Racket Programming Assignment #4: RLP and HoFs

Learning Abstract:

RLP is recursive list processing and HOFs are high order functions. There are a couple exercises to stretch recursive thinking while concerning list processing. High order functions are functions that accept a function as a parameter.

Task #1: Generate Uniform List

```
#lang racket

( define ( generate-uniform-list count item )
   ( cond
      ( ( = count 0 ) '() )
      ( else
        ( append ( list item ) ( generate-uniform-list ( - count 1 ) item ) )
      )
   )
)
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-uniform-list 5 'kitty )
'(kitty kitty kitty kitty kitty)
> ( generate-uniform-list 10 2 )
'(2 2 2 2 2 2 2 2 2 2)
>  ( generate-uniform-list 0 'whatever )
'()
> ( generate-uniform-list 2 '(racket prolog haskell rust) )
'((racket prolog haskell rust) (racket prolog haskell rust))
>
```

Task #2: Association List Generator

```
( define ( a-list list-1 list-2 )
   ( cond
      ( ( = ( length list-1 ) 0 ) null )
      ( else
        ( cons ( cons ( car list-1 ) ( car list-2 ) ) ( a-list ( cdr list-1 ) ( cdr list-2 ) ) )
      )
   )
)
```

Language: racket, with debugging; memory limit: 128 MB.

```
> ( a-list '(one two three four five) '(un deux trois quatre cinq) )
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
>  ( a-list '() '() )
'()
> ( a-list '( this ) '( that ) )
'((this . that))
>  ( a-list '(one two three) '( (1) (2 2) ( 3 3 3 ) ) )
'((one 1) (two 2 2) (three 3 3 3))
>
```

Task #3: Assoc

```
; i is an item in the list
; l is the list object
( define ( assoc i l )
   ( cond
      ( ( = ( length l ) 0 ) null )
      ( ( eq? ( car ( car l ) ) i ) ( car l ) )
      ( else
        ( assoc i ( cdr l ) ) )
      )
   )
)
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define all
     ( a-list '(one two three four ) '(un deux trois quatre ) )
  )
> ( define al2
( a-list '(one two three) '( (1) (2 2) (3 3 3) ) )
)
> all
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( assoc 'two all )
'(two . deux)
> ( assoc 'five all )
'()
> al2
'((one 1) (two 2 2) (three 3 3 3))
>  ( assoc 'three al2 )
'(three 3 3 3)
>  ( assoc 'four al2 )
'()
>
```

Task #4: Rassoc

```
; i is an item in the list
; l is the list object
( define ( rassoc i l )
   ( cond
      ( ( = ( length l ) 0 ) null )
      ( ( equal? ( cdr ( car l ) ) i ) ( car l ) )
      ( else
        ( rassoc i ( cdr l ) ) )
      )
   )
)
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define all
     ( a-list '(one two three four ) '(un deux trois quatre ) )
   )
> ( define al2
     ( a-list '(one two three) '( (1) (2 2) ( 3 3 3 ) ) )
  )
> all
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( rassoc 'three all )
'()
> ( rassoc 'trois all )
'(three . trois)
> al2
'((one 1) (two 2 2) (three 3 3 3))
> ( rassoc '(1) al2 )
'(one 1)
> ( rassoc '(3 3 3) al2 )
'(three 3 3 3)
> ( rassoc 1 al2 )
'()
>
```

Task #5: Los->s

```
; l is the list object
( define ( los->s l )
   ( cond
      ( ( eq? ( length l ) 0 ) "" )
      ( ( eq? ( length l ) 1 ) ( car l ) )
      ( else
        ( string-append ( car l ) " " ( los->s ( cdr l ) ) )
      )
   )
)
```

```
Language: racket, with debugging; memory limit: 128 MB.
> ( los->s '( "red" "yellow" "blue" "purple" ) )
"red yellow blue purple"
> ( los->s ( generate-uniform-list 20 "-" ) )
"- - - - - - - - - - - - - - - - - - - -"
> ( los->s '() )
""
> ( los->s '( "whatever" ) )
"whatever"
>
```

Task #6: Generate list

```
( define ( generate-list i list-func )
   ( cond
      ( ( eq? i 0 ) '() )
      ( else
        ( append ( list ( list-func ) ) ( generate-list ( - i 1 ) list-func ) )
      )
   )
)

; GIVEN CODE \/
( define ( roll-die ) ( + ( random 6 ) 1 ) )
( define ( dot )
   ( circle ( + 10 ( random 41 ) ) "solid" ( random-color ) )
)
( define ( big-dot )
   ( circle ( + 10 ( random 141 ) ) "solid" ( random-color ) )
)
( define ( random-color )
   ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) )
)
( define ( rgb-value )
   ( random 256 )
)
( define ( sort-dots loc )
   ( sort loc #:key image-width < )
)
; GIVEN CODE /\
```

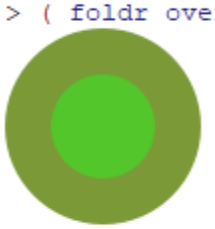
Demo 1:

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-list 10 roll-die )
'(3 1 5 3 5 2 6 6 1 2)
>  ( generate-list 20 roll-die )
'(1 4 4 4 5 2 3 6 1 5 4 3 2 1 4 6 2 6 1 1)
> ( generate-list 12
( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) )
)
'(blue yellow red yellow yellow blue yellow red yellow yellow red red)
>
```

Demo 2:

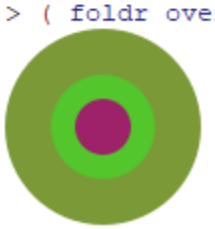
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define dots ( generate-list 3 dot ) )
> dots
```

```
(list                    )
> ( foldr overlay empty-image dots )
```



```
> ( sort-dots dots )
```

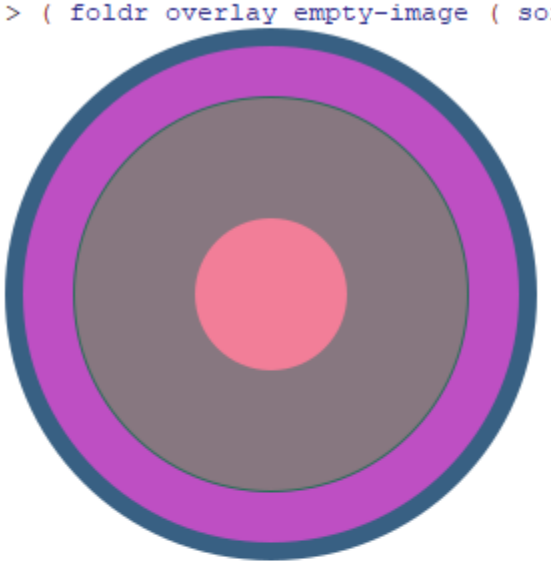
```
(list                    )
> ( foldr overlay empty-image ( sort-dots dots ) )
```



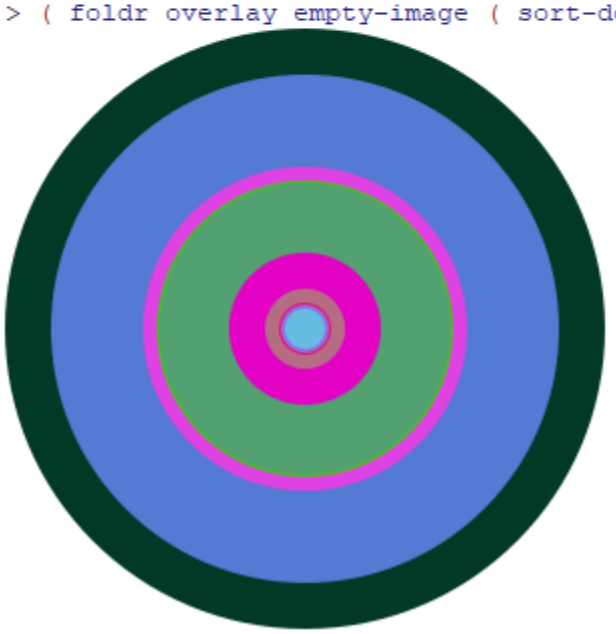
```
>
```

Demo 3:

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define a ( generate-list 5 big-dot ) )
> ( foldr overlay empty-image ( sort-dots a ) )
```



```
> ( define b ( generate-list 10 big-dot ) )
> ( foldr overlay empty-image ( sort-dots b ) )
```

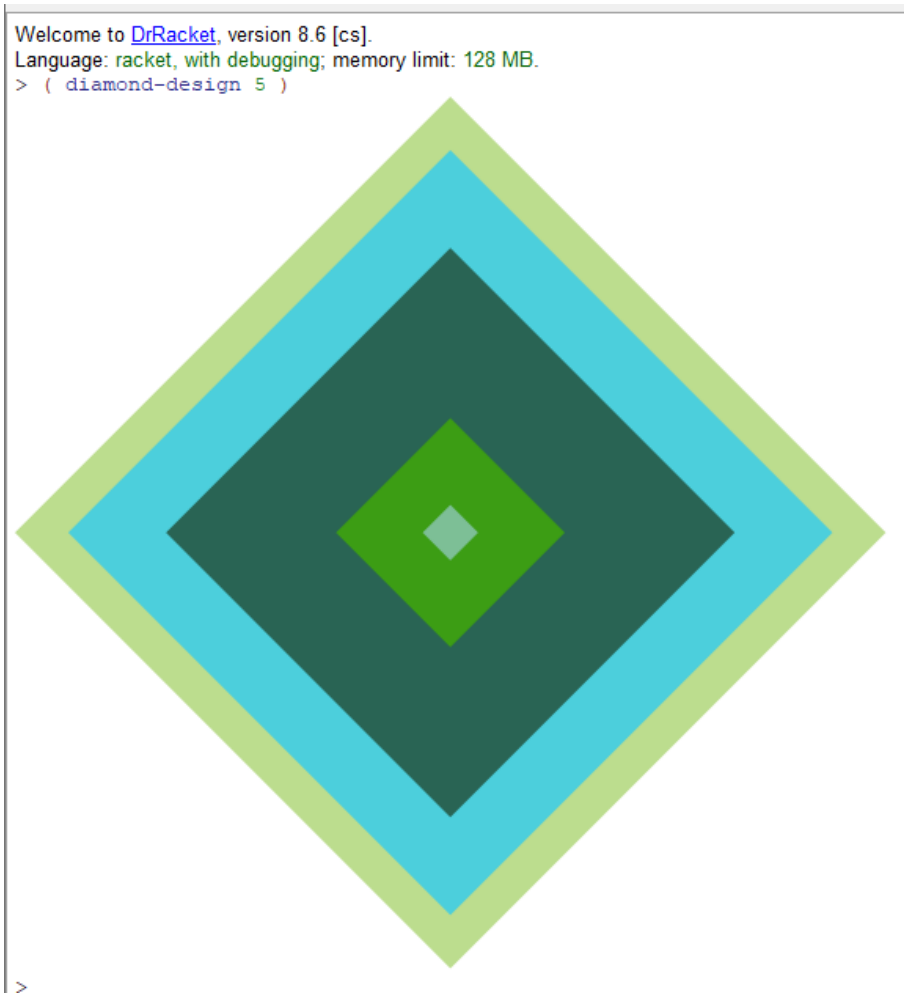


```
>
```

Task #7: The Diamond

```
5  ; HELPER FUNCTIONS
6  ; uses given random-color
7  ( define ( diamond )
8     ( define side ( + 20 ( random 380 ) ) )
9     ( define color ( random-color ) )
0     ( rotate 45 ( square side 'solid color ) )
1  )
2  ( define ( sort-diamond loc )
3     ( sort loc #:key image-width < )
4  )
5
6  ; uses generate-list /\ (ln.57)
7  ( define ( diamond-design i )
8     ( foldr overlay empty-image ( sort-diamond ( generate-list i diamond ) ) )
9  )
```

Demo 1:



Demo 2:

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 20 )
```

```
>
```

Task #8: Chromesthetic renderings

```
; GIVEN CODE \/
( define pitch-classes '( c d e f g a b ) )
( define color-names '( blue green brown purple red yellow orange ) )
( define ( box color )
   ( overlay
     ( square 30 "solid" color )
     ( square 35 "solid" "black" )
   )
)
( define boxes
   ( list
     ( box "blue" )
     ( box "green" )
     ( box "brown" )
     ( box "purple" )
     ( box "red" )
     ( box "gold" )
     ( box "orange" )
)
)
( define pc-a-list ( a-list pitch-classes color-names ) )
( define cb-a-list ( a-list color-names boxes ) )
( define ( pc->color pc )
   ( cdr ( assoc pc pc-a-list ) )
)
( define ( color->box color )
   ( cdr ( assoc color cb-a-list ) )
)
; GIVEN CODE /\

( define ( play pitch-lst )
   ( define color-lst ( map pc->color pitch-lst ) )
   ( define box-lst ( map color->box color-lst ) )
   ( foldr beside empty-image box-lst )
)
;test cases:
; ( play '( c d e f g a b c c c b g f e d c ) )
; ( play '( c c g g a a g g f f e e d d c c ) )
; ( play '( c d e c c d e c e f g g e f g g ) )
;( trace play )
```

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

```
> ( play '( c d e f g a b c c c b g f e d c ) )
```

```
> ( play '( c c g g a a g g f f e e d d c c ) )
```

```
> ( play '( c d e c c d e c e f g g e f g g ) )
```

```
>
```

Task #9: Diner

```
#lang racket
( require racket/trace )

; GIVEN CODE \/
( define menu '((fish . 8.5)
                (steak . 12.75)
                (burger . 5.48)
                (fries . 1)
                (salad . 1)
                (soup . 2.5)) )
( define sales '( burger fries salad burger salad fish soup fries burger soup steak fries
                         burger salad steak soup salad steak salad fish soup burger salad fish
                         salad fish soup burger fries steak salad fish fries steak salad steak
                         fries fish fries fish fries fries steak soup fish fries salad ) )
; GIVEN CODE /\

( define ( price item )
  ( cdar ( filter ( lambda ( menu-item ) ( equal? ( car menu-item ) item ) ) menu ) )
)
;( trace price )

( define ( total sold-items item )
   ( foldr + 0 ( map ( lambda ( sold-item )
                       ( cond
                          ( ( equal? sold-item item ) ( price sold-item ) )
                          ( else 0 )) )
                     sold-items ) )
)
;( trace total )
; TEST CASES
;; ( total sales 'hamburger )
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> menu
'((fish . 8.5) (steak . 12.75) (burger . 5.48) (fries . 1) (salad . 1) (soup . 2.5))
> sales
'(burger
  fries
  salad
  burger
  salad
  fish
  soup
  fries
  burger
  soup
  steak
  fries
  burger
  salad
  steak
  soup
  salad
  steak
  salad
  fish
  soup
  burger
  salad
  fish
  salad
  fish
  soup
  burger
  fries
  steak
  salad
  fish
  fries
  steak
  salad
  steak
  fries
  fish
  fries
  fish
  fries
  fries
  steak
  soup
  fish
  fries
  salad)
>
```

```
  salad)
> ( total sales 'fish )
68.0
> ( total sales 'steak )
89.25
> ( total sales 'burger )
32.88
> ( total sales 'fries )
10
> ( total sales 'salad )
10
> ( total sales 'soup )
15.0
>
```

Task #10: Grapheme Color Synesthesia

```
( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
( define DI (text "D" 36 "Crimson") )
( define EI (text "E" 36 "Maroon") )
( define FI (text "F" 36 "Pink") )
( define GI (text "G" 36 "Orchid") )
( define HI (text "H" 36 "Brown") )
( define II (text "I" 36 "Coral") )
( define JI (text "J" 36 "Orange") )
( define KI (text "K" 36 "Goldenrod") )
( define LI (text "L" 36 "Olive") )
( define MI (text "M" 36 "Lime") )
( define NI (text "N" 36 "Cyan") )
( define OI (text "O" 36 "Teal") )
( define PI (text "P" 36 "Navy") )
( define QI (text "Q" 36 "Indigo") )
( define RI (text "R" 36 "Purple") )
( define SI (text "S" 36 "Magenta") )
( define TI (text "T" 36 "Black") )
( define UI (text "U" 36 "Turquoise") )
( define VI (text "V" 36 "Wheat") )
( define WI (text "W" 36 "Salmon") )
( define XI (text "X" 36 "Orange Red") )
( define YI (text "Y" 36 "Aquamarine") )
( define ZI (text "Z" 36 "CornflowerBlue") )

( define alphabet '(A B C D E F G H I J K L M N O P Q R S T U V W X Y Z) )
( define alphapic ( list AI BI CI DI EI FI GI HI II JI KI LI MI
                         NI OI PI QI RI SI TI UI VI WI XI YI ZI) )

( define a->i ( a-list alphabet alphapic ) )

( define ( letter->image letter )
   ( cdr ( assoc letter a->i ) )
)

( define ( gcs letter-list )
   ( foldr beside empty-image ( map letter->image letter-list ) )
)
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> alphabet
'(A B C D E F G H I J K L M N O P Q R S T U V W X Y Z)
> alphapic
(list A B C D E F G H I J K L M N O P Q R S T U V W X Y Z)
> ( display a->i )
((A . A) (B . B) (C . C) (D . D) (E . E) (F . F) (G . G) (H . H) (I . I) (J . J) (K . K) (L . L) (M . M) (N . N) (O . O) (P . P) (Q . Q) (R . R) (S . S) (T . T) (U . U) (V . V) (W . W) (X . X) (Y . Y) (Z . Z))
> ( letter->image 'A )
A
> ( letter->image 'B )
B
> ( gcs '( C A B ) )
CAB
> ( gcs '( B A A ) )
BAA
> ( gcs '( B A B A ) )
BABA
>
```

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

```
> ( gcs '( D U S T I N ) )
```

DUSTIN

```
> ( gcs '( D A P H N E ) )
```

DAPHNE

```
> ( gcs '( C O N N O R ) )
```

CONNOR

```
> ( gcs '( K O R B Y N ) )
```

KORBYN

```
> ( gcs '( L Y L A ) )
```

LYLA

```
> ( gcs '( M C M A H O N ) )
```

MCMAHON

```
> ( gcs '( P E T E R S O N ) )
```

PETERSON

```
> ( gcs '( A L P H A B E T ) )
```

ALPHABET

```
> ( gcs '( D A N D E L I O N ) )
```

DANDELION

```
> ( gcs '( P I E ) )
```

PIE

```
>
```