

## Racket Programming Assignment #4: RLP and HoFs

### **Learning Abstract**

For this assignment we continue on with recursion as well as adding the idea of less processing of higher order functions. Some of the tasks involved being able to produce different forms of word base list the others use the idea of making shapes from list. In addition, the last couple of tasks involved using synesthesia and portray how color and letters can intertwine with each other.

## Task 1 - Generate Uniform List

### Code

```
( define ( generate-uniform-list n item )
  ( cond
    (( > n 0 )
     ( cons item ( generate-uniform-list ( - n 1 ) item ) ))
    (else ( append '() ))
  )
)
```

### 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

```
( define ( a-list item-one item-two )
  ( cond ( ( empty? item-one )
    ( append '() ) )
    ( else ( cons ( cons ( car item-one ) ( car item-two ) ) ( a-list ( cdr item-one ) ( cdr item-one ) ) ) )
  )
)
```

### Demo

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

## Task 3 - Assoc

### Code

```
#lang racket
( define ( assoc item list )
  ( cond
    ( ( null? list ) null )
    ( ( equal? item ( car ( car list ) ) ) ( car list ) )
    ( else ( assoc item ( cdr list ) ) )
  )
)
```

### 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 . two) (three . three) (four . four))
> ( assoc 'two all )
'(two . two)
> ( assoc 'five all )
'()
> al2
'((one 1) (two . two) (three . three))
> ( assoc 'three al2 )
'(three . three)
> ( assoc 'four al2 )
'()
~
```

## Task 4 - Rassoc

### Code

```
( define ( rassoc obj list )
  ( cond
    ( ( null? list ) null )
    ( ( equal? list ( cdr ( car list ) ) ) ( car list ) )
    ( else ( rassoc list ( cdr list ) ) )
  )
)
```

### 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 . two) (three . three) (four . four))
> ( rassoc 'three all )
'(three . three)
> ( rassoc 'trois all )
'()
> al2
'((one 1) (two . two) (three . three))
> ( rassoc '(1) al2 )
'(one 1)
> ( rassoc '(3 3 3) al2 )
'()
> ( rassoc 1 al2 )
'()
>

```

## Task 5 - Los->s

### Code

```

( define ( los->s item )
  ( cond
    ( ( empty? item ) ( append "" ) )
    ( else ( string-append ( car item ) " " ( los->s ( cdr item ) ) ) )
  )
)

```

### 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

```

#lang racket
(require 2htdp/image)
( define ( roll-die ) ( + ( random 6 ) 1 ) )
( define ( dot )
  ( circle ( + 20 ( random 41 ) ) "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 < )
)

( define ( big-dot )
  ( circle ( + 20 ( random 100 ) ) "solid" ( random-color ) )
)
( define ( generate-list number thing )
  ( cond
    ( ( = number 0 ) ( append '() ) )
    ( else ( cons ( thing ) ( generate-list ( - number 1 ) thing ) ) )
  ) )

```

## Demo 1:

```

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

```

## Demo 2:

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



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



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



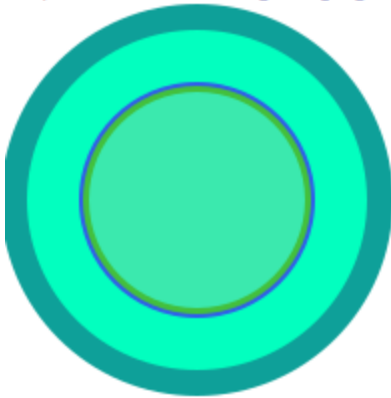
```
(list  
> ( foldr overlay empty-image ( sort-dots dots )
```



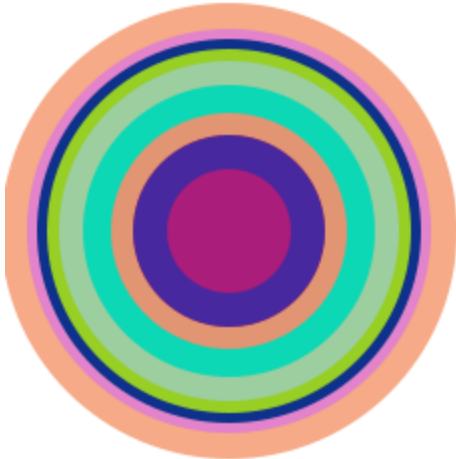
```
> |
```

Demo 3:

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



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



>

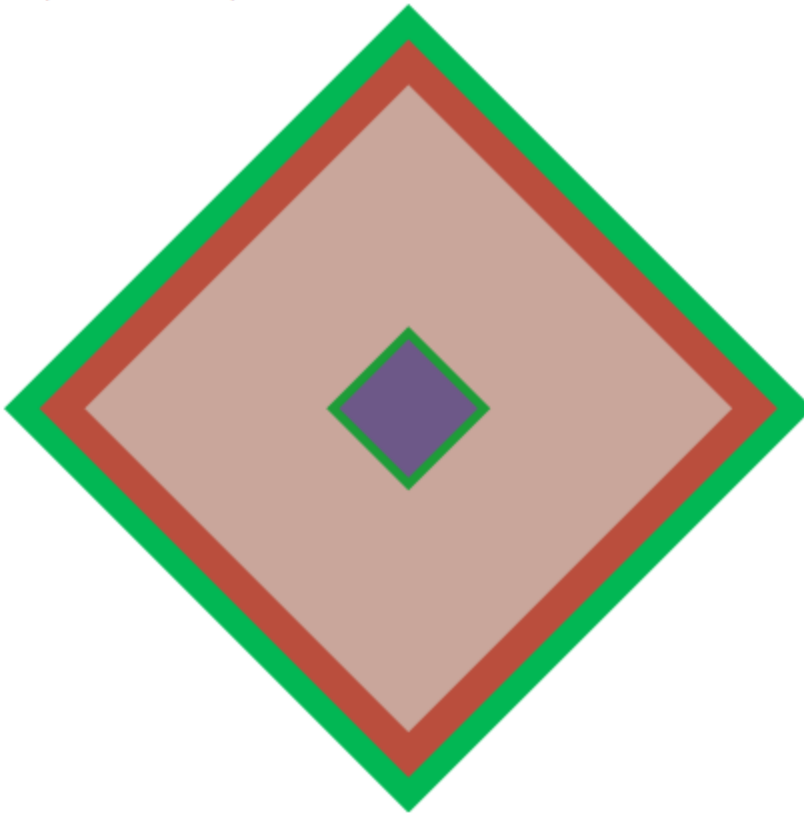
## Task 7 - The Diamond

### Code

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

### Demo 1

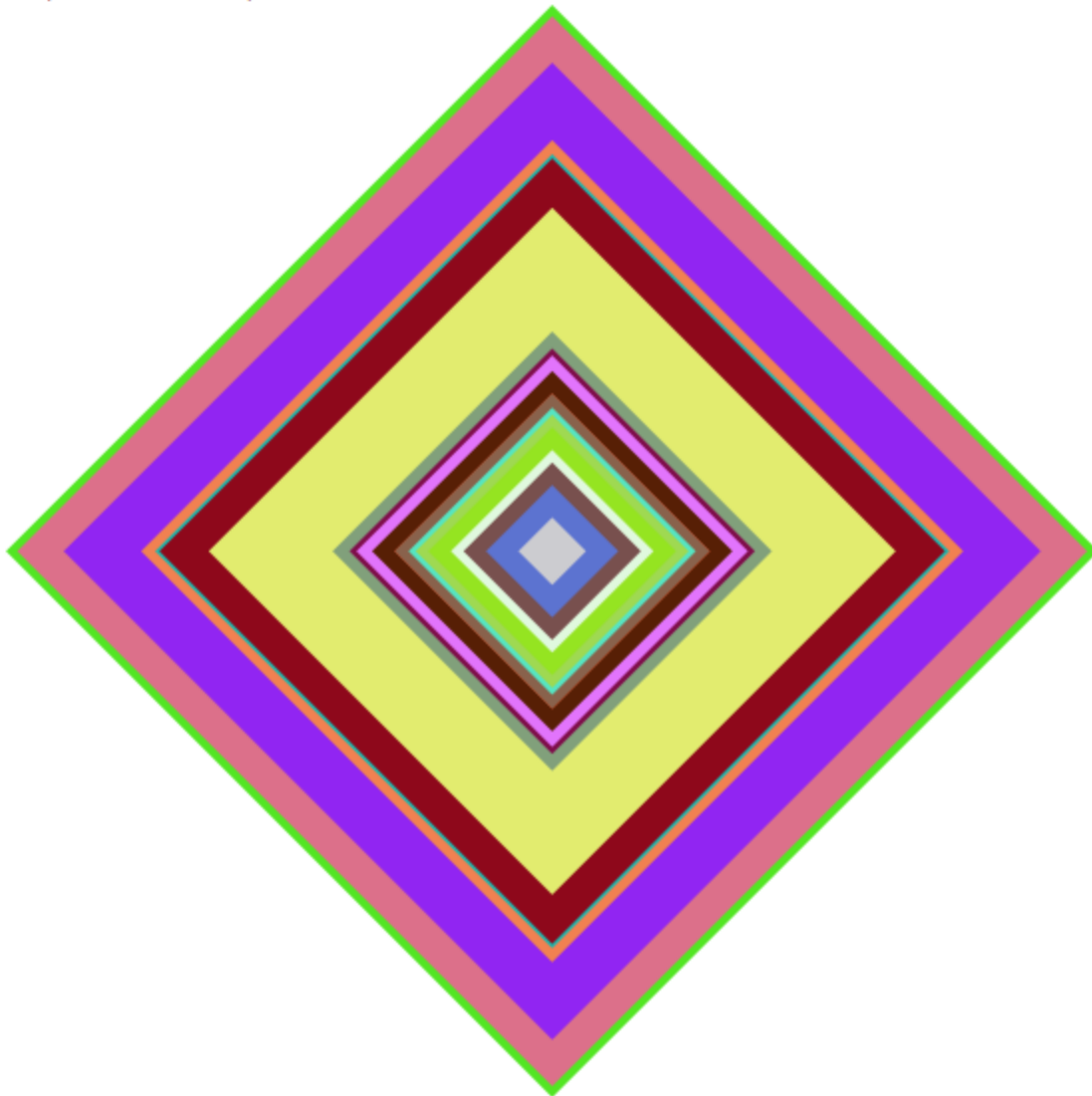
Language: Rust, with debugging, memory limit: 128 MB.  
> ( diamond 5 )



Demo 2



```
> ( diamond 20 )
```



```
>
```

## Task 8 - Chromesthetic renderings

Code




```

( define ( play pitch-list )
  ( define color-list ( map pc->color pitch-list ) )
  ( define box-list ( map color->box color-list ) )
  ( foldr beside empty-image box-list )
)
( 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 ) )
)

```

## 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

```

( define menu
'( ( apple . 1.50 )
  ( pear . 2 )
  ( orange . 3)
  ( pizza . 9 )
  ( pie . 10 )
  ( cake . 5)
)
)

( define sales
'(
apple
apple
apple
apple
apple|
pear
pear
pear
pear
pear
pear
orange
orange
orange
orange
orange
orange
pizza
pizza
pizza
pizza
pizza
pie
pie
pie
pie
pie
pie
cake
cake
cake
cake
cake
)
)

( define ( item->price item )
  ( cdr ( assoc item menu ) )
)

( define ( total pricetotal menuitem )
  ( define occurrences ( filter ( lambda (l) ( eq? l menuitem ) ) pricetotal ) )
  ( define pricelist ( map item->price occurrences ) )
  ( foldr + 0 pricelist )
)

```

## Demo

```
> menu
'((apple . 1.5) (pear . 2) (orange . 3) (pizza . 9) (pie . 10) (cake . 5))
> sales
'(apple apple apple apple apple pear pear pear pear pear orange orange orange orange orange pizza pizza pizza pizza pizza pie pie pie pie pie cake cake cake cake
cake)
> ( total sales 'apple )
7.5
> ( total sales 'pear )
10
> ( total sales 'orange )
15
> ( total sales 'pizza )
45
> ( total sales 'pie )
50
> ( total sales 'cake )
25
~
```

## 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 "salmon") )
( define EI (text "E" 36 "olive") )
( define FI (text "F" 36 "tan") )
( define GI (text "G" 36 "chartreuse") )
( define HI (text "H" 36 "navy") )
( define II (text "I" 36 "indigo") )
( define JI (text "J" 36 "plum") )
( define KI (text "K" 36 "violet") )
( define LI (text "L" 36 "magenta") )
( define MI (text "M" 36 "gray") )
( define NI (text "N" 36 "lime") )
( define OI (text "O" 36 "maroon") )
( define PI (text "P" 36 "peru") )
( define QI (text "Q" 36 "thistle") )
( define RI (text "R" 36 "green") )
( define SI (text "S" 36 "bieve") )
( define TI (text "T" 36 "firebrick") )
( define UI (text "U" 36 "honeydew") )
( define VI (text "V" 36 "goldenrod") )
( define WI (text "W" 36 "burlywood") )
( define XI (text "X" 36 "gold") )
( define YI (text "Y" 36 "khaki") )
( define ZI (text "Z" 36 "hotpink") )
( 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 )
  ( define image-list ( map letter->image letter-list ) )
  ( foldr beside empty-image image-list )
)

```

## Demo 1

```

> 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,
  ( (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
  < |

```

## Demo 2

```
> ( gcs '( ALPHABET ) )  
ALPHABET  
> ( gcs '( DANDELION ) )  
DANDELION  
> ( gcs '( CHEMISTRY ) )  
CHEMISTRY  
> ( gcs '( BIOCHEMISTRY ) )  
BIOCHEMISTRY  
> ( gcs '( LAMBDA ) )  
LAMBDA  
> ( gcs '( MATH ) )  
MATH  
> |
```

```
> ( gcs '( CODING ) )  
CODING  
> ( gcs '( RACKET ) )  
RACKET  
> ( gcs '( PYTHON ) )  
PYTHON  
> ( gcs '( CALCULUS ) )  
CALCULUS  
> ( gcs '( OSWEGO ) )  
OSWEGO  
> ( gcs '( RAVEN ) )  
RAVEN  
>
```