

Lab 2: Recursion in Racket

Seven Lewis

4/12/24

Part 1

1

Run this function on the following lists:

```
(define (mystery L)
  (if (null? L)
      L
      (append (mystery (cdr L))
               (list (car L))
              )
  )
)

(mystery '(1 2 3))           ; '(3 2 1)
(mystery '((1 2) (3 4) 5 6)) ; '(6 5 (3 4) (1 2))
```

*; This function reverses a given list. It does this by recursing
; down the list until we reach the 'null' pointer for the list
; that was returned by the final 'cdr'. Then, we go back up the
; list grabbing the value in current node with 'car' and appending
; it to the end of our growing reverse list. Eventually we return
; the 'mystery' procedure when the outermost append returns.*

As you may have noticed, there is no return statement here. Explain how the return value is determined in the above function.

*; A lot of Racket abides by pure functional programming including the
; snippet above. One of the main aspects of this design is that many
; things are just nested expressions that will be evaluated at runtime.
; After evaluation, the expression will simply be in its simplest form,
; which for our purposes is what we call the 'return' value for our
; function call. There is no need to have a 'return' statement because
; it's a natural simplification of the expression.*

*; This is quite different from how a procedural programming language
; might handle things as ideas like "expressions" are less common
; in languages like C and Java. Functions and methods in these
; languages generally need to have explicit declaration of a particular
; return value to compensate for this.*

```

; As for the particular example above, you can think about it like this:
; Every time you see some expression like '(mystery '(1 2 3))', it can
; simply be replaced with:
; (if (null? '(1 2 3))
;     '(1 2 3)
;     (append (mystery (cdr '(1 2 3)))
;              (list (car '(1 2 3)))))
; )
; )

; where every instance of the first argument in 'define' is replaced
; with the second. (Obviously because this function is recursive,
; you actually need to replace the inner '(mystery (cdr '(1 2 3)))'
; as well.

```

3

Modify the program from Question 1 to the follows:

```

(define (mystery L)
  (if (null? L)
      L
      (begin
        (displayln L)
        (append
         (mystery (cdr L))
         (list (car L)))
        )
      )
  )
)

```

What does `begin` do? What does `displayln` do?

```

; 'begin' simply evaluates each expression passed to it, only
; returning the final expression's return value.

; 'displayln' prints to the console the data passed to it.

```

Part 2

Please view the associated 'lab2.rkt' file.