

Racket Programming Assignment #4: RLP and HoFs

Bryan McLean

Learning Abstract:

This assignment is designed to give students practice with both recursion and higher order functions. It features five recursion problems and five problems that include higher order functions.

Task 1: Generate Uniform List

Code:

```
#lang racket
( define ( generate-uniform-list x ls )
  ( cond
    ( ( = x 0 ) ( list ) )

    ( else
      ( cons ls ( generate-uniform-list ( - x 1 ) ls ) )
    )
  )
)
```

Demo:

```
> ( 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

Code:

```
#lang racket
( define ( a-list list1 list2 )
  ( cond
    ( ( = ( length list1 ) 0 ) ( list ) )
    ( else
      ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr list1 ) ( cdr list2 ) ) )
    )
  )
)
```

Demo:

```
> ( 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

Code:

```
( define ( assoc element ls )
  ( cond
    ( ( empty? ls ) ( list ) )
    ( ( equal? element ( car ( car ls ) ) )
      ( car ls )
    )
    ( else
      ( assoc element ( cdr ls ) )
    )
  )
)
```

Demo:

```
> ( 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

Code:

```
( define ( rassoc element ls )
  ( cond
    ( ( empty? ls ) ( list ) )
    ( ( equal? element ( cdr ( car ls ) ) )
      ( car ls )
    )
    ( else
      ( rassoc element ( cdr ls ) )
    )
  )
)
```

Demo:

```
> ( 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

Code:

```
( define ( los->s ls )
  ( cond
    ( ( empty? ls ) "" )
    ( ( = ( length ls ) 1 ) ( car ls ) )
    ( else
      ( string-append ( car ls ) " " ( los->s ( cdr ls ) ) )
    )
  )
)
```

Demo:

```
> ( 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

Code:

```
( define ( generate-list x fun )
  ( cond
    ( ( = x 0 ) ( list ) )
    ( else
      ( cons ( fun ) ( generate-list ( - x 1 ) fun ) )
    )
  )
)
```

Demo 1:

```
> ( generate-list 10 roll-die )
'(1 1 1 2 5 6 2 2 4 4)
> ( generate-list 20 roll-die )
'(6 6 5 5 4 6 1 6 6 4 4 1 1 3 4 3 5 2 5 2)
> ( generate-list 12
  ( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) )
)
'(blue yellow yellow blue red red blue blue blue red red blue)
>
```

Demo 2:

```
> ( define dots ( generate-list 3 dot ) )
> dots
```



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

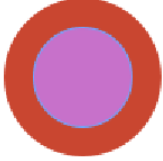


```
> ( sort-dots dots )
```



```
(list
```

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



```
>
```

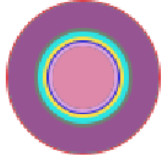
Demo 3:

```
> ( define a ( generate-list 5 dot ) )  
> ( foldr overlay empty-image ( sort-dots a ) )
```



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

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



```
> |
```

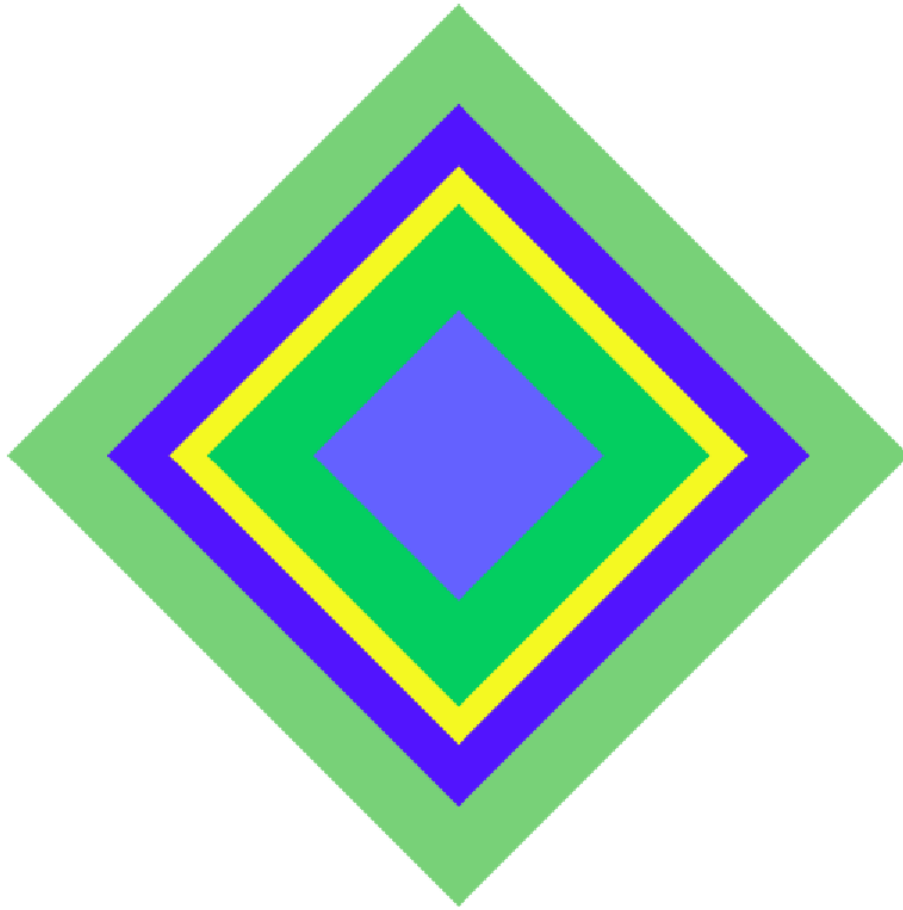
Task 7: The Diamond

Code:

```
( define ( diamond x )  
  ( define ( d ) ( rotate 45 ( square ( random 20 400 ) "solid" ( random-color ) ) ) )  
  ( define diamonds ( generate-list x d ) )  
  ( foldr overlay empty-image ( sort-dots diamonds ) )  
)
```

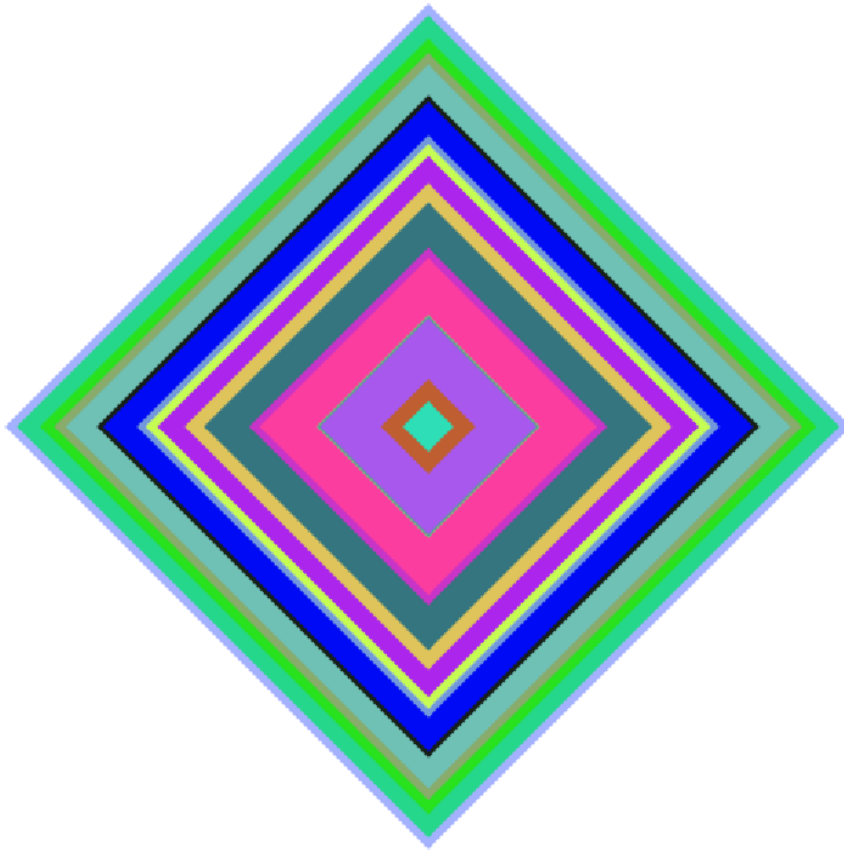
Demo 1:

```
> ( diamond 5 )
```



Demo 2:

```
> ( diamond 20 )
```





Task 8: Chromesthetic renderings

Code:

```
( define ( play ls )  
  ( foldr beside empty-image ( map box ( map pc->color ls ) ) )  
)
```

Demo:

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

Code:

```
#lang racket
( define ( a-list list1 list2 )
  ( cond
    ( ( = ( length list1 ) 0 ) ( list ) )
    ( else
      ( cons ( cons ( car list1 ) ( car list2 ) ) ( a-list ( cdr list1 ) ( cdr list2 ) ) )
    )
  )
)

( define items '( pizza pasta soda fries steak cake ) )

( define prices '( 15 12 2.5 3 17 5.5 ) )

( define menu ( a-list items prices ) )

( define sales '( pizza soda fries cake steak pizza pasta pasta soda cake fries steak pizza steak fries pasta pizza cake pasta
  cake soda pizza soda fries cake steak pizza pasta pasta soda ) )

( define ( items->prices items )
  ( cdr ( assoc items menu ) )
)

( define ( total sales items )
  ( define lookup ( filter ( lambda ( search ) ( eq? search items ) ) sales ) )
  ( cond
    ( ( eq? lookup 0 ) 0 )
    ( else
      ( define sold ( map items->prices lookup ) )
      ( foldr + 0 sold )
    )
  )
)
```

Demo:

```
> menu
'((pizza . 15) (pasta . 12) (soda . 2.5) (fries . 3) (steak . 17) (cake . 5.5))

> sales
'(pizza
soda
fries
cake
steak
pizza
pasta
pasta
soda
cake
fries
steak
pizza
steak
fries
pasta
pizza
cake
pasta
cake
soda
pizza
soda
fries)
```



```

cake
steak
pizza
pasta
pasta
soda)
> ( total sales 'pizza )
90
> ( total sales 'hotdog )
0
> ( total sales 'pasta )
72
> ( total sales 'soda )
12.5
> ( total sales 'steak )
68
>

```

Task 10: Grapheme Color Synesthesia

Code:

```

( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
( define DI (text "D" 36 "purple") )
( define EI (text "E" 36 "olivedrab") )
( define FI (text "F" 36 "pink") )
( define GI (text "G" 36 "tomato") )
( define HI (text "H" 36 "maroon") )
( define II (text "I" 36 "Peru") )
( define JI (text "J" 36 "gold") )
( define KI (text "K" 36 "lime") )
( define LI (text "L" 36 "turquoise") )
( define MI (text "M" 36 "cyan") )
( define NI (text "N" 36 "dodgerblue") )
( define OI (text "O" 36 "teal") )
( define PI (text "P" 36 "navy") )
( define QI (text "Q" 36 "slateblue") )
( define RI (text "R" 36 "magenta") )
( define SI (text "S" 36 "violet") )
( define TI (text "T" 36 "gray") )
( define UI (text "U" 36 "lavender") )
( define VI (text "V" 36 "azure") )
( define WI (text "W" 36 "cadetblue") )
( define XI (text "X" 36 "mintcream") )
( define YI (text "Y" 36 "lightgreen") )
( define ZI (text "Z" 36 "black") )

( 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 ls )
  ( cdr ( assoc ls a->i ) )
)

( define ( gcs ls )
  ( cond
    ( ( empty? ls ) ( empty-image ) )
    ( else
      ( foldr beside empty-image ( map letter->image ls ) )
    )
  )
)

```

Demo:

```
> alphabet
'(A B C)
> alphapic

(list A B C)
> ( display a->i )

((A . A) (B . B) (C . C))
> ( letter->image 'A )
A
> ( letter->image 'B )
B
> ( gcs '( C A B ) )
CAB
> ( gcs '( B A A ) )
BAA
> ( gcs '( B A B A ) )
BABA
>
```

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

ALPHABET

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

DANDELION

```
> ( gcs '( R A C K E T ) )
```

RACKET

```
> ( gcs '( F O O T B A L L ) )
```

FOOTBALL

```
> ( gcs '( L A U N D R Y ) )
```

LAUNDRY

```
> ( gcs '( M O U N T A I N ) )
```

MOUNTAIN

```
> ( gcs '( H A L L O W E E N ) )
```

HALLOWEEN

```
> ( gcs '( C O M P U T E R ) )
```

COMPUTER

```
> ( gcs '( I C I N G ) )
```

ICING

```
> ( gcs '( I M P A L A ) )
```

IMPALA

```
>
```