

Racket Programming Assignment #3: Lambda and Basic Lisp

Learning abstract:

The first two tasks of this assignment showcase interactions with lambda functions and list processing operations in lisp. The second task expands to adding more programs that represent the “basic list processing” exhibited in lesson 6. The third task gives me the freedom to play around with different color interpretations within the racket environment. Finally the fourth task which builds up from the card code in lesson 6. This also allows me to be creative and expand my thinking bto different perspectives to get expected outputs.

Task 1a- Three ascending integers

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ((lambda (x) (cons x (cons (+ x 1) (cons (+ x 2) '()))))) 5)
'(5 6 7)
> ((lambda (x) (cons x (cons (+ x 1) (cons (+ x 2) '()))))) 0)
'(0 1 2)
> ((lambda (x) (cons x (cons (+ x 1) (cons (+ x 2) '()))))) 108)
'(108 109 110)
>
```

Task 1b - Make list in reverse order

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ((lambda (x y z) (cons z (cons y (cons x '()))))) 'red 'yellow 'blue)
'(blue yellow red)
> ((lambda (x y z) (cons z (cons y (cons x '()))))) 10 20 30)
'(30 20 10)
> ((lambda (x y z) (cons z (cons y (cons x '()))))) "Professor Plum" "Colonel Mustard" "Miss Scarlet")
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
> |
```

Task 1c - Random number generator


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ((lambda (a b) (random (+ b 1))) 3 5)
5
> ((lambda (a b) (random (+ b 1))) 3 5)
5
> ((lambda (a b) (random (+ b 1))) 3 5)
1
> ((lambda (a b) (random (+ b 1))) 3 5)
3
> ((lambda (a b) (random (+ b 1))) 3 5)
0
> ((lambda (a b) (random (+ b 1))) 3 5)
5
> ((lambda (a b) (random (+ b 1))) 3 5)
3
> ((lambda (a b) (random (+ b 1))) 3 5)
4
> ((lambda (a b) (random (+ b 1))) 3 5)
4
> ((lambda (a b) (random (+ b 1))) 3 5)
3
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
14
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
13
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
15
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
12
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
11
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
14
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
15
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
15
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
12
> ((lambda (a b) (+ a(random (- (+ b 1) a)))) 11 17)
17
>
```

Task 2 - List Processing Referencers and Constructors

Welcome to [DrRacket](#), version 8.6 [cs].

Language: [racket](#), with [debugging](#); memory limit: 128 MB.

```
> ( 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
> ( cdr ( cdr colors ) )
'(yellow orange)
> ( cadr colors )
'blue
> ( caddr 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) )
> ( car ( cdr ( cdr ( cdr pitches ) ) ) )
'fa
> ( caddr pitches )
'fa
> ( list-ref pitches 3 )
'fa
```

```
> ( define a a 'alligator )
 define: bad syntax (multiple expressions after identifier) in: (define a a (quote alligator))
> ( 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

```
#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 )
)
```

Welcome to [DrRacket](#), version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.

```
> ( sampler )
(?) : ( red orange yellow green blue indigo violet )
indigo
(?) : ( red orange yellow green blue indigo violet )
blue
(?) : ( red orange yellow green blue indigo violet )
red
(?) : ( red orange yellow green blue indigo violet )
green
(?) : ( red orange yellow green blue indigo violet )
yellow
(?) : ( red orange yellow green blue indigo violet )
yellow
(?) : ( aet ate eat eta tae tea )
aet
(?) : ( aet ate eat eta tae tea )
tae
(?) : ( aet ate eat eta tae tea )
tea
(?) : ( aet ate eat eta tae tea )
tea
(?) : ( aet ate eat eta tae tea )
ate
(?) : ( aet ate eat eta tae tea )
tae
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
5
(?) : ( 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 )
9
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
3
(?) : ( 0 1 2 3 4 5 6 7 8 9 )
4
```

```
(?) : . user break
```



read: illegal use of `.`

```
>
```

Task 3b - Color Thing interpreter

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

( define ( color-thing )
  ( display "(?): " )
  ( define the-list ( read ) )
  ( define one ( car the-list ) )
  ( define two ( cadr the-list ) )

  ( define the-element
    ( list-ref two ( random ( length two ) ) ) )

  ( define (rectang cl) ( rectangle 600 30 "solid" cl))

  ( define (all-colors n )
    ( define color (list-ref two (- n 1 )))
    ( display (rectang color ) )
    ( display "\n" )
    (cond
      ((eq? 1 n) ( display "\n" ) )
      (else
       (all-colors (- n 1)))))

  ( define ( option one)
    (cond
      ((eq? one 'random)
       ( display (rectang the-element ) ) ( display "\n" ))
      ((eq? one 'all-colors)
       (all-colors (length two)))
      (else
       (display (rectang (list-ref two (- one 1 )))) (display "\n")) ))

  ( option one )
  ( color-thing )
)
```

Welcome to [DrRacket](#), version 8.6 [cs].

Language: racket, with debugging; memory limit: 128 MB.

> (color-thing)

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



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



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



(?): (all-colors (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))



(?): |

Welcome to [DrRacket](#) version 8.6 [cs].

Language: racket, with debugging; memory limit: 128 MB.

> (color-thing)

(?): (random (saddlebrown salmon orchid bisque aquamarine burlywood))



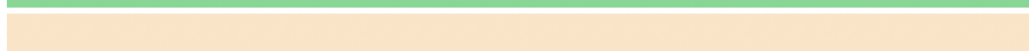
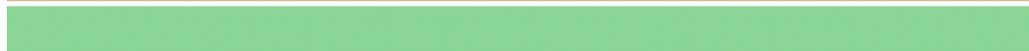
(?): (random (saddlebrown salmon orchid bisque aquamarine burlywood))



(?): (random (saddlebrown salmon orchid bisque aquamarine burlywood))



(?): (all-colors (saddlebrown salmon orchid bisque aquamarine burlywood))



(?): (2 (saddlebrown salmon orchid bisque aquamarine burlywood))



(?): (3 (saddlebrown salmon orchid bisque aquamarine burlywood))



(?): (5 (saddlebrown salmon orchid bisque aquamarine burlywood))



(?): |

Task 4 - Establishing the Card code from lesson #6

Note:

```
. #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: 128 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 ) )
#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 )
'(7 D)
> ( pick-a-card )
'(2 C)
> ( pick-a-card )
'(8 S)
> ( pick-a-card )
'(8 D)
> ( pick-a-card )
'(J S)
> ( pick-a-card )
'(8 C)
> |
```

Task 4b - Two Card Poker Classifier, IR Version

Note: in this assignment 4b and 4c were very difficult and it took more time than usual.
(need to review and practice to build a better understanding).

```
#lang racket
( require racket/trace )

( 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 card1 (list-ref cards(random(length cards))))
  (define card2 (list-ref cards(random(length cards))))
  (cond
    ((eq? card1 card2)
      (pick-two-cards)))
    (list card1 card2)
  )
)

(define (number-of-rank rank)
  (cond
    ((eq? rank 2 ) 1)
    ((eq? rank 3 ) 2)
    ((eq? rank 4 ) 3)
    ((eq? rank 5 ) 4)
    ((eq? rank 6 ) 5)
    ((eq? rank 7 ) 6)
    ((eq? rank 8 ) 7)
    ((eq? rank 9 ) 8)
    ((eq? rank 'X ) 9)
    ((eq? rank 'J ) 10)
    ((eq? rank 'Q ) 11)
    ((eq? rank 'K ) 12)
    ((eq? rank 'A ) 13)
  )
)
```

```

(define (higher-rank card1 card2)
  (define rank-card1(number-of-rank(list-ref card1 0)))
  (define rank-card2(number-of-rank(list-ref card2 0)))
  (cond
    (( > rank-card1 rank-card2) (list-ref card1 0))
    (else
     (list-ref card2 0))
  )
)

(define (classify-two-cards-ur list)
  (define card1(list-ref list 0))
  (define card2(list-ref list 1))

  (define rank-card1 (number-of-rank (list-ref card1 0)))
  (define rank-card2 (number-of-rank (list-ref card2 0)))
  (display list) (display ":")

  (define higher(higher-rank card1 card2))
  (display higher) (display "high" )
  (cond
    ((or ( = rank-card1 (- 1 rank-card2)) (= rank-card2 (- 1 rank-card1)))
     (display "straight"))
    ((eq? (list-ref card1 1 ) (list-ref card2 1 )) (display "flush"))
    ( = rank-card1 rank-card2 ) (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 )
  )
)

```

Welcome to [DrRacket](#), version 8.6 [cs].

Language: racket, with debugging; memory limit: 128 MB.

```
> ( classify-two-cards-ur (pick-two-cards) )
((J D) (3 H)):Jhigh
> ( classify-two-cards-ur (pick-two-cards) )
((Q S) (3 H)):Qhigh
> ( classify-two-cards-ur (pick-two-cards) )
((2 H) (8 D)):8high
> ( classify-two-cards-ur (pick-two-cards) )
((K D) (9 H)):Khigh
> ( classify-two-cards-ur (pick-two-cards) )
((J H) (8 H)):Jhighflush
> ( classify-two-cards-ur (pick-two-cards) )
((7 C) (2 H)):7high
> ( classify-two-cards-ur (pick-two-cards) )
((K C) (9 D)):Khigh
> ( classify-two-cards-ur (pick-two-cards) )
((J D) (6 S)):Jhigh
> ( classify-two-cards-ur (pick-two-cards) )
((9 C) (A C)):Ahighflush
> ( classify-two-cards-ur (pick-two-cards) )
((A C) (4 H)):Ahigh
> ( classify-two-cards-ur (pick-two-cards) )
((X D) (9 D)):Xhighflush
> ( classify-two-cards-ur (pick-two-cards) )
((6 S) (8 H)):8high
> ( classify-two-cards-ur (pick-two-cards) )
((A D) (8 D)):Ahighflush
> ( classify-two-cards-ur (pick-two-cards) )
((7 D) (4 S)):7high
>
```

Welcome to [DrRacket](#), version 8.6 [cs].

Language: racket, with debugging; memory limit: 128 MB.

> (pick-two-cards)

'((X H) (J C))

> (pick-two-cards)

'((X S) (2 H))

> (pick-two-cards)

'((5 H) (3 D))

> (pick-two-cards)

'((A C) (9 D))

> (pick-two-cards)

'((8 D) (4 D))

>

Welcome to [DrRacket](#), version 8.6 [cs].

Language: racket, with debugging; memory limit: 128 MB.

> (higher-rank (pick-a-card) (pick-a-card))

7

> (higher-rank (pick-a-card) (pick-a-card))

'Q

> (higher-rank (pick-a-card) (pick-a-card))

'A

> (higher-rank (pick-a-card) (pick-a-card))

8

> (higher-rank (pick-a-card) (pick-a-card))

7

> |

Task 4c - Two Card Poker Classifier

Welcome to [DrRacket](#), version 8.6 [cs].

Language: racket, with debugging; memory limit: 128 MB.

```
> ( classify-two-cards-ur ( pick-two-cards ) )  
((8 C) (A H)):Ahigh  
> ( classify-two-cards-ur ( pick-two-cards ) )  
((X H) (2 S)):Xhigh  
> ( classify-two-cards-ur ( pick-two-cards ) )  
((J S) (J D)):Jhigh  
> ( classify-two-cards-ur ( pick-two-cards ) )  
((6 H) (X C)):Xhigh  
> ( classify-two-cards-ur ( pick-two-cards ) )  
((J H) (4 S)):Jhigh  
> ( classify-two-cards-ur ( pick-two-cards ) )  
((Q C) (X C)):Qhighflush  
>
```