CSC324: OOP and Macros

1. Here is a `Point` class, which includes a proposed implementation of `same-distance`. The method `same-distance` is designed to return true iff the current point and `other-point` are the same distance from the origin.

```
#lang racket
(define (Point x y)
  (lambda (msg)
    (cond [(equal? msg 'x) x]
          [(equal? msg 'y) y]
          [(equal? msg 'from-origin)
           (lambda ()
             (sqrt (+ (* x x) (* y y))))]
          [(equal? msg 'same-distance)
           (lambda (other-point)
             (= ((other-point 'from-origin)) ('from-origin)))]
          [else "Unrecognized message"])))
```

   Does `same-distance` work? If so, explain how. If not, what is the problem?

2. Write the resulting function after the `mystery` macro is expanded. If an error occurs, write "Error".

   Note: do not evaluate any of the resulting expressions! This question is a good check to make sure you understand the difference between the macro expansion and evaluation phases of the Racket interpreter.

```
#lang racket
(define-syntax mystery
  (syntax-rules ()
    [(mystery x) x]
    [(mystery x y) (+ x (- x 1))]
    [(mystery x y ...) (+ x y ...)]))

(mystery 4)




(mystery 4 5)




(mystery 4 5 9)
```

CSC324: Macro Ellipses

3. Which of the following expressions match the macro pattern `(my-macro <attr> ...)`?

   (A) `(my-macro)`
   (B) `(my-macro 2)`
   (C) `(my-macro 2 3)`
   (D) `(my-macro (+ 2 3))`
   (E) `(my-macro 1 (2 3))`

4. Which of the following expressions match the macro pattern `(my-macro (<x> <y>) ...)`?

   (A) `(my-macro)`
   (B) `(my-macro 2 3)`
   (C) `(my-macro (2 3))`
   (D) `(my-macro (+ 2))`
   (E) `(my-macro (2 3) (4 5))`
   (F) `(my-macro (2 3) ((+ 8 9) 5))`

5. Which of the following expressions match the macro pattern `(my-macro (<a> ...) ...)`?

   (A) `(my-macro)`
   (B) `(my-macro 2 3)`
   (C) `(my-macro (2 3))`
   (D) `(my-macro (+ 2 3 4))`
   (E) `(my-macro (1 2) (3 4 5 6) (7 8))`
   (F) `(my-macro (1 2) (3 4 5 6) (7 8) ())`
   (G) `(my-macro (1 2) (3 4 5 6) 7 (8))`

6. Consider the following macro:

```
(define-syntax my-macro
  (syntax-rules ()
    [(my-macro (<x> <y>) ...) (list '1st (- <x> <y>) ...)]
    [(my-macro (<a> ...) ...) (list '2nd (+ <a> ...) ...)]
    [(my-macro <e> ...)       (list '3rd)]))
```

   What do the following macro expressions expand to?

   (A) `(my-macro)`

```
(list '1st)
```

   (B) `(my-macro 2 3)`

```
(list '3rd)
```

   (C) `(my-macro (3 2) (7 5))`

```
(list '1st (- 3 2) (- 7 5))
```

   (D) `(my-macro (2 3) (4 5 6) (2 2 6))`

```
(list '2nd (+ 2 3) (+ 4 5 6) (+ 2 2 6))
```

   (E) `(my-macro (+ 2) (- 6) (+ +))`

```
(list '1st (- + 2) (- - 6) (- + +))
```