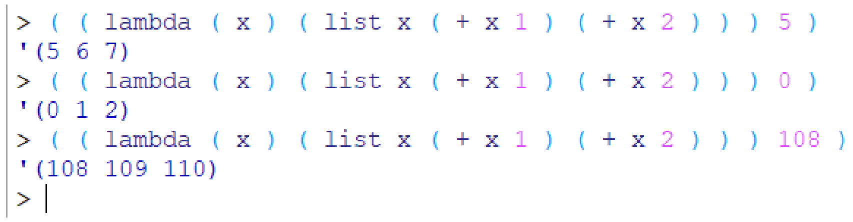
Sven Kappeler - Racket Assignment #3:

Learning Abstract

This assignment is an introduction to the Racket programming language. The purpose of this assignment is to get myself acquainted with recursive functions and lambda functions whilst using the Racket programming language.

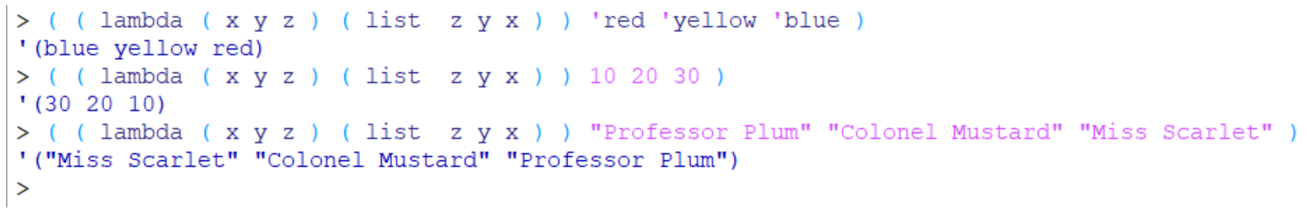
Task 1: Lambda Functions

a)



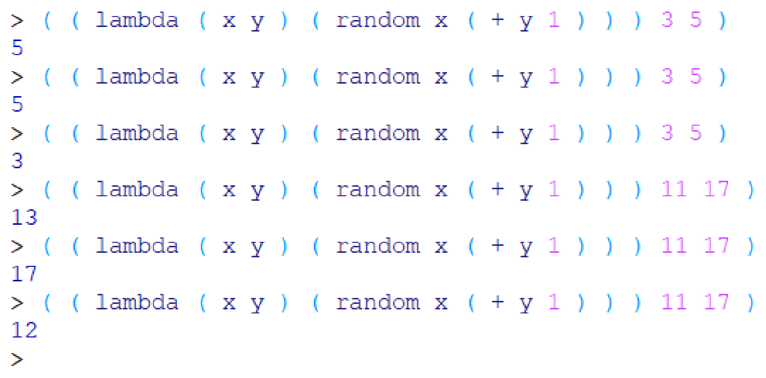
```
> ( ( lambda ( x ) ( list x ( + x 1 ) ( + x 2 ) ) ) 5 )
'(5 6 7)
> ( ( lambda ( x ) ( list x ( + x 1 ) ( + x 2 ) ) ) 0 )
'(0 1 2)
> ( ( lambda ( x ) ( list x ( + x 1 ) ( + x 2 ) ) ) 108 )
'(108 109 110)
> 
```

b)



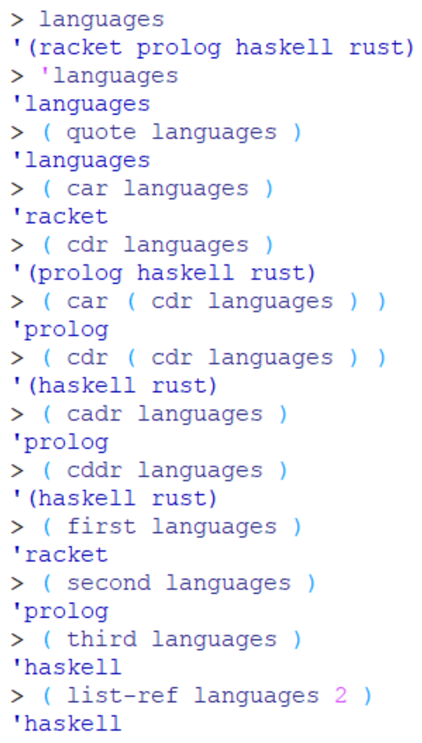
```
> ( ( lambda ( x y z ) ( list  z y x ) ) 'red 'yellow 'blue )
'(blue yellow red)
> ( ( lambda ( x y z ) ( list  z y x ) ) 10 20 30 )
'(30 20 10)
> ( ( lambda ( x y z ) ( list  z y x ) ) "Professor Plum" "Colonel Mustard" "Miss Scarlet" )
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
>
```

c)

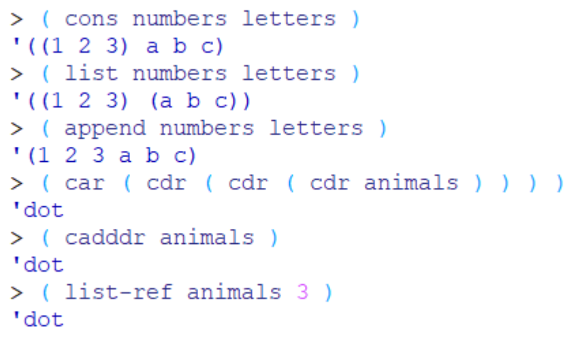


```
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
5
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
5
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 3 5 )
3
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
13
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
17
> ( ( lambda ( x y ) ( random x ( + y 1 ) ) ) 11 17 )
12
>
```

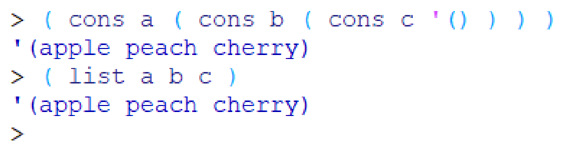
Task 2: List Processing



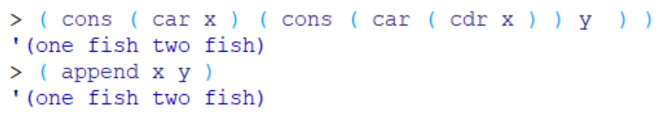
```
> languages
'(racket prolog haskell rust)
> 'languages
'languages
> ( quote languages )
'languages
> ( car languages )
'racket
> ( cdr languages )
'(prolog haskell rust)
> ( car ( cdr languages ) )
'prolog
> ( cdr ( cdr languages ) )
'(haskell rust)
> ( cadr languages )
'prolog
> ( cddr languages )
'(haskell rust)
> ( first languages )
'racket
> ( second languages )
'prolog
> ( third languages )
'haskell
> ( list-ref languages 2 )
'haskell
```



```
> ( 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)
> ( car ( cdr ( cdr ( cdr animals ) ) ) )
'dot
> ( cadddr animals )
'dot
> ( list-ref animals 3 )
'dot
```

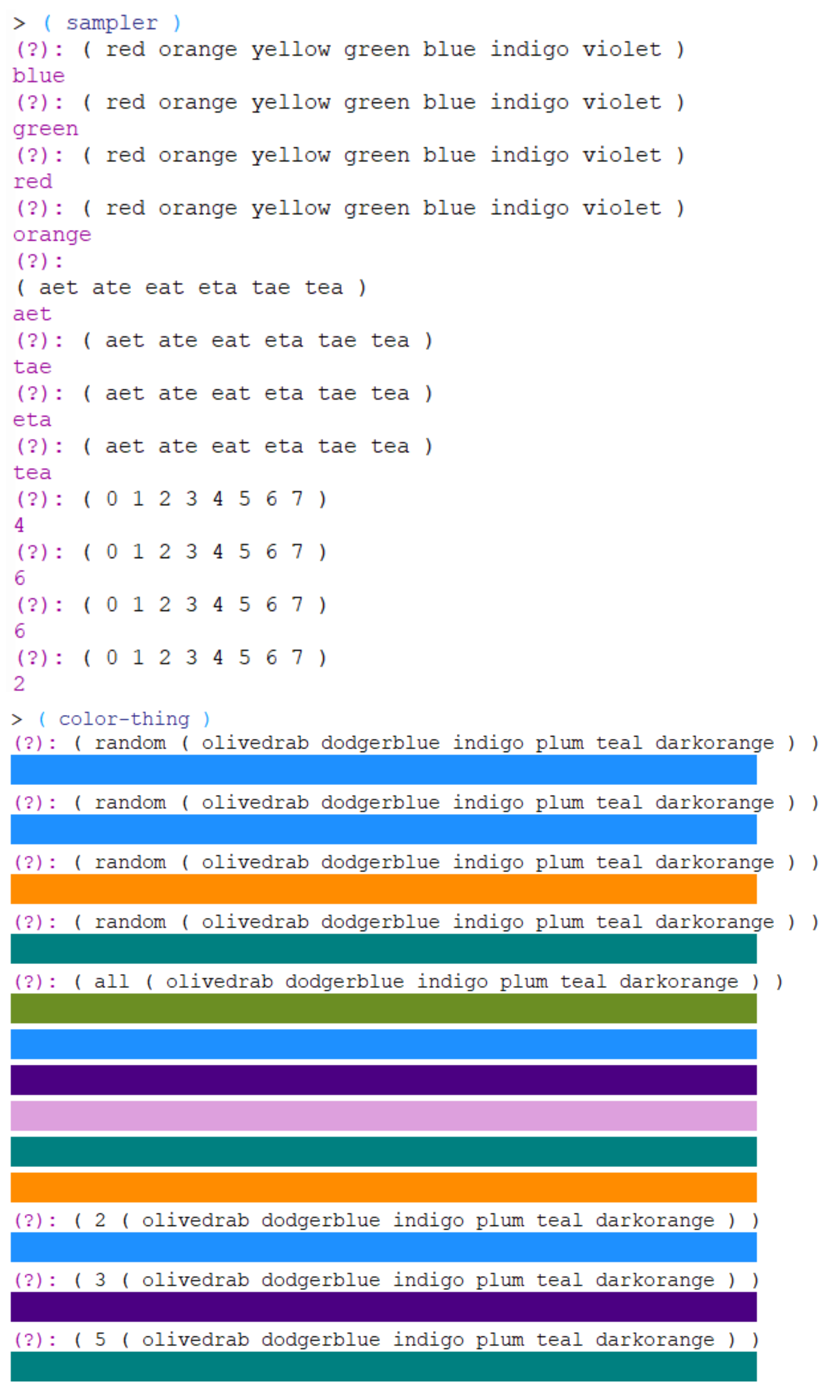


```
> ( cons a ( cons b ( cons c '() ) ) )
'(apple peach cherry)
> ( list a b c )
'(apple peach cherry)
>
```



```
> ( cons ( car x ) ( cons ( car ( cdr x ) ) y  ) )
'(one fish two fish)
> ( append x y )
'(one fish two fish)
```

Task 3: Color Interperter



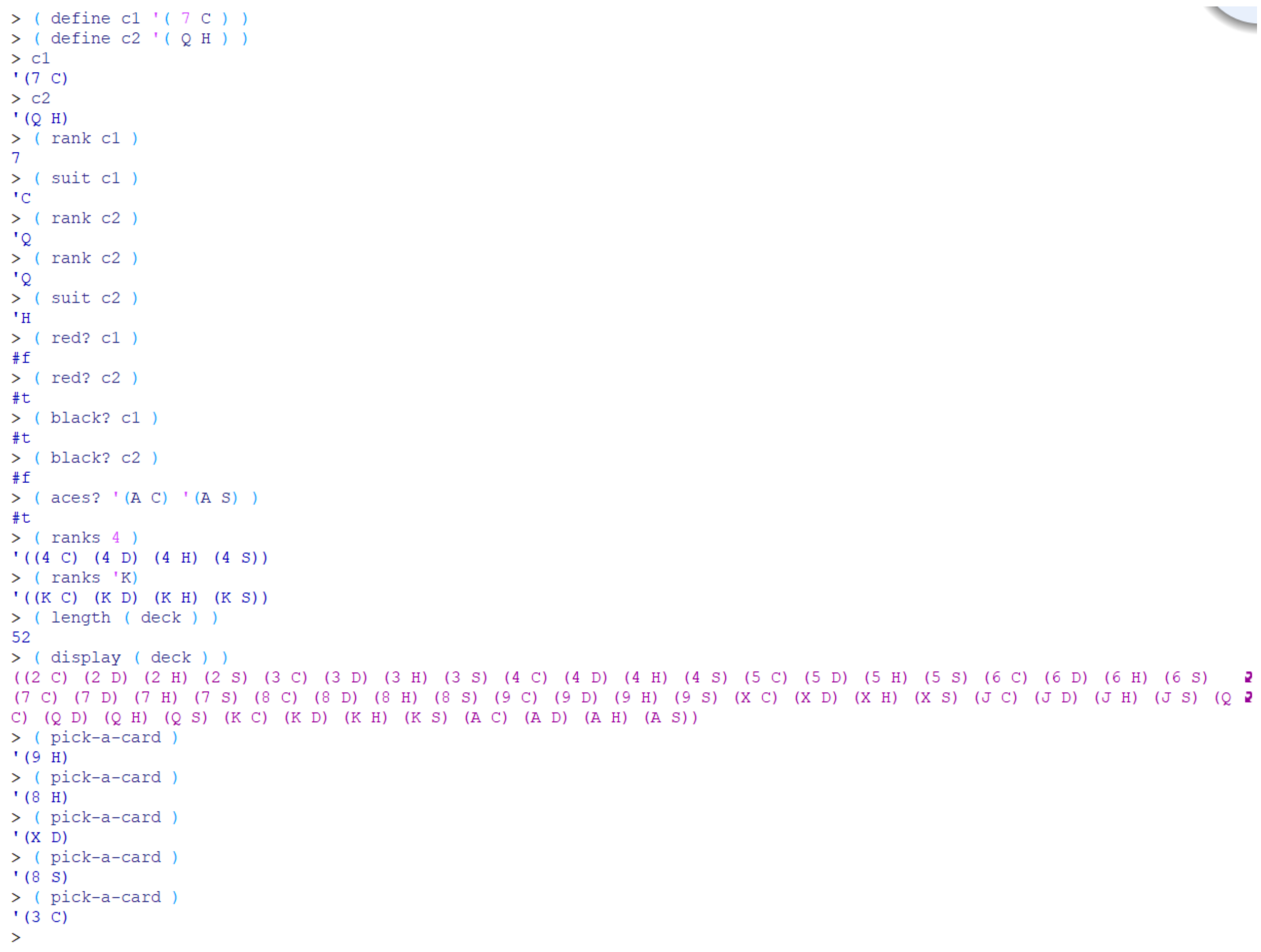
```
> ( sampler )
(?): ( red orange yellow green blue indigo violet )
blue
(?): ( red orange yellow green blue indigo violet )
green
(?): ( red orange yellow green blue indigo violet )
red
(?): ( red orange yellow green blue indigo violet )
orange
(?):
( aet ate eat eta tae tea )
aet
(?): ( aet ate eat eta tae tea )
tae
(?): ( aet ate eat eta tae tea )
eta
(?): ( aet ate eat eta tae tea )
tea
(?): ( 0 1 2 3 4 5 6 7 )
4
(?): ( 0 1 2 3 4 5 6 7 )
6
(?): ( 0 1 2 3 4 5 6 7 )
6
(?): ( 0 1 2 3 4 5 6 7 )
2
> ( color-thing )
(?): ( random ( olivedrab dodgerblue indigo plum teal darkorange ) )
(?): ( 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 ) )
```

```
(?): ( random ( tomato maroon chocolate gold lime cyan ) )
🟥
(?): ( random ( tomato maroon chocolate gold lime cyan ) )
🟩
(?): ( random ( tomato maroon chocolate gold lime cyan ) )
🟦
(?): ( all ( tomato maroon chocolate gold lime cyan ) )
🟥
🟪
🟫
🟨
🟩
🟦
(?): ( 2 ( tomato maroon chocolate gold lime cyan ) )
🟪
(?): ( 5 ( tomato maroon chocolate gold lime cyan ) )
🟩
(?): ( 3 ( tomato maroon chocolate gold lime cyan ) )
🟫
```

```racket
#lang racket

( require 2htdp/image )

; Rectangle ---

( define ( draw-rect color )
   ( display ( rectangle 500 20 "solid" color ) )
   ( display "\n" )
)

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

( define ( all-color lst )
   ( cond
      [ ( empty? lst ) ( display empty-image ) ]
      [ else
        ( draw-rect ( car lst ) )
        ( all-color ( cdr lst ) ) ]
  )
)

( define ( random-color lst )
   ( define rand ( random ( length lst ) ) )
   ( draw-rect ( list-ref lst rand ) )
)

( define ( pick-color select lst )
   ( draw-rect ( list-ref lst ( - select 1 ) ) )
)
```

Task 4: Two Card Poker

A)



```
> ( define c1 '( 7 C ) )
> ( define c2 '( Q H ) )
> c1
'(7 C)
> c2
'(Q H)
> ( rank c1 )
7
> ( suit c1 )
'C
> ( rank c2 )
'Q
> ( rank c2 )
'Q
> ( suit c2 )
'H
> ( red? c1 )
#f
> ( red? c2 )
#t
> ( black? c1 )
#t
> ( black? c2 )
#f
> ( aces? '(A C) '(A S) )
#t
> ( 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 )
'(9 H)
> ( pick-a-card )
'(8 H)
> ( pick-a-card )
'(X D)
> ( pick-a-card )
'(8 S)
> ( pick-a-card )
'(3 C)
>
```

B)

inprogress