Homework 02

● Graded

Student

Giancarlos Marte

Total Points

94 / 94 pts

Autograder Score

89.0 / 89.0

Passed Tests

Exercise 1. pair (0/0)
Exercise 1. pair-left (1/1)
Exercise 1. pair-right (1/1)
Exercise 1.a. pair-set-right (3/3)
Exercise 1.b. pair-set-left (3/3)
Exercise 1.c. pair-swap (3/3)
Exercise 1.d. pair-add (4/4)
Exercise 2.b. first-name (4/4)
Exercise 2.c. last-name (4/4)
Exercise 2.d. full name (4/4)
Exercise 2.e. initials (5/5)
Exercise 3. max-from (5/5)
Exercise 4. min-from (5/5)
Exercise 6. count (11/11)
Exercise 7. sum (12/12)
Exercise 8. occurrences (12/12)
Exercise 9. norm (12/12)

Question 2

Adjustments

0 / 0 pts

✔ **– 0 pts** No adjustments

– 5 pts (2) Representation is not a function (`lambda`)

– 0 pts (2) Only `first-name` and `last-name` need to be "methods"

– 1 pt (9) should use `(map sqr _)` and `sum` (from exercise 7)

– 5 pts Late penalty

Question 3

Problem 5

5 / 5 pts

✔ **– 0 pts** Correct

– 5 pts Missing

– 4 pts Incomplete or doesn't work

– 2 pts Did not redefine min-from, max-from

Autograder Results

Exercise 1. pair (0/0)

Exercise 1. pair-left (1/1)

Exercise 1. pair-right (1/1)

Exercise 1.a. pair-set-right (3/3)

Exercise 1.b. pair-set-left (3/3)

Exercise 1.c. pair-swap (3/3)

Exercise 1.d. pair-add (4/4)

Exercise 2.b. first-name (4/4)

Exercise 2.c. last-name (4/4)

Exercise 2.d. full name (4/4)

Exercise 2.e. initials (5/5)

Exercise 3. max-from (5/5)

Exercise 4. min-from (5/5)

Exercise 6. count (11/11)

Exercise 7. sum (12/12)

Exercise 8. occurrences (12/12)

Exercise 9. norm (12/12)

Submitted Files

hw2.rkt Download

```
#lang racket
(require rackunit)
#|
         ########################################################
         ###  PLEASE DO NOT DISTRIBUTE SOLUTIONS PUBLICLY  ###
         ########################################################
|#
(provide (all-defined-out))
;; ^^^^^ DO NOT CHANGE ANY CODE ABOVE THIS LINE ^^^^^

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Exercise 1

;; point, point?, point-left, point-right
;; pair: Symbol Symbol
(struct pair (left right) #:transparent)

#|(define pair 'delete-me-after-defining-your-struct)
(define pair-left 'delete-me-after-defining-your-struct)
(define pair-right 'delete-me-after-defining-your-struct)
|#

;; Exercise 1.a
;; pair-set-left: Pair Symbol -> Pair
(define (pair-set-left p l)
  (match p
    [(pair lh rh)
     (pair l rh)]))

;; Exercise 1.b
;; pair-set-right: Pair Symbol -> Pair
(define (pair-set-right p r)
  (match p
    [(pair lh rh)
     (pair lh r)]))

;; Exercise 1.c
;; pair-swap: Pair -> Pair
(define (pair-swap p)
  (match p
    [(pair lh rh)
     (pair rh lh)]))

;; Exercise 1.d
;; You can only use match* one time. You cannot use match.
;; pair-add: Pair Pair -> Pair
```

```
(define (pair-add p1 p2)
  (match* (p1 p2)
    [((pair lh rh) (pair lh2 rh2)) (pair (+ lh lh2) (+ rh rh2))]))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Exercise 2.a
;; name: String String -> (X -> String)
(define (name first last)
  (lambda (m) (m first last)))

;; Exercise 2.b
;; first-name: Name -> String
(define (first-name p)
  (p (lambda (f l) f)))

;; Exercise 2.c
;; last-name: Name -> String
(define (last-name p)
  (p (lambda (f l) l)))

;; Exercise 2.d
;; full-name: Name -> String
(define (full-name p)
  (p (lambda (f l)
       (string-append (string-append f " ") l))))

;; Exercise 2.e
;; initials: Name -> String
(define (initials p)
  (p (lambda (f l)
       (string-append
         (substring f 0 1) (substring l 0 1)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
#|
;; Exercise 3
;; max-from: Real (Listof Real) -> Real
(define (max-from n l)
  (match l
    ['() n]
    [(cons f r) (max-from (max n f) r)]))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Exercise 4
;; min-from Real (Listof Real) -> Real
(define (min-from n l)
  (match l
    ['() n]
    [(cons f r) (min-from (min n f) r)]))
```

```
|#

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Exercise 5: revisit Exercise 3 and Exercise 4

;; Exercise 3
;; max-from: Real (Listof Real) -> Real
(define (max-from n l)
  (from n l max-from max))

;; Exercise 4
;; min-from Real (Listof Real) -> Real
(define (min-from n l)
  (from n l min-from min))

;; auxilary
;; from: Real (Listof Real) (Real (Listof Real) -> Real) (Real Real -> Real) -> Real
(define (from n l mainf innerf)
  (match l
    ['() n]
    [(cons f r) (mainf (innerf n f) r)]))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Exercise 6
;; count: (Listof Real) -> Real
(define (count l)
  (match l
    ['() 0]
    [(cons f r) (+ 1 (count r))]))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Exercise 7
;; sum: (Listof Real) -> Real
(define (sum l)
  (match l
    ['() 0]
    [(cons f r) (+ f (sum r))]))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Exercise 8
;; occurrences: Symbol (Listof symbol) -> Real
(define (occurrences x l)
  (match l
    ['() 0]
    [(cons f r)
     (cond
       [(equal? f x) (+ 1 (occurrences x r))]
       [else (+ 0 (occurrences x r))])]))

```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Exercise 9
;; norm: (Listof Real) -> Real
(define (norm l)
  (define addSqr (sum(map sqr l)))
  (sqrt addSqr))
```