

(not x)

## Student Cheatsheet 1

```
Creating a Vector
                                                    Creating a Map
(vector 1 2 3)
                                                    (hash-map :a 1 :b 2)
;=> [1 2 3]
                                                    ;=> {:a 1, :b 2}
[1 2 3 4]
;=> [1 2 3 4]
                                                    {:a 1 :b "two"}
                                                    ;=> {:a 1, b "two"}
Vector Examples
(conj [5 10] 15)
                                                    Map examples
;=> [5 10 15]
                                                    (get {:first "Sally" :last "Brown"} :first)
                                                    ;=> "Sally"
(count [5 10 15])
                                                    (get {:first "Sally"} :last :MISS)
;=> 3
                                                    :=> :MISS
                                                    (assoc {:first "Sally"} :last "Brown")
(nth [5 10 15] 1)
                                                    ;=> {:first "Sally", :last "Brown"}
:=> 10
                                                    (dissoc {:first "Sally" :last "Brown"} :last)
(first [5 10 15])
                                                    ;=> {:first "Sally"}
;=> 5
                                                    (merge {:first "Sally"} {:last "Brown"})
                                                    ;=> {:first "Sally", :last "Brown"}
Defining a value
(def name "Sally")
                                                    (count {:first "Sally" :last "Brown"})
                                                    ;=> 2
Defining a function
(defn function-name
                                                    (keys {:first "Sally" :last "Brown"})
  "description of function, optional"
                                                    ;=> (:first :last)
  [param1 param2]
                                                    (vals {:first "Sally" :last "Brown"})
  (function-body))
                                                    ;=> ("Sally" "Brown")
Flow Control
(if conditional-expression
                                                    l et
  expression-to-evaluate-when-true
  expression-to-evaluate-when-false)
                                                    (let [first-name user)
                                                          message (str "Hello, " first-name "!")]
Logic Functions
                                                      (println message))
(= \times 4)
(> x \ 4) (>= x \ 4)
(< \times 4) (<= \times 4)
(and x y)
(or x y)
```