Assignment #4

Trevor Strickland, CSC344

October 6, 2022

Learning Abstract: This assignment dealt with list processing in Racket (and by extension, Lisp) as it's main topic, which includes legacy Lisp commands like car, cdr, cons, eval, etc. It also covered working with lambda functions and user input, with a hint of recursion.

Task 1: Lambdas

Demo:

```
> ((lambda (x) (list x (+ x 1) (+ x 2))) 5)
'(5 6 7)
> ((lambda (x) (list x (+ x 1) (+ x 2))) 0)
'(0 1 2)
> ((lambda (x) (list x (+ x 1) (+ x 2))) 108)
'(108 109 110)
> ((lambda (x y z) (list z y x)) 'red 'yellow 'blue)
'(blue yellow red)
> ((lambda (x y z) (list z y x)) 10 20 30)
'(30 20 10)
> ((lambda (x y z) (list z y x)) "Professor Plum" "Colonel Mustard" "Miss Scarlet")
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
```

```
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
3
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
4
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
3
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
4
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
5
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
3
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
4
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
4
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
4
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
3
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
4
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
5
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
3
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
3
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 3 5)
3
```

```
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
14
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
13
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
17
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
14
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
16
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
13
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
11
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
15
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
11
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
13
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
15
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
15
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
17
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
17
> ((lambda (x y) (+ (random (- (+ y 1) x)) x)) 11 17)
14
```

Task 2: List Processing in Lisp

Demo:

```
> (define colors '(red blue yellow orange))
> colors
'(red blue yellow orange)
> 'colors
'colors
> (quote colors)
'colors
> (car colors)
'red
> (cdr colors)
'(blue yellow orange)
> (car (cdr colors))
'blue
> (cadr colors)
'blue
> (cddr colors)
'(yellow orange)
> (first colors)
'red
> (second colors)
'blue
> (third colors)
'yellow
> (list-ref colors 2)
'yellow
> (define key-of-c '(c d e))
> (define key-of-g '(g a b))
> (cons key-of-c key-of-g)
'((c d e) g a b)
> (list key-of-c key-of-g)
'((c d e) (g a b))
> (append key-of-c key-of-g)
'(c d e g a b)
> (define pitches '(do re mi fa so la ti do))

      > (define animals '(alligator pussycat chimpanzee fungus))
      > (car (cdr (cdr (cdr animals))))
      'fungus
      > (cadddr pitches)
      'fa
      > (list-ref pitches 3)
      'fa
      > (define a 'alligator)
      > (define b 'pussycat)
      > (define c 'chimpanzee)
      > (cons a (cons b (cons c '())))
      '(alligator pussycat chimpanzee)
      > (list a b c)
      '(alligator pussycat chimpanzee)
      > (define x '(1 one))
      > (define y '(2 two))
      > (cons (car x) (cons (car (cdr x)) y))
      '(1 one 2 two)
      > (append x y)
      '(1 one 2 two)
```

Task 3: Little Color Interpreter

Code:

sampler.rkt

```
#lang racket

(define (sampler)
  (display "(?): ")
  (define the-list (read))
  (define the-element
    (list-ref the-list (random (length the-list)))
  )
  (display the-element) (display "\n")
  (sampler)
)
```

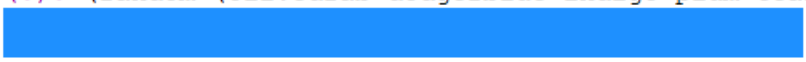
color_thing.rkt

```
#lang racket
(require 2htdp/image)

(define (rect color)
  (rectangle 400 25 "solid" color)
)

(define (rect-all colors n)
  (display (rect (list-ref colors n)))
  (cond
    ((< n (- (length colors) 1))
      (rect-all colors (+ n 1))
    )
  )
)

(define (color-thing)
  (display "(?): ")
  (define the-list (read))
  (define command (list-ref the-list 0))
  (define colors (list-ref the-list 1))
  (cond
    ((equal? command 'random)
     (display (rect (list-ref colors (random (length colors)))))
     )
    ((equal? command 'all)
     (rect-all colors 0)
     )
    (else
     (display (rect (list-ref colors (- command 1))))
    ))
  (display "\n")
```

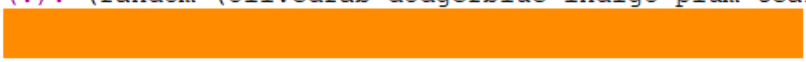
```
  (color-thing)
)
```

Demo:

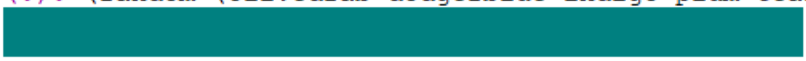
```
> (color-thing)
(?): (random (olivedrab dodgerblue indigo plum teal darkorange))
```



```
(?): (random (olivedrab dodgerblue indigo plum teal darkorange))
```

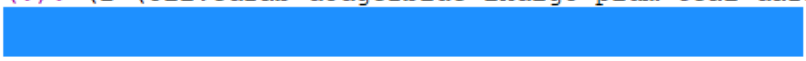


```
(?): (random (olivedrab dodgerblue indigo plum teal darkorange))
```

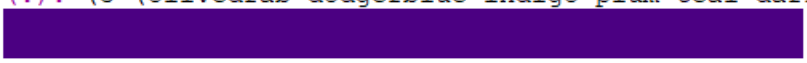


```
(?): (all (olivedrab dodgerblue indigo plum teal darkorange))
```

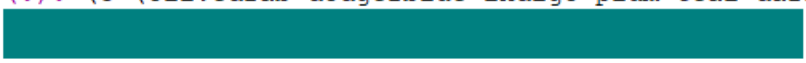
```
(?): (2 (olivedrab dodgerblue indigo plum teal darkorange))
```

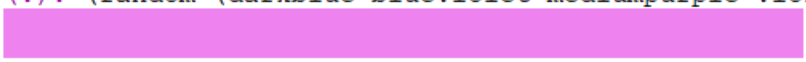


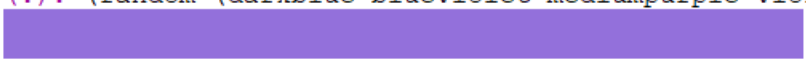
```
(?): (3 (olivedrab dodgerblue indigo plum teal darkorange))
```

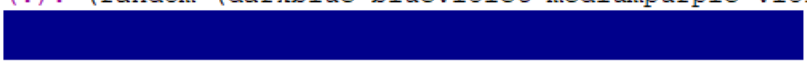


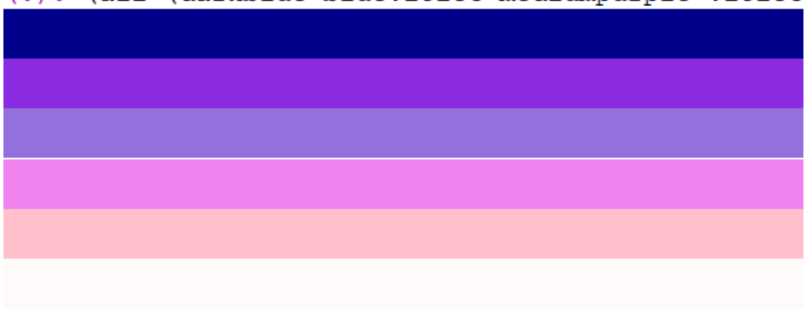
```
(?): (5 (olivedrab dodgerblue indigo plum teal darkorange))
```



```
(?): (random (darkblue blueviolet mediumpurple violet pink snow))
```


```
(?): (random (darkblue blueviolet mediumpurple violet pink snow))
```


```
(?): (random (darkblue blueviolet mediumpurple violet pink snow))
```


```
(?): (all (darkblue blueviolet mediumpurple violet pink snow))
```


```
(?): (1 (darkblue blueviolet mediumpurple violet pink snow))
```


```
(?): (5
 (darkblue blueviolet mediumpurple violet pink
snow))
```


```
(?): (4 (darkblue blueviolet mediumpurple violet pink snow))
```


Task 4: Two Card Poker

Code:

```
classifier.rkt
;(contains all relevant code from classifier_ur.rkt and cards.rkt)

#lang racket

(define (ranks rank)
  (list
    (list rank 'C)
    (list rank 'D)
    (list rank 'H)
    (list rank 'S)
  )
)

(define (deck)
  (append
    (ranks 2)
    (ranks 3)
    (ranks 4)
    (ranks 5)
    (ranks 6)
    (ranks 7)
    (ranks 8)
    (ranks 9)
    (ranks 'X)
    (ranks 'J)
    (ranks 'Q)
    (ranks 'K)
    (ranks 'A)
  )
)

(define (pick-a-card)
  (define cards (deck))
  (list-ref cards (random (length cards)))
)

(define (pick-two-cards)
  (define cards (deck))
  (define a (list-ref cards (random (length cards))))
  (define b (list-ref cards (random (length cards))))
  (cond ((equal? a b) (pick-two-cards)))
  (list a b)
)

(define (int-rank rank)
  (cond
    ((equal? rank 2) 1)
```

```
    ((equal? rank 3) 2)
    ((equal? rank 4) 3)
    ((equal? rank 5) 4)
    ((equal? rank 6) 5)
    ((equal? rank 7) 6)
    ((equal? rank 8) 7)
    ((equal? rank 9) 8)
    ((equal? rank 'X) 9)
    ((equal? rank 'J) 10)
    ((equal? rank 'Q) 11)
    ((equal? rank 'K) 12)
    ((equal? rank 'A) 13)
  )
)


(define (higher-rank a b)
  (define ra (int-rank (list-ref a 0)))
  (define rb (int-rank (list-ref b 0)))

  (cond
    ((> ra rb) (list-ref a 0))
    (else (list-ref b 0))
  )
)

(define (classify-two-cards-ur list)
  (define a (list-ref list 0))
  (define b (list-ref list 1))
  (define ra (int-rank (list-ref a 0)))
  (define rb (int-rank (list-ref b 0)))

  (display list) (display ": ")

  (define higher (higher-rank a b))
  (display higher) (display " high")

  (cond
    ((or (= ra (- 1 rb)) (= rb (- 1 ra))) (display " straight"))
    ((equal? (list-ref a 1) (list-ref b 1)) (display " flush"))
    ((= ra rb) (display " pair"))
  )

  (display "\n")
)


(define (show card)
  (display (rank card))
  (display (suit card))
)

(define (rank card)
```

```
  (car card)
)

(define (suit card)
  (cadr card)
)

(define (red? card)
  (or
    (equal? (suit card) 'D)
    (equal? (suit card) 'H)
  )
)

(define (black? card)
  (not (red? card))
)

(define (aces? card1 card2)
  (and
    (equal? (rank card1) 'A)
    (equal? (rank card2) 'A)
  )
)
```

Demo:

```
> (define c1 '(7 C))
> (define c2 '(Q H))
> c1
'(7 C)
> c2
'(Q H)
> (rank c1)
7
> (suit c1)]
'C
```

```
> (rank c2)
'Q
> (suit c2)
'H
> (red? c1)
#f
> (red? c2)
#t
> (black? c1)
#t
> (black? c2)
#f
> (aces? '(A C) '(A S))
#t
> (aces? '(K S) '(A C))
#f
> (ranks 4)
'((4 C) (4 D) (4 H) (4 S))
> (ranks 'K)
'((K C) (K D) (K H) (K S))
> (length (deck))
52
> (display (deck))
((2 C) (2 D) (2 H) (2 S) (3 C) (3 D) (3 H) (3 S) (4 C) (4 D) (4 H) (4 S) (5 C) (5 D) (5 H)
(5 S) (6 C) (6 D) (6 H) (6 S) (7 C) (7 D) (7 H) (7 S) (8 C) (8 D) (8 H) (8 S) (9 C) (9 D)
(9 H) (9 S) (X C) (X D) (X H) (X S) (J C) (J D) (J H) (J S) (Q C) (Q D) (Q H) (Q S) (K C)
(K D) (K H) (K S) (A C) (A D) (A H) (A S))
```

```
> (pick-a-card)
'(9 H)
> (pick-a-card)
'(3 S)
> (pick-a-card)
'(Q S)
> (pick-a-card)
'(9 S)
> (pick-a-card)
'(K S)
> (pick-a-card)
'(2 S)
> (pick-a-card)
'(5 D)
> (pick-a-card)
'(4 D)
> (pick-a-card)
'(6 H)
```

```
> (pick-two-cards)
'((K S) (A D))
> (pick-two-cards)
'((A C) (3 D))
> (pick-two-cards)
'((Q H) (6 H))
> (pick-two-cards)
'((X C) (4 H))
> (pick-two-cards)
'((X D) (9 C))
```

```
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(4 C) '(Q H))
<'Q
'Q
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(2 C) '(A C))
<'A
'A
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(7 D) '(7 H))
<7
7
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(K D) '(2 C))
<'K
'K
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(9 C) '(6 H))
<9
9
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(J H) '(2 D))
<'J
'J
```

```
> (classify-two-cards-ur (pick-two-cards))
((Q S) (2 S)): Q high flush
> (classify-two-cards-ur (pick-two-cards))
((5 H) (Q S)): Q high
> (classify-two-cards-ur (pick-two-cards))
((9 S) (2 S)): 9 high flush
> (classify-two-cards-ur (pick-two-cards))
((8 D) (J D)): J high flush
> (classify-two-cards-ur (pick-two-cards))
((K H) (9 D)): K high
> (classify-two-cards-ur (pick-two-cards))
((3 C) (X D)): X high
> (classify-two-cards-ur (pick-two-cards))
((9 H) (7 C)): 9 high
> (classify-two-cards-ur (pick-two-cards))
((Q C) (3 S)): Q high
> (classify-two-cards-ur (pick-two-cards))
((5 S) (7 C)): 7 high
> (classify-two-cards-ur (pick-two-cards))
((6 C) (8 C)): 8 high flush
> (classify-two-cards-ur (pick-two-cards))
((J D) (7 S)): J high
> (classify-two-cards-ur (pick-two-cards))
((4 C) (7 H)): 7 high
> (classify-two-cards-ur (pick-two-cards))
((J C) (5 D)): J high
> (classify-two-cards-ur (pick-two-cards))
((8 S) (2 D)): 8 high
> (classify-two-cards-ur (pick-two-cards))
((6 C) (7 H)): 7 high
> (classify-two-cards-ur (pick-two-cards))
((Q S) (Q H)): Q high pair
> (classify-two-cards-ur (pick-two-cards))
((5 H) (8 C)): 8 high
> (classify-two-cards-ur (pick-two-cards))
((4 C) (X C)): X high flush
> (classify-two-cards-ur (pick-two-cards))
((8 H) (6 C)): 8 high
> (classify-two-cards-ur (pick-two-cards))
((K H) (X D)): K high
```

```
> (classify-two-cards (pick-two-cards))
((Q D) (2 D)): queen high flush
> (classify-two-cards (pick-two-cards))
((6 S) (X H)): five high
> (classify-two-cards (pick-two-cards))
((7 C) (6 C)): seven high flush
> (classify-two-cards (pick-two-cards))
((2 D) (X S)): five high
> (classify-two-cards (pick-two-cards))
((9 D) (2 H)): nine high
> (classify-two-cards (pick-two-cards))
((J D) (A H)): ace high
> (classify-two-cards (pick-two-cards))
((6 D) (X D)): five high flush
> (classify-two-cards (pick-two-cards))
((4 D) (4 C)): four high pair
> (classify-two-cards (pick-two-cards))
((K D) (8 D)): king high flush
> (classify-two-cards (pick-two-cards))
((8 D) (K H)): king high
> (classify-two-cards (pick-two-cards))
((5 H) (4 H)): five high flush
> (classify-two-cards (pick-two-cards))
((8 D) (9 C)): nine high
> (classify-two-cards (pick-two-cards))
((X H) (6 H)): five high flush
> (classify-two-cards (pick-two-cards))
((2 C) (A S)): ace high
> (classify-two-cards (pick-two-cards))
((X H) (9 D)): five high
> (classify-two-cards (pick-two-cards))
((X S) (X H)): five high pair
> (classify-two-cards (pick-two-cards))
((4 D) (8 S)): eight high
> (classify-two-cards (pick-two-cards))
((Q D) (3 D)): queen high flush
> (classify-two-cards (pick-two-cards))
((K S) (3 C)): king high
> (classify-two-cards (pick-two-cards))
((K S) (K C)): king high pair
```