternative to case in situations where when the first expression evaluates to boolean true, the second expression is evaluated, and if no matches are found, evaluates to nil

(cond
  (neg? 5) :negative
  (pos? 5) :positive)  # this evaluates to :positive

(cond
  (neg? 5) :negative
  (neg? 3) :negative) # this evaluates to nil

(cond) # this evaluates to nil

# LOOPS
    # loop => creates its own internal lexical scope to loop within
        # (loop [bindings] expression)
    # recur => creates a recursive loop
        # (recur expression)
    # foreach => allows for iteration over data structures
        # (foreach [value valueExpression] expression)
        # (foreach [key value valueExpression] expression)
```