Racket Programming Assignment# 3: Lambda and Basic Lisp

Learning Abstract

This assignment features programs in Lisp. The first two tasks are highly constrained. One of these pertains to lambda functions, and the other to basic list processing operations in Lisp. The second two tasks, which are considerably more involved extend programs that are presented in Lesson 6 "Basic Lisp Programming".

Task 1a: Three Ascending Integers

```
> (define (asc n)
    (cons n ( cons (+ n 1) ( cons (+ n 2) '() ) ) )
    )
> (asc 5)
'(5 6 7)
> (asc 0)
'(0 1 2)
> (asc 108)
'(108 109 110)
```

Task 1b: Make list in reverse order

```
> (define (mlr x y z)
    (cons z ( cons y ( cons x '() ) ) )
    )
> (mlr 'red 'yellow 'blue)
'(blue yellow red)
> (mlr 10 20 30)
'(30 20 10)
> (mlr "Professor Plum" "Colonel Mustard" "Miss Scarlet")
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
```

Task 1c: Random Number Generator

```
> (define (rn x y)
    (list-ref (cons x(cons y '() ) ) (random 2) )
    )
> (rn 3 5)
3
> (rn 3 5)
3
> (rn 3 5)
5
> (rn 3 5)
3
> (rn 3 5)
5
> (rn 3 5)
5
> (rn 3 5)
3
> (rn 3 5)
5
> (rn 3 5)
5
> (rn 3 5)
3
> (rn 11 17)
11
> (rn 11 17)
17
> (rn 11 17)
17
> (rn 11 17)
11
> (rn 11 17)
11
> (rn 11 17)
11
> (rn 11 17)
11
> (rn 11 17)
11
> (rn 11 17)
11
> (rn 11 17)
11
> (rn 11 17)
11
> 
```

Task 2: List Processing Referencers and Constructors

Definitions:

```
1  #lang racket
2  (define colors '(red blue yellow orange))
3  (define key-of-c '(c d e))
4  (define key-of-g '(g a b))
5  (define pitches '(do re mi fa so la ti))
6  (define a 'alligator)
7  (define b 'pussycat)
8  (define c 'chimpanzee)
9  (define x '(1 one))
10 (define y '(2 two))
11
```

Demos:

```
> colors
'(red blue yellow orange)
> 'colors
'colors
> (quote colors)
'colors
> (car colors)
'red
> (cdr colors)
'(blue yellow orange)
> (car (cdr colors) )
'blue
> (cdr (cdr colors) )
'(yellow orange)
> (cadr colors)
'blue
> (cddr colors)
'(yellow orange)
> (first colors)
'red
> (second colors)
'blue
> (third colors)
'yellow
> (list-ref colors 2)
'yellow
>
```

```
> (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)
>
```


```
> (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)
>
```


```
> (car ( cdr ( cdr ( cdr animals))))
```

```
 animals: undefined;
 cannot reference an identifier before its definition
> (car ( cdr ( cdr ( cdr pitches))))
'fa
> (cadddr pitches)
'fa
> (list-ref pitches 3)
'fa
>
```

```
> (cons a (cons b( cons c '())))
'(alligator pussycat chimpanzee)
> (list a b c)
'(alligator pussycat chimpanzee)
>
```

```
> (cons (car x) (cons (car(cdr x)) y))
'(1 one 2 two)
> (append x y)
'(1 one 2 two)
>
```

Task 3a: Establishing the Sampler code

Code:

```
#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 )
)
```

Demo:

```
> (sampler)
(?): (lion cat dog rat monkey elephant)
rat
(?): (lion cat dog rat monkey elephant)
monkey
(?): (lion cat dog rat monkey elephant)
elephant
(?): (lion cat dog rat monkey elephant)
dog
(?): (lion cat dog rat monkey elephant)
rat
(?): (lion cat dog rat monkey elephant)
cat
(?): (lion cat dog rat monkey elephant)
lion
(?): (lion cat dog rat monkey elephant)
lion
(?): (aet ate eat eta tae tea)
eat
(?): (aet ate eat eta tae tea)
eat
(?): (aet ate eat eta tae tea)
ate
(?): (aet ate eat eta tae tea)
ate
(?): ( 0 1 2 3 4 5 6 7 8 9 )
9
(?): ( 0 1 2 3 4 5 6 7 8 9 )
2
(?): ( 0 1 2 3 4 5 6 7 8 9 )
8
(?): ( 0 1 2 3 4 5 6 7 8 9 )
7
(?): ( 0 1 2 3 4 5 6 7 8 9 )
4
(?): ( 0 1 2 3 4 5 6 7 8 9 )
3
(?): ( 0 1 2 3 4 5 6 7 8 9 )
4
(?): ( 0 1 2 3 4 5 6 7 8 9 )
7
```

Task 3b: Color Thing Interpreter

```
#lang racket
(require 2htdp/image)
( define ( color-thing )
   ( display "(?): " )
   ( define input ( read ) )
   ( define command (car input) )
   ( define list (cadr input) )
   ( cond
      (
       (eq? command 'random)
       ( define the-element ( list-ref list ( random ( length list ) ) ) )
       ( color-block the-element)
       )
      (
       (eq? command 'all)
       (all-colors list)
      )
      (
        else
           (color-block (list-ref list (- command 1)))
      )
    )
    (color-thing)
)

( define ( all-colors list )
    ( cond
        (
         ( = ( length list ) 0 )
         ( display "" )
        )
        (
         else
         ( color-block ( car list) )
         ( all-colors ( cdr list ) )
        )
    )
)

(define (color-block color)
  ( display ( rectangle 500 25 'solid color ) )
  ( display "\n")
)
```

Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.

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

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

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

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

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

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

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

Task 4a: Establishing the Card code

```
#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 (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)
   )
  )
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (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))
S: undefined;
 cannot reference an identifier before its definition
> (aces? (A C) '(A S))
A: undefined;
 cannot reference an identifier before its definition
> (aces? '(A C) '(A S))
#t
> (aces? '(K S) '(K 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)
'(J H)
> (pick-a-card)
'(Q C)
> (pick-a-card)
'(J S)
> (pick-a-card)
'(J H)
> (pick-a-card)
'(X H)
> (pick-a-card)
'(X H)
>
```

Task 4b: Establishing the Card code

```
(define (pick-two-cards)
  (define test1 (pick-a-card))
  (define test2 (pick-a-card))
  ( cond
    (
      (equal? test1 test2)
      (pick-two-cards)
    )
    ( (list test1 test2) )
)
  )

(define (higher-rank card1 card2)
  (define order '(2 3 4 5 6 7 8 9 X J Q K A) )
  (define rank1 (car card1) )
  (define rank2 (car card2) )
  (define compare-rank ( - (index-of order rank1) (index-of order rank2) ) )
  ( cond
    ( (> compare-rank 0) rank1)
    ( (<= compare-rank 0) rank2)
  )
)
;(trace higher-rank)

(define (classify-two-cards-ur cards)
  (display cards)
  (display ": ")

  (define c1 (car cards) )
  (define c2 (cadr cards) )

  (define high-rank (higher-rank c1 c2))
  (display high-rank)
  (define flush? (eq? (cadr c1) (cadr c2) ))
  (define pair? (eq? (car c1) (car c2) ))

  (define rank-order '(2 3 4 5 6 7 8 9 X J Q K A) )
  (define straight?
    (or
      (= ( + (index-of rank-order (car c1) ) 1 ) (index-of rank-order (car c2) ) )
      (= ( - (index-of rank-order (car c1) ) 1 ) (index-of rank-order (car c2) ) )
    )
  )

  (cond
    (pair? (display " pair"))
    (else (display " high"))
  )

  (cond (straight? (display " straight")))
  (cond (flush? (display " flush")))

)
```

```
Language: racket, with debug
> (pick-two-cards)
'((9 S) (J S))
> (pick-two-cards)
'((K H) (K D))
> (pick-two-cards)
'((A C) (Q H))
> (pick-two-cards)
'((X C) (A S))
> (pick-two-cards)
'((9 H) (3 H))
>
```

```
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(9 S) '(A H))
<'A
'A
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(J C) '(3 S))
<'J
'J
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(5 H) '(A C))
<'A
'A
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(5 S) '(J D))
<'J
'J
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(6 D) '(X S))
<'X
'X
> (higher-rank (pick-a-card) (pick-a-card))
>(higher-rank '(3 S) '(7 H))
<7
7
>
```

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

Task 4c: Two Card Poker Classifier

```
(define (classify-two-cards cards)
  (display cards)
  (display ": " )

  (define rank-order '(2 3 4 5 6 7 8 9 X J Q K A) )
  (define rank-name-parallel '(two three four five six seven eight nine ten jack queen king ace) )

  (define card1 (car cards) )
  (define card2 (cadr cards) )

  (define high-rank (higher-rank card1 card2) )
  (define high-rank-name (list-ref rank-name-parallel (index-of rank-order high-rank)))
  (display high-rank-name)

  (define flush? (eq? (cadr card1) (cadr card2) ))
  (define pair? (eq? (car card1) (car card2) ))

  (define straight?
    (or
      ( = (+ (index-of rank-order (car card1) ) 1) (index-of rank-order (car card2) ) )
      ( = (- (index-of rank-order (car card1) ) 1) (index-of rank-order (car card2) ) )
    )
  )

  (cond
    (pair? (display " pair"))
    (else (display " high"))
  )

  ( cond ( straight? ( display " straight" ) ) )
  ( cond ( flush? ( display " flush" ) ) )

)
```

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