Racket Assignment 3

Eli Fereira

Abstract

Now that we're comfortable with racket and recursion, lab 3 aims specifically at lambdas and list processing. Whether it is searching through a list to find an element, or getting random elements in a list, mostly everything in this lab involves some sort of list. It's especially important that we get used to lists, as it is the base data type in lisp.

Task 1: Lambdas

1a:

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

1b:

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( el1 el2 el3 ) ( list el3 el2 el1 ) ) 'red 'yellow 'blue )
'(blue yellow red)
> ( ( lambda ( el1 el2 el3 ) ( list el3 el2 el1 ) ) 10 20 30 )
'(30 20 10)
> ( ( lambda ( el1 el2 el3 ) ( list el3 el2 el1 ) ) "Professor Plum" "Colonel Mustard" "Miss Scarlet" )
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
>
```

1c:

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
3.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
4.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
5.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
4.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
4.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
4.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
5.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
5.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
4.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 3 5 )
4.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
12.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
16.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
12.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
16.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
12.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
12.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
14.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
12.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
12.0
> ( ( lambda ( low high ) ( round ( + ( * ( random ) ( - high low ) ) low ) ) ) 11 17 )
11.0
>
```

Task 2

```
> ( 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
> ( 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 ) )
> ( 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 animals ) ) ) )
🛑 animals: undefined;
 cannot reference an identifier before its definition
> ( 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: Color Thing

sampler.rkt

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

color_thing.rkt

```
#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 ) ) ) )
       ( display-color the-element ) ( display "\n" )
      )
      (
       ( eq? command 'all )
       ( all-colors list )
      )
      (
       else
       ( display-color ( list-ref list ( - command 1 ) ) ) ( display "\n" )
       )
   )
   ( color-thing )
   )

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

( define ( display-color c )
   ( display ( rectangle 500 25 'solid c ) )
   )
( color-thing )
```

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

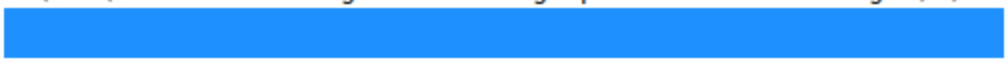
```
? ( random ( olivedrab dodgerblue indigo plum teal darkorange ) )
```

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

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

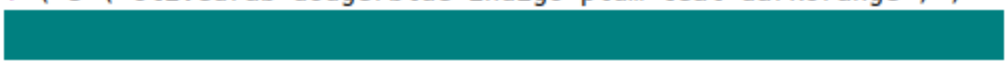
```
? ( all ( olivedrab dodgerblue indigo plum teal darkorange ) )
```

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



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

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



```
?
```

Task 4a

Cards 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
> ( black? c1 )
#t
> ( black? c2 )
#f
> ( aces? '( A C ) '( A S ) )
#t
> ( aces? '( K S ) '( A C ) )
#f
> ( 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 C)
> ( pick-a-card )
'(3 C)
> ( pick-a-card )
'(2 H)
> ( pick-a-card )
'(7 D)
> ( pick-a-card )
'(5 H)
> ( pick-a-card )
'(K H)
>
```

Task 4b

Pick Two Cards Demo

```
> ( pick-two-cards )
'((A S) (X C))
> ( pick-two-cards )
'((5 H) (2 H))
> ( pick-two-cards )
'((K C) (3 D))
> ( pick-two-cards )
'((9 D) (5 C))
> ( pick-two-cards )
'((6 H) (6 D))
>
```

Higher Rank Demo

```
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(5 H) '(7 S))
<7
7
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(K S) '(K D))
<'K
'K
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(K S) '(A C))
<'A
'A
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(3 C) '(K C))
<'K
'K
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(4 H) '(X D))
<'X
'X
>
```

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

Task 4c

Classify Two Cards Demo

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

Task 4 Code:

classifier.rkt

```
#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 ( 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 cards ( list ( pick-a-card ) ( pick-a-card ) ) )
   ( cond
      ( ( equal? ( car cards ) ( cadr cards ) )
        ( pick-two-cards )
        )
      ( else cards )
      )
   )

( define ( higher-rank c1 c2 )
   ( define rank-order '( 2 3 4 5 6 7 8 9 X J Q K A ) )
   ( define c1-rank ( car c1 ) )
   ( define c2-rank ( car c2 ) )
   ( define rank-compare ( - ( index-of rank-order c1-rank) ( index-of rank-order
c2-rank ) ) )
   ( cond
      ( ( > rank-compare 0 ) c1-rank )
      ( ( <= rank-compare 0 ) c2-rank )
      )
   )

; ( 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" ) ) )
   )

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

   ( define high-rank ( higher-rank c1 c2 ) )
   ( define high-rank-name ( list-ref rank-name-parallel ( index-of rank-order
high-rank ) ) )
   ( display high-rank-name )

   ( define flush? ( eq? ( cadr c1 ) ( cadr c2 ) ) )
   ( define pair? ( eq? ( car c1 ) ( car c2 ) ) )

   ( 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" ) ) )
  )
```