

Learning Abstract

The purpose of this assignment was to become more adapt to racket programming. These assignments feature a mixture of introduction to lambda functions

Part 1.A

```
> ( ( λ ( x )
      ( cons x
        ( cons (+ x 1)
          ( cons (+ x 2)
            ' ( ( ( ( ( ) ) ) ) ) 5 )
          )
        )
      )
  ' (5 6 7)
> ( ( λ ( x )
      ( cons x
        ( cons (+ x 1)
          ( cons (+ x 2)
            ' ( ( ( ( ( ) ) ) ) ) 0 )
          )
        )
      )
  ' (0 1 2)
> ( ( λ ( x )
      ( cons x
        ( cons (+ x 1)
          ( cons (+ x 2)
            ' ( ( ( ( ( ) ) ) ) ) 108 )
          )
        )
      )
  ' (108 109 110)
> |
```

Part 1.B

```
> ( ( λ ( x y z )
      (list z y x) )
  'red 'yellow 'blue)
' (blue yellow red)
> ( ( λ ( x y z )
      (list z y x) )
  10 20 30)
' (30 20 10)
> ( ( λ ( x y z )
      (list z y x) )
  "Professor Plum" "Colonel Mustard" "Miss Scarlet" )
' ("Miss Scarlet" "Colonel Mustard" "Professor Plum")
> |
```

1.C

```
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 3 5 )  
5  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 3 5 )  
3  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 3 5 )  
5  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 3 5 )  
3  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 3 5 )  
5  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 3 5 )  
3  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 3 5 )  
3  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 3 5 )  
3  
~
```

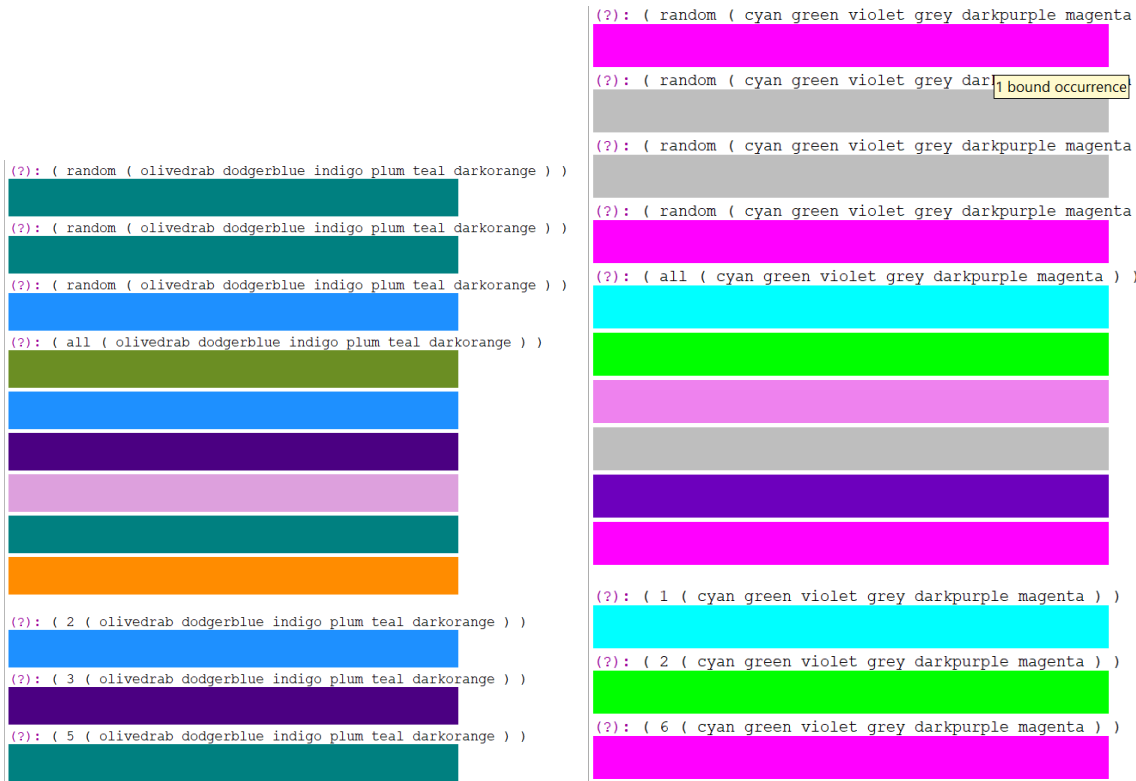
```
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
17  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
11  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
17  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
11  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
17  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
17  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
17  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
17  
> ( ( λ ( a b )  
      ( define L ( list a b ))  
      ( define r ( random 0 2))  
      ( cond  
        [(eq? r 0)(display (list-ref L 0))]  
        [else (display (list-ref L 1 ))]  
      )) 11 17 )  
17  
~
```

```
> ( define languages '(racket prolog haskell rust ) )
> languages
'(racket prolog haskell rust)
> 'languages
'languages
> ( quote languages )
'languages
> ( car languages )
'racket
> ( cdr languages )
'(prolog haskell rust)
> ( car ( cdr languages ) )
'prolog
> ( cdr ( cdr languages ) )
'(haskell rust)
> ( cadr languages )
'prolog
> ( caddr languages )
'(haskell rust)
> ( first languages )
'racket
> ( second languages )
'prolog
> ( third languages )
'haskell
> ( list-ref languages 2 )
'haskell
> ( define '(1 2 3 ) )
❌ define: not an identifier, identifier with default, or keyword for procedure argument in: (1 2 3)
> ( define numbers '( 1 2 3 ) )
> ( define letters '( a b c ) )
> ( cons numbers letters )
'((1 2 3) a b c)
> ( list numbers letters )
'((1 2 3) (a b c))
> ( append numbers letters )
'(1 2 3 a b c)
> ( define animals '(ant bat cat dot eel ) )
> ( car ( cdr ( cdr ( cdr animals ) ) ) )
'dot
> ( caddr animals )
'cat
> ( list-ref animals 3 )
'dot
> ( define a 'apple )
> ( define b 'peach )
> ( define c 'cherry)
> ( cons a ( cons b ( cons c '() ) ) )
'(apple peach cherry)
> ( list a b c )
'(apple peach cherry)
> ( define x '(one fish ) )
> ( define y '(two fish ) )
> ( cons ( car x ) (cons ( car ( cdr x ) ) y ) )
'(one fish two fish)
> ( append x y )
'(one fish two fish)
> |
```

Part 3.A

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

Part 3.B



```
#lang racket

( require 2htdp/image )

( define ( color-thing )
  ( display "(?): " )
  ( define the-list ( read ) )
  ( cond
    [( equal? ( car the-list ) 'all )
      ( display-all-bars ( cadr the-list ) ) ]
    [( equal? ( car the-list ) 'random )
      ( display-random ( cadr the-list ) ) ]
    [else
      (display-one ( car the-list ) ( cadr the-list ) ) ] )
  ( display "\n" )
  ( color-thing ) )

( define ( bar color ) ( display ( rectangle 600 50 "solid" color ) ) )

( define ( display-all-bars color-list )
  (cond
    [(not ( empty? color-list ) )
      ( bar ( car color-list ) )
      ( display "\n" )
      ( display-all-bars ( cdr color-list ) ) ] ) )

( define ( display-random color-list )
  ( bar ( list-ref color-list ( random ( length color-list ) ) ) ) )

( define ( display-one choice color-list )
  ( bar ( list-ref color-list ( - choice 1 ) ) ) )
```

Part 4.A

Welcome to [DrRacket](#), version 8.3 [cs].

Language: [racket](#), with [debugging](#); memory limit: 1024 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))
aces: undefined;
cannot reference an identifier before its definition
> (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)
'(X S)
> (pick-a-card)
'(J S)
> (pick-a-card)
'(K S)
> (pick-a-card)
'(2 H)
> (pick-a-card)
'(K H)
> (pick-a-card)
'(Q D)
```

```
#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 )  
  )  
 )
```

Part 4.B

```
> ( pick-two-cards )  
'((8 H) (7 H))  
> ( pick-two-cards )  
'((3 S) (A H))  
> ( pick-two-cards )  
'((2 D) (5 H))  
> ( pick-two-cards )  
'((Q C) (4 C))  
> ( pick-two-cards )  
'((K S) (5 C))  
> ( pick-two-cards )  
'((A C) (2 S))  
> |
```

```
( define ( pick-two-cards )  
  ( list ( pick-a-card ) ( pick-a-card ) ) )
```

Part 4.B cont

```
71 ( define ( rank-indexer card )
72   ( define card-rank ( car card ) )
73   ( cond
74     [(number? card-rank) card-rank]
75     [else (face-card-indexer card-rank)]))
76
77 ( define ( face-card-indexer card-rank )
78   ( cond
79     [(eq? 'X card-rank) 10]
80     [(eq? 'J card-rank) 11]
81     [(eq? 'Q card-rank) 12]
82     [(eq? 'K card-rank) 13]
83     [(eq? 'A card-rank) 14]
84     [else 0]))
85
86 ( define ( higher-rank card1 card2 )
87   ( define card1-rank ( rank-indexer card1 ) )
88   ( define card2-rank ( rank-indexer card2 ) )
89   ( cond
90     [(< card1-rank card2-rank) display (car card2)]
91     [(> card1-rank card2-rank) display (car card1)] ) )
92
93
94 ( trace higher-rank )
95
96
```

Welcome to [DrRacket](#), version 8.3 [cs].
Language: racket, with debugging; memory limit: 1024 MB.

```
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(6 D) '(8 S))
<8
8
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(2 C) '(A C))
<'A
'A
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(A S) '(9 S))
<'A
'A
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(2 S) '(3 D))
<3
3
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(A S) '(3 S))
<'A
'A
>
```

Part 4.B cont

```
( define ( classify-two-cards-ur card-pair )
  ( define card1 ( car card-pair ) )
  ( define card2 ( cadr card-pair ) )
  ( define card1-rank ( rank-indexer card1))
  ( define card2-rank ( rank-indexer card2))
  ( define card1-suit ( suit card1))
  ( define card2-suit ( suit card2))
  ( define high-card ( higher-rank card1 card2))
  ( display card-pair)
  (display ": ")
  ( cond
    ((equal? card1-suit card2-suit)
     (cond
      ((or
        (= 1 ( - card1-rank card2-rank))
        (= 1 ( - card2-rank card1-rank)))
       (display high-card (display " high straight flush" )))
      (else
       (display high-card) (display " high flush "))))
    (else
     (cond
      ((or
        (= 1 ( - card1-rank card2-rank))
        (= 1 ( - card2-rank card1-rank)))
       (display high-card) (display " high straight" )))
      (else
       (cond
        ((equal? (car card1) (car card2))
         (display "Pair of " ) (display (car card1))(display "'s"))
        (else
         (display high-card) (display " high" ))))))))))))
```

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