Racket Programming Assignment #3: Lambda and Basic Lisp

Learning Abstract:

Interaction #1: Lambda Function Play

---

a. List of 3 increasing numbers given smallest

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

b. Reverse order 3 atoms

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( cs1 cs2 cs3 ) ( list cs3 cs2 cs1 ) ) 'red 'yellow 'blue )
'(blue yellow red)
> ( ( lambda ( cs1 cs2 cs3 ) ( list cs3 cs2 cs1 ) ) 10 20 30 )
'(30 20 10)
> ( ( lambda ( cs1 cs2 cs3 ) ( list cs3 cs2 cs1 ) ) "Professor Plum" "Colenel Mustard" "Miss Scarlet" )
'("Miss Scarlet" "Colenel Mustard" "Professor Plum")
>
```

c. Random number between 2 numbers. I made it so order doesn't matter, making it hard to read in the screenshot. Code below:

```
(( lambda ( num1 num 2)
        ( cond
                (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 ))
                (( > num2 num1 )( + ( random ( + 1 ( - num2 num1 ))) num1 ))
        )) 3 5 )
```

```
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
3
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
5
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
3
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
3
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
5
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
4
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
3
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
5
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
5
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 3 5 )
4
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
12
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
14
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
13
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
17
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
17
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
12
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
13
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
14
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
11
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
13
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
13
> ( ( lambda ( num1 num2 ) ( cond (( > num1 num2 )( + ( random ( + 1 ( - num1 num2 ))) num2 )) (( < num1 num2 )( + ( random ( + 1 ( - num2 num1 ))) num1 )))) 11 17
14
```

Interaction #2: List Processing Functions & Constructors

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

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( 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-of0c key-of-g )
key-of0c: undefined;
 cannot reference an identifier before its definition
> ( append key-of-c key-of-g )
'(c d e g a b)
>
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define pitches '(do re mi fa so la ti) )
> ( 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
> ( 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)
>
```

Interaction #3: Little Color Interpreter

Part 1:

```
1  #lang racket
2
3  ( define ( sampler )
4     ( display "(?): " )
5     ( define the-list ( read ) )
6     ( define the-element
7        ( list-ref the-list ( random ( length the-list ) ) )
8     )
9     ( display the-element ) ( display "\n" )
10    ( sampler )
11 )
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( sampler )
(?): ( red orange yellow green blue indigo violet )
orange
(?): ( red orange yellow green blue indigo violet )
red
(?): ( red orange yellow green blue indigo violet )
yellow
(?): ( red orange yellow green blue indigo violet )
indigo
(?): ( red orange yellow green blue indigo violet )
red
(?): ( red orange yellow green blue indigo violet )
yellow
(?): ( aet ate eat eta tae tea )
tea
(?): ( aet ate eat eta tae tea )
ate
(?): ( aet ate eat eta tae tea )
tea
(?): ( aet ate eat eta tae tea )
aet
(?): ( aet ate eat eta tae tea )
eta
(?): ( aet ate eat eta tae tea )
ate
(?): ( 0 1 2 3 4 5 6 7 8 9 )
0
(?): ( 0 1 2 3 4 5 6 7 8 9 )
6
(?): ( 0 1 2 3 4 5 6 7 8 9 )
5
(?): ( 0 1 2 3 4 5 6 7 8 9 )
1
(?): ( 0 1 2 3 4 5 6 7 8 9 )
0
(?): ( 0 1 2 3 4 5 6 7 8 9 )
4
(?): . . user break
read: illegal use of `.`
>
```

Part 2:

```
#lang racket

(require 2htdp/image)

( define ( color-thing )
   ( display "(?): " )
   ( define cmd-list ( read ) )
   ( define the-cmd ( car cmd-list ))
   ( define params ( car ( cdr cmd-list ) ) )
   ( cond
      ( ( equal? the-cmd 'random )
        ( draw-random params )
      )
      ( ( equal? the-cmd 'all )
        ( draw-all params )
      )
      ( else
        ( draw-line ( list-ref params ( - the-cmd 1 ) ) )
      )
   )
   ( color-thing )
)

;; Draw a line for each color in the list (recursive)
( define ( draw-all color-list )
   ( cond
      ( ( empty? ( cdr color-list ) ) ( draw-line ( car color-list ) ) )
      ( else
        ( draw-line ( car color-list ) )
        ( draw-all ( cdr color-list ) )
      )
   )
)

;; Draw a line of a random color
( define ( draw-random color-list )
   ( draw-line ( list-ref color-list ( random ( length color-list ) ) ) )
)

;; Draw a line of a given color (BASE FUNCTION)
( define ( draw-line line-color )
   ( display ( rectangle 1000 20 'solid line-color ) ) ( display "\n" )
)
```

```
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 ( 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 ( olivedrab dodgerblue indigo cyan teal firebrick ) )

(?): ( random ( olivedrab dodgerblue indigo cyan teal firebrick ) )

(?): ( random ( olivedrab dodgerblue indigo cyan teal firebrick ) )

(?): ( all ( olivedrab dodgerblue indigo cyan teal firebrick ) )

(?): ( 2 ( olivedrab dodgerblue indigo cyan teal firebrick ) )

(?): ( 4 ( olivedrab dodgerblue indigo cyan teal firebrick ) )

(?): ( 6 ( olivedrab dodgerblue indigo cyan teal firebrick ) )

(?):
```

Interaction #4: Two Card Poker

1. Given Code Demonstration

```
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 S ) )
#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 )
'(8 H)
> ( pick-a-card )
'(3 S)
> ( pick-a-card )
'(4 S)
> ( pick-a-card )
'(6 H)
> ( pick-a-card )
'(3 C)
> ( pick-a-card )
'(8 S)
>
```

2. Two Card Demonstration

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( pick-two-cards )
'((8 D) (K D))
> ( pick-two-cards )
'((X H) (A H))
> ( pick-two-cards )
'((2 C) (3 D))
> ( pick-two-cards )
'((2 S) (7 S))
> ( pick-two-cards )
'((5 C) (8 S))
> ( pick-two-cards )
'((6 C) (2 S))
> ( pick-two-cards )
'((Q C) (Q H))
> ( pick-two-cards )
'((X D) (A S))
> ( pick-two-cards )
'((J S) (K H))
> ( pick-two-cards )
'((X H) (K S))
> ( pick-two-cards )
'((6 S) (J C))
> ( pick-two-cards )
'((7 D) (J C))
> ( pick-two-cards )
'((K D) (9 D))
> ( pick-two-cards )
'((K D) (9 D))
> ( pick-two-cards )
'((5 D) (8 S))
> ( pick-two-cards )
```

3. Higher Rank Demonstration

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(X D) '(7 H))
<'X
'X
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(J H) '(5 S))
<'J
'J
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(J H) '(5 H))
<'J
'J
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(3 H) '(8 D))
<8
8
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(9 C) '(6 H))
<9
9
>
```

4. Hand Rank Demonstration

```
Language: racket, with debugging; memory limit: 128 MB.
> ( classify-two-cards-ur ( pick-two-cards ) )
((Q D) (8 D)):  Q high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 H) (2 D)):  4 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((J H) (9 H)):  J high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 D) (4 D)):  4 high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 S) (A D)):  A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 D) (8 D)):  8 high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 C) (A S)):  A high
> ( classify-two-cards-ur ( pick-two-cards ) )
((X S) (X C)):  Pair of Xs
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 H) (4 D)):  7 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((K S) (X C)):  K high
> ( classify-two-cards-ur ( pick-two-cards ) )
((Q H) (4 D)):  Q high
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 H) (J S)):  J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 D) (Q S)):  Q high
> ( classify-two-cards-ur ( pick-two-cards ) )
((6 D) (5 H)):  6 high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((Q S) (X D)):  Q high
> ( classify-two-cards-ur ( pick-two-cards ) )
((9 D) (2 D)):  9 high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((K D) (J C)):  K high
> ( classify-two-cards-ur ( pick-two-cards ) )
((K D) (X S)):  K high
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 H) (9 C)):  9 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((J C) (X D)):  J high straight
> ( classify-two-cards-ur ( pick-two-cards ) )
((3 H) (J D)):  J high
> ( classify-two-cards-ur ( pick-two-cards ) )
((7 S) (K S)):  K high flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 H) (9 D)):  9 high
> ( classify-two-cards-ur ( pick-two-cards ) )
((4 D) (5 D)):  5 high straight flush
> ( classify-two-cards-ur ( pick-two-cards ) )
((2 S) (3 S)):  3 high straight flush
>
```

5. The Code!

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

;;GIVEN CODE BELOW HERE
( 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 ) 'H )
     ( equal? ( suit card ) 'D )
   )
)

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

( define ( aces? card1 card2 )
   ( and
     ( equal? ( rank card1 ) 'A )
     ( equal? ( rank card2 ) 'A )
   )
)
;;GIVEN CODE ABOVE HERE

; Pick 2 different cards ( a hand )
( define ( pick-two-cards )
   ( define c1 ( pick-a-card ) )
   ( define c2 ( pick-a-card ) )
   ( cond
      ( ( and ( equal? ( rank c1 ) ( rank c2 ) ) ( equal? ( suit c1 ) ( suit c2 ) ) ) ( pick-two-cards ) )
      ( else ( list c1 c2 ) )
   )
)
;( trace pick-two-cards )
```

```
; HELPER-FUNCTION
; numeric value for card
( define ( card-value card )
   ( define the-rank ( rank card ) )
   ( cond
      ( ( equal? the-rank 'X ) 10 )
      ( ( equal? the-rank 'J ) 11 )
      ( ( equal? the-rank 'Q ) 12 )
      ( ( equal? the-rank 'K ) 13 )
      ( ( equal? the-rank 'A ) 14 )
      ( else ( + the-rank 0 ) )
   )
)
;( trace card-value )

; Higher rank of two cards
( define ( higher-rank card1 card2 )
   ( define val1 ( card-value card1 ) )
   ( define val2 ( card-value card2 ) )
   ( cond
      ( ( > val1 val2 ) ( rank card1 ) )
      ( else ( rank card2 ) )
   )
)
( trace higher-rank )

;; HELPER FUNCTIONS FOR CLASSIFY
; is pair takes two cards
( define ( pair? card1 card2 )
   ( equal? ( rank card1 ) ( rank card2 ) )
)
;( trace pair? )

; is straight
; card rankings 1 appart
( define ( straight? card1 card2 )
   ( or
     ( = ( + 1 ( card-value card1 ) ) ( card-value card2 ) )
     ( = ( - ( card-value card1 ) 1 ) ( card-value card2 ) ) )
)
;( trace straight? )
```

```
; is flush
( define ( flush? card1 card2 )
   ( equal? ( suit card1 ) ( suit card2 ) )
)
;( trace flush? )

; Type of hand
; *assumes* hand like '((card1) (card2))
( define ( classify-two-cards-ur hand )
   ( define card1 ( car hand  ) )
   ( define card2 ( car ( cdr hand  ) ) )
   ( display hand )( display ":  " )
   ( cond
      ( ( pair? card1 card2 ) ( display "Pair of " ) ( display ( rank card1 ) ) ( display "s" ) )
      ( else
        ( display ( higher-rank card1 card2 ) ) ( display " high" )
        ( cond ( ( straight? card1 card2 ) ( display " straight" ) ) )
        ( cond ( ( flush? card1 card2 ) ( display " flush" ) ) )
      )
   )
)
;( trace classify-two-cards-ur )
```