```
(- 2 1) ; evaluates to 1
(* 1 2) ; evaluates to 2
(/ 2 1) ; evaluates to 2
(mod 5 2) ; evaluates to 1

(not true) ; evaluates to false
(false and true) ; evaluates to false
(false or true) ; evaluates to true

(= 1 1) ; evaluates to true
(= 2 1) ; evaluates to false
(= 1 "1") ; evaluates to true
(== 1 "1") ; evaluates to false
```

# Control structures

```
; ---------- CONTROL STRUCTURE ----------

; CONDITIONAL CHECKS
    ; if => operates as you'd expect, called as a function like everything else in clojure
        ; general syntax is (if {CONDITIONAL CHECK} {THEN EXPRESSION} {ELSE EXPRESSION})
    ; cond => evaluates each specified predicate in order and evaluates to the value of the first true condition, equivalent to an if elseif else chain in other languages
        ; :else => added as the final else case in a cond chain
    ; case => equivalent to match-case chain in other languages
        ; :else => added as the final default case in a case chain

(def x 10)
(if (> x 5)
  "Greater than 5"
  "Less than or equal to 5") ; this evaluates to "Greater than 5"

(def y 10)
(cond
  (> y 15) "Greater than 15"
  (> y 10) "Greater than 10"
  :else "10 or less") ; this evaluates to "10 or less"

(def day-of-week 3)
(case day-of-week
  1 "Sunday"
  2 "Monday"
  3 "Tuesday"
  :else "Unknown day") ; this evaluates to "Tuesday"

; LOOPS
    ; for most iterative solutions, clojure's functional programming constructs are encouraged instead of loops (map, reduce, filter)
    ; loop => creates and defines the lexical scope of a loop, used alongside recur
    ; recur => indicates to initiate another iteration of the current loop with recursion
    ; doseq => creates a loop that iterates over a specified structure, achieved without explicit recursion

(defn countdown-recur [n]
  (loop [i n]
    (if (<= i 0)
      "Blast off!"
      (do
        (println i)
        (recur (dec i))))))) ; function definition for a function that contains a recursive loop which counts down from n to 0, then prints blast off, here recur is used to decrement i by 1 and initiate another iteration of the loop

(defn countdown-iter [n]
  (doseq [i (range n 0 -1)]
    (println i))
  (println "Blast off!")) ; function defintion for a function that achieves the same thing as the above function, but does it using doseq to create a loop that does not rely on explicit recursion but instead iterates over a vector created using the range function
```