```clojure
;; make-point - Make a point that consists of an x and y component
(defn make-point [x y]
    (list x y))

;; x-coord - A selector for a point object's x coordinate
(defn x-coord [point]
    (first point))

;; y-coord - A selector for a Point object's y coordinate
(defn y-coord [point]
    (second point))

;; random-point - takes no arguments; returns a randomly generated Point with coordiates in the interval [0, 1]. See rand.
(defn random-point []
    (make-point (rand) (rand)))

;; throw-darts - Takes single argument n representing number of darts to throw. Generates a list of points of length n. Use "repeatedly" to generate an infinite sequence of random-point calls; use "take" on the result to collect only the first "n" results. (You can improve this by making the "throw-darts" function into "get-sample". Then replace the repeated function with an anonymous function)
(defn throw-darts [n]
    (take n (repeatedly #(random-point))))

;; Safe method of performing a squareroot (no imaginary numbers)
(defn sqrt [x]
  (if (> x 0)
      (Math/sqrt x)
      0))

;; is-hit? - takes a point representing a dart and returns if the dart hit the quarter circle. (Hint: calculate distance from the origin to the dart's coordinates, and decide if that distance means the dart lands inside or outside the quarter circle. Use selectors for Point's attributes)
(defn is-hit [point]
    ; a^2 + b^2 = c^2
    ; if c is less than 1, true
    (if (< (sqrt (+ (* (x-coord point) (x-coord point)) (* (y-coord point) (y-coord point)))) 1)
        1
        0
    )
)

;; count-hits - takes single argument n representing how many darts to "throw" and counts the number of darts that hit correctly.
(defn count-hits [n]
    (reduce
      +
      (map #(is-hit %) (throw-darts n))
    )
)

;; estimate-pi - takes single parameter n, uses "count-hits" and correct math to estimate the value of pi
(defn estimate-pi [n]
    (float (* (/ (count-hits n) n) 4))
)
```