;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;PRINT & COMMENT

(println "Hello World!" )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;NEW VARIABLE

(def s 10 )

(println s )

;;;NOTES:

; 1. Variable names can only contain alphanumeric characters

; and ‘\* + ! / . : - \_ ?’ but must not begin with a numeral or colon.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; DATA TYPES

; Types are assigned based on the value assigned

(def aLong 15 ) ; -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

(def aDouble 1.23456 ) ; 4.94065645841246544e-324d to 1.79769313486231570e+308d

(def aString "Hello" ) ; A String

(def aBoolean true ) ; true or false

(type true) ; Get the data type

(nil? aLong) ; Check if no value (nil)

(pos? aLong) ; Check if a value is positive

(neg? aLong) ; Check if a number is negative

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; VARIABLES

(def aLong 100 )

(def aDouble 5.12345 )

(def aString "Hello" )

(def aBool true )

;; FORMATTED OUTPUT

(format "This is a string %s" aBool)

(format "Add n spaces: %7d" aLong)

(format " x digits with leading zeros %05d" aLong)

(format "%-5d left justified and spaces" aLong)

(format "Set n decimal places %.3f" aDouble)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(def str1 " The Cat in the Hat ")

;; STRING LIBRARY

(clojure.string/blank? str1 ) ; Check if blank

(clojure.string/includes? str1 "in" ) ; Does string contain "x"

(clojure.string/index-of str1 "the" ) ; Index of match

(clojure.string/join " " ["Cat" 1 "Hat"]) ; Collection to 1 string w/ seperator

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;REPLACE

(clojure.string/replace "a x c d" #"x" "b" ) ; Replace all Regex with a string

(clojure.string/replace-first "a x x c d" #"x" "b" ) ; Replace first

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;CHANGE CASE

(def str1 " The Cat in the Hat ")

(clojure.string/upper-case str1) ; Uppercase

(clojure.string/lower-case str1) ; Lowercase

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;TRIM

(def str1 " The Cat in the Hat ")

(clojure.string/trim str1) ; Remove white space at start/end

(clojure.string/triml str1) ; Remove left white space

(clojure.string/trimr str1) ; Remove right white space

(println str1)

(println (clojure.string/trim-newline str1)) ;Remove newline

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; MATH

(println "Add : " (+ 1 2 3) ) ; Add values together

(println "Subtract : " (- 5 3 2) ) ; Subtract values

(println "Multiply : " (\* 2 5) ) ; Multiply Values

(println "Divide : " (/ 10 5) ) ; Divide Values

(println "Modulus : " (mod 12 5) ) ; Modulus

(println "Increment : " (inc 2) ) ; Increment by 1

(println "Decrement : " (dec 10) ) ; Decrement by 1

(println "Random # : " (rand-int 20)) ; Generate random number

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;MATH 2

(println "Absolute : " (Math/abs -10) ) ; Absolute Value

(println "Cube Root : " (Math/cbrt 8) ) ; Cube Root

(println "Square Root : " (Math/sqrt 4) ) ; Square Root

(println "Power : " (Math/pow 2 2) ) ; Power

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;MATH 3

(println "Ceil : " (Math/ceil 4.5) ) ; Round up

(println "Floor : " (Math/floor 4.5) ) ; Round down

(println "Max : " (Math/max 1 5) ) ; Show max value

(println "Min : " (Math/min 1 5) ) ; Show least value

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;MATH 4

(println "Add collection :" (reduce + [1 2 3])); add a collection

(println "Subtract collection :" (reduce - [1 2 3])); subtract a collection

(println (format "Pi : %.3f" (Math/PI)))

(println (format "e^n : %.3f" (Math/exp 1)) ) ; e to the power of 1

(println "ln: :" (Math/log 2.71828)) ; Natural logarithm

(println "log10 :" (Math/log10 100) ) ; Base 10 log

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;MATH 5

(println (format "Hypot : %.3f" (Math/hypot 2 2)) ) ; sqrt(x^2 + y^2)

;Trig Functions

(println "cos :" (Math/cos 0))

(println "sin :" (Math/sin 0))

(println "tan :" (Math/tan 0))

;Inverse Trig Functions

(println "acos :" (Math/acos 0))

(println "asin :" (Math/asin 0))

(println "atan :" (Math/atan 0))

;Hyperbolic Functions

(println "cosh :" (Math/cosh 0))

(println "sinh :" (Math/sinh 0))

(println "tanh :" (Math/tanh 0))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; LISTS

(list "Cat" 1 3.4 true ) ; Stores list of values w/ multiple data types

(first (list 1 2 3) ) ; Get 1st Value

(rest (list 1 2 3) ) ; Get rest of values

(nth (list 1 2 3) 0 ) ; Get defined index

(list\* 1 2 [3 4] ) ; Add values to the right

(cons 3 (list 1 2) ) ; Add value to the left

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; SETS

(set '(1 1 2) ) ; List of unique values

(conj (set '(3 2) ) 4 ) ; Append a value

(contains? (set '(3 2) ) 3 ) ; Is value in set

(disj (set '(3 2) ) 2 ) ; Remove value from set

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; VECTORS

(vector 1 "Cat" )

(get (vector 3 2) 1 ) ; Get index

(conj (vector 1 2) 3 ) ; Append element

(pop (vector 3 2 1) ) ; Remove last item

(subvec (vector 1 2 3 4 4 5 1) 0 5) ; Return vector from one point to another

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; MAPS

(hash-map "Name" "Carl" "Age" 24 "BT" "O") ; Collection of key/value pairs

(sorted-map 3 4 2 "Bananas" 1 "Cat") ; Sorted lists sort based on keys

(get (hash-map "Name" "Cat") "Name") ; Return value mapped to key

(find (hash-map "Name" "Cat" "Age" 5) "Name") ; Get key/value for the key

(contains? (hash-map "Name" "Cat") "Age") ; Does map contain a key

(keys (hash-map "Name" "Cat" "Age" 5)) ; Get list of keys

(vals (hash-map "Name" "Cat" "Age" 5)) ; Get values

(merge-with + (hash-map "Name" "Cat") (hash-map "Age" 5)) ; Merge maps

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;FUNCTIONS - example 1

(defn hi [name] (println "Hi, " name))

(defn hi-many [& names] (map hi names))

(hi "man")

(hi-many "Timmy" "Jimmy" "Kimmy")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;FUNCTIONS - example 2

(defn add2 [x y] (+ x y))

(defn add2or3 ([x y z] (+ x y z)) ([x y] (+ x y)) )

(add2 34 53)

(add2or3 2 4)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; RELATIONAL OPERATORS

(println "x=y: " (= 4 5) )

(println "x <> y: " (not= 4 5) )

(println "x<y: " (< 4 5) )

(println "x<=y: " (<= 4 5) )

(println "x>y: " (> 4 5) )

(println "x>=y: " (>= 4 5) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;Logical Operators

(and (< 3 4) (< 5 6)) ;if both arguments are true, then true

(or (< 3 4) (> 5 6)) ;if one argument is true, then true

(not (< 3 4) ) ;if argument true, then flase

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Conditionals

(if (= 0 0) (println "true") (println "false"))

(if (= 0 0)

(do (println "zero is zero") (println "Amazing!"))

(println "false")

)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; IF THEN ELSE

(defn size [number] ;Function Name, Input Name

;Code Block:

(if (>= number 10) (println "Large") (println "Regular") ))

;logical test value if true value if false

(size 3)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;When

(when (> 6 4) (println "Yes") (println "false"))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; COND

;; Check for multiple conditions

;<3 is small

;<5 is medium

;<8 is large

;>=8 is extra large

(defn manyif [n] ; name and input

(cond ; add multiple conditions

(< n 3) ; logical test <-- 1

"small" ; value if true

(< n 5) ; logical test <-- 2

"medium" ; value if true

(< n 8) ; logical test <-- 3

"large" ; value if true

:else "extra large") ;value if false

)

(manyif 7)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; DOTIMES LOOP

; Execute a statement a set number of times

; i is incremented by one

(dotimes [i 8] ;i will be 0,1,2,3,4,5,6,7

;Code Block:

(println (\* i 2) ) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; LOOP LOOP

; Loop through values using recur to change a value until a condition is no longer true

(loop [i 0] ;starting value for i

;Code block

(when (<= i 5) ; logical test

(println "i = "i) ; value if true

(recur (+ i 1)) ; recur tail that sets a new value for i

))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; DOSEQ LOOP

; Iterates through a sequence

(defn print-list [& values] ; Holds list passed

(doseq [x values] (println x))) ; As you cycle through the list store each item in x

(print-list 1 2 3 "Example")