

Learning Abstract: This assignment will introduce Lambda and Basic Lisp. With this knowledge, I will be able to manipulate lists in Racket.

## Task 1

Demo for 1a Three ascending integers:

---

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( x )
      ( define y (+ x 1) )
      ( define z (+ x 2) )
      ( cons x ( cons y ( cons z '() ) ) )
    )
  5
)
'(5 6 7)
> ( ( lambda ( x )
      ( define y (+ x 1) )
      ( define z (+ x 2) )
      ( cons x ( cons y ( cons z '() ) ) )
    )
  0
)
'(0 1 2)
> ( ( lambda ( x )
      ( define y (+ x 1) )
      ( define z (+ x 2) )
      ( cons x ( cons y ( cons z '() ) ) )
    )
  108
)
'(108 109 110)
>
```

Demo for 1b Make list in reverse :

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( x y z )
      ( cons z ( cons y ( cons x '() ) ) )
    )
  'red 'yellow 'blue
)
'(blue yellow red)
> ( ( lambda ( x y z )
      ( cons z ( cons y ( cons x '() ) ) )
    )
  10 20 30
)
'(30 20 10)
> ( ( lambda ( x y z )
      ( cons z ( cons y ( cons x '() ) ) )
    )
  "Professor Plum" "Colonel Mustard" "Miss Scarlet"
)
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
>
```

## Demo for 1c Random number generator:





```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
5
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
3
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
3
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
3
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
3
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
5
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
5
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
3
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
5
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 3 5 )
3
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
11
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
17
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
11
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
17
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
11
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
17
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
11
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
17
> ( ( lambda ( x y ) ( define outcome ( random 2 ) ) ( cond [ ( eq? outcome 0) x ] [ else y ] ) ) 11 17 )
17
> |
```



---

## Task 2

### Demo:

```

Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( 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
> ( caddr 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) )
> ( car ( cdr ( cdr ( cdr animals ) ) ) )
  animals: undefined;
cannot reference an identifier before its definition
> ( caddr pitches )
'fa
> ( list-ref pitches 3 )
'fa
> ( define a 'alligator )
  'alligator: undefined;
cannot reference an identifier before its definition
> ( define a 'alligator )
> ( define b 'pussycat )

> ( define c 'chimpanzee )
  'chimpanzee: undefined;
cannot reference an identifier before its definition
> ( 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 ) )
#<procedure:>>
'(1 one 2 two)
> ( append x y )
'(1 one 2 two)
> |

```

---

## Task 3

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:

```
Welcome to DrRacket, version 8.7 [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 )  
yellow  
(?): ( red orange yellow green blue indigo violet )  
blue  
(?): ( red orange yellow green blue indigo violet )  
indigo  
(?): ( red orange yellow green blue indigo violet )  
indigo  
(?): ( red orange yellow green blue indigo violet )  
indigo  
(?): ( aet ate eat eta tae tea )  
eat  
(?): ( aet ate eat eta tae tea )  
tae  
(?): ( aet ate eat eta tae tea )  
tea  
(?): ( aet ate eat eta tae tea )  
ate  
(?): ( aet ate eat eta tae tea )  
ate  
(?): ( aet ate eat eta tae tea )  
eta  
(?): ( 0 1 2 3 4 5 6 7 8 9 )  
6  
(?): ( 0 1 2 3 4 5 6 7 8 9 )  
2  
(?): ( 0 1 2 3 4 5 6 7 8 9 )  
0  
(?): ( 0 1 2 3 4 5 6 7 8 9 )  
1  
(?): ( 0 1 2 3 4 5 6 7 8 9 )  
7  
(?): ( 0 1 2 3 4 5 6 7 8 9 )  
9  
(?): : . . user break
```

---

## Task 4

Code:

```
( 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 )  
  
  )  
  
)
```

## Demo:

Welcome to [DrRacket](#), version 8.7 [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 C ))
#f
> (ranks 4)
'((4 C) (4 D) (4 H) (4 S))
> (ranks 'K)
❗❗ 'K: undefined;
  cannot reference an identifier before its definition
> (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 2
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) 2
(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)
'(5 S)
> (pick-a-card)
'(Q C)
> (pick-a-card)
'(5 C)
> (pick-a-card)
'(J C)
> (pick-a-card)
'(Q H)
> (pick-a-card)
'(A C)
>
```