Racket Programming Assignment #3: Lambda and Basic Lisp

Learning Abstract

For this assignment, we take a glimpse of looking into working with lisp and seeing the basics working with list functions and lambda.

Task 1 - Lambda

1a

```
> ( ( lambda ( x )
      ( define y ( + x 1 ) )
      ( define z ( + x 2 ) )
      ( define w ( list x y z ) )
      w ) 5 )
'(5 6 7)
> ( ( lambda ( x )
      ( define y ( + x 1 ) )
      ( define z ( + x 2 ) )
      ( define w ( list x y z ) )
      w ) 0 )
'(0 1 2)
> ( ( lambda ( x )
      ( define y ( + x 1 ) )
      ( define z ( + x 2 ) )
      ( define w ( list x y z ) )
      w ) 108 )
'(108 109 110)
```

1b

```
> ( ( lambda ( x y z )
      ( define w ( reverse ( list x y z ) ) )
      w ) 'red 'yellow 'blue )
'(blue yellow red)
> ( ( lambda ( x y z )
      ( define w ( reverse ( list x y z ) ) )
      w ) '10 '20 '30 )
'(30 20 10)
> ( ( lambda ( x y z )
      ( define w ( reverse ( list x y z ) ) )
      w )  "Professor Plum" "Colonel Mustard" "Miss Scarlet" )
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
```

1c

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

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

```
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
16
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
12
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
14
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
15
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
14
```

```
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
13
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
12
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
15
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
17
> ( ( lambda ( x y )
    ( random x ( + y 1 ) )
    ) 11 17 )
14
```

Task 2 - List Processing Referencers and Constructors

```
> ( define colors `(red blue yellow orange) )
> colors
'(red blue yellow orange)
> 'colors
'colors: undefined;
 cannot reference an identifier before its definition
> ( 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
>
```

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

3a

Code

```
( 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 )
(?): ( red orange yellow green blue indigo violet )
yellow
(?): ( red orange yellow green blue indigo violet )
violet
(?): ( red orange yellow green blue indigo violet )
green
(?): ( red orange yellow green blue indigo violet )
green
(?): ( red orange yellow green blue indigo violet )
violet
(?): ( red orange yellow green blue indigo violet )
orange

(?):  ( aet ate eat eta tae tea )
eta
(?):  ( aet ate eat eta tae tea )
tea
(?):  ( 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
```

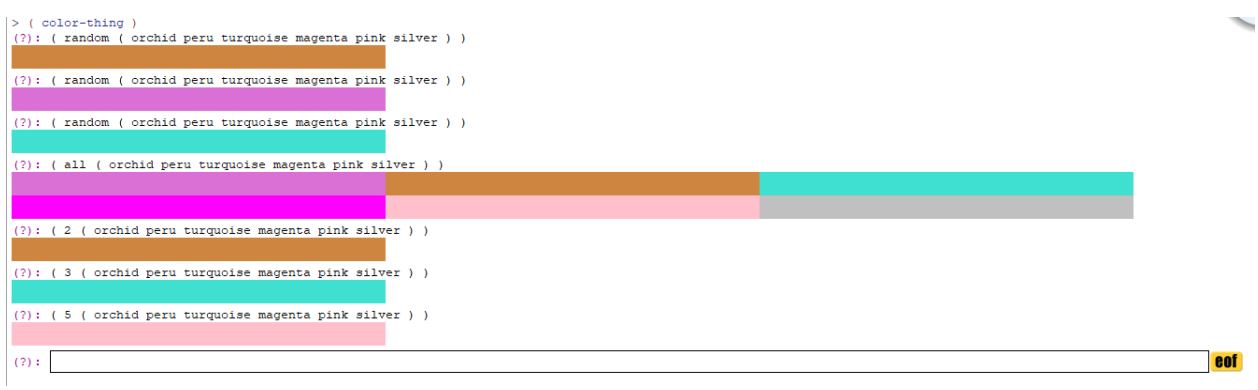
```
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
4
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
8
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
3
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
8
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
9
(?):  ( 0 1 2 3 4 5 6 7 8 9 )
7
```

3b

Code

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

Demo



Task 4 - Two Card Poker

4a

```
>  ( 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 )
'(8 H)
> ( pick-a-card )
'(2 H)
> ( pick-a-card )
'(6 C)
> ( pick-a-card )
'(X S)
> ( pick-a-card )
'(8 H)
> ( pick-a-card )
'(4 S)
```

4b & 4c

Code

```
( 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 ( 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 )
 )
)
```

```
( define ( pick-two-cards )
 ( define card1 ( pick-a-card ) )
 ( define card2 ( pick-a-card ) )
 ( cond
 ( ( equal? card1 card2 )
 ( pick-two-cards )
 )
 ( else
 ( list card1 card2 )
 )
 )
)

( define ( higher-rank card1 card2 )
 ( display ( list card1 card2 ) ) ( display "\n" )
 ( cond
 ( ( equal? ( rank card1 ) 'A )
 ( display ( rank card1 ) )
 )
 ( ( equal? ( rank card2 ) 'A )
 ( display ( rank card2 ) )
 )
 ( ( equal? ( rank card1 ) 'K )
 ( display ( rank card1 ) )
 )
 ( ( equal? ( rank card2 ) 'K )
 ( display ( rank card2 ) )
 )
 ( ( equal? ( rank card1 ) 'Q )
 ( display ( rank card1 ) )
 )
 ( ( equal? ( rank card2 ) 'Q )
 ( display ( rank card2 ) )
 )
```

```
 ( ( equal? ( rank card1 ) 'J )
 ( display ( rank card1 ) )
 )
 ( ( equal? ( rank card2 ) 'J )
 ( display ( rank card2 ) )
 )
 ( ( equal? ( rank card1 ) 'X )
 ( display ( rank card1 ) )
 )
 ( ( equal? ( rank card2 ) 'X )
 ( display ( rank card2 ) )
 )
 ( else
 ( cond
 ( ( equal? ( rank card1 ) ( rank card2 ) )
 ( rank card1 )
 )
 ( ( > ( rank card1 ) ( rank card2 ) )
 ( display ( rank card1 ) )
 )
 ( ( < ( rank card1 ) ( rank card2 ) )
 ( display ( rank card2 ) )
 )
 )
 )
 )
)


( define ( classify-two-cards-ur cards )
 ( display cards ) ( display ": " )
 ( define card1 ( first cards ) )
 ( define card2 ( second cards ) )
```

```
( cond
( ( equal? ( rank card1 ) ( rank card2 ) )
( display "Pair of " ) ( display ( rank card1 ) ) ( display "'s" )
)
( ( equal? ( rank card1 ) 'A )
( display ( rank card1 ) ) ( display " High" )
)
( ( equal? ( rank card2 ) 'A )
( display ( rank card2 ) ) ( display " High" )
)
( ( equal? ( rank card1 ) 'K )
( display ( rank card1 ) ) ( display " High" )
)
( ( equal? ( rank card2 ) 'K )
( display ( rank card2 ) ) ( display " High" )
)
( ( equal? ( rank card1 ) 'Q )
( display ( rank card1 ) ) ( display " High" )
)
( ( equal? ( rank card2 ) 'Q )
( display ( rank card2 ) ) ( display " High" )
)
( ( equal? ( rank card1 ) 'J )
( display ( rank card1 ) ) ( display " High" )
)
( ( equal? ( rank card2 ) 'J )
( display ( rank card2 ) ) ( display " High" )
)
( ( equal? ( rank card1 ) 'X )
( display ( rank card1 ) ) ( display " High" )
)
( ( equal? ( rank card2 ) 'X )
( display ( rank card2 ) ) ( display " High" )
)
```

```
 ( else
 ( cond
 ( ( > ( rank card1 ) ( rank card2 ) )
 ( display ( rank card1 ) ) ( display " High" )
 )
 ( ( < ( rank card1 ) ( rank card2 ) )
 ( display ( rank card2 ) ) ( display " High" )
 )
 )
 )
 )
 ( straight? card1 card2 )
 ( flush? card1 card2 )
)


( define ( straight? card1 card2 )
 ( cond
 ( ( equal? ( rank card1 ) 'A )
 ( cond
 ( ( equal? ( rank card2 ) 'K )
 ( display " Straight" )
 )
 )
 )
 ( ( equal? ( rank card1 ) 'K )
 ( cond
 ( ( equal? ( rank card2 ) 'A )
 ( display " Straight" )
 )
 ( ( equal? ( rank card2 ) 'Q )
 ( display " Straight" )
 )
 )
 )
```

```
( ( equal? ( rank card1 ) 'K )
( cond
( ( equal? ( rank card2 ) 'A )
( display " Straight" )
)
( ( equal? ( rank card2 ) 'Q )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) 'Q )
( cond
( ( equal? ( rank card2 ) 'K )
( display " Straight" )
)
( ( equal? ( rank card2 ) 'J )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) 'J )
( cond
( ( equal? ( rank card2 ) 'Q )
( display " Straight" )
)
( ( equal? ( rank card2 ) 'X )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) 'X )
( cond
( ( equal? ( rank card2 ) 'J )
( display " Straight" )
)
```

```
( ( equal? ( rank card2 ) "9" )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) '9 )
( cond
( ( equal? ( rank card2 ) 'X )
( display " Straight" )
)
( ( equal? ( rank card2 ) '8 )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) '8 )
( cond
( ( equal? ( rank card2 ) '9 )
( display " Straight" )
)
( ( equal? ( rank card2 ) '7 )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) '7 )
( cond
( ( equal? ( rank card2 ) '8 )
( display " Straight" )
)
( ( equal? ( rank card2 ) '6 )
( display " Straight" )
)
)
)
```

```
( ( equal? ( rank card1 ) '6 )
( cond
( ( equal? ( rank card2 ) '7 )
( display " Straight" )
)
( ( equal? ( rank card2 ) '5 )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) '5 )
( cond
( ( equal? ( rank card2 ) '6 )
( display " Straight" )
)
( ( equal? ( rank card2 ) '4 )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) '4 )
( cond
( ( equal? ( rank card2 ) '5 )
( display " Straight" )
)
( ( equal? ( rank card2 ) '3 )
( display " Straight" )
)
)
)
( ( equal? ( rank card1 ) '3 )
( cond
( ( equal? ( rank card2 ) '4 )
( display " Straight" )
```

```
 ( ( equal? ( rank card1 ) '2 )
 ( cond
 ( ( equal? ( rank card2 ) '3 )
 ( display " Straight" )
 )
 ( ( equal? ( rank card2 ) '1 )
 ( display " Straight" )
 )
 )
 )
 ( ( equal? ( rank card1 ) '1 )
 ( cond
 ( ( equal? ( rank card2 ) '2 )
 ( display " Straight" )
 )
 )
 )
 )
)


( define ( flush? card1 card2 )
 ( cond
 ( ( equal? ( suit card1 ) ( suit card2 ) )
 ( display " Flush" )
 )
 )
)

( trace higher-rank )
```

Demo

```
>  ( classify-two-cards-ur ( pick-two-cards ) )
((7 C) (9 C)): 9 High Flush
>  ( classify-two-cards-ur ( pick-two-cards ) )
((8 S) (7 H)): 8 High Straight
>  ( classify-two-cards-ur ( pick-two-cards ) )
((K S) (Q S)): K High Straight Flush
>  ( classify-two-cards-ur ( pick-two-cards ) )
((K C) (6 S)): K High
>  ( classify-two-cards-ur ( pick-two-cards ) )
((A S) (8 D)): A High
>  ( classify-two-cards-ur ( pick-two-cards ) )
((6 C) (6 S)): Pair of 6's
>  ( classify-two-cards-ur ( pick-two-cards ) )
((6 H) (5 D)): 6 High Straight
>  ( classify-two-cards-ur ( pick-two-cards ) )
((Q S) (A H)): A High
>  ( classify-two-cards-ur ( pick-two-cards ) )
((A H) (6 H)): A High Flush
>  ( classify-two-cards-ur ( pick-two-cards ) )
((3 H) (7 C)): 7 High
>
```