Racket Programming Assignment #4: RLP and HoFs

Learning Abstract

This assignment emphasizes recursive list processing and list processing with Higher Order Functions. Each task in this assignment builds off the previously completed task. In addition to the list processing exercises racket functions map, foldr are regularly used.

Task 1: Generate Uniform List

```
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 1 – Code

```
#lang racket
( define ( generate-uniform-list n obj )
   ( cond
      [( > n 0)( cons obj ( generate-uniform-list ( - n 1 ) obj ) )]
      [else ( append '() )]
   )
)
```

Task 2: Association List Generator

```
Welcome to DrRacket, version 8.6 [cs].
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 2 – Code

```
( define ( a-list list1 list2 )
   ( cond
      [( empty? list1 )( append '() )]
      [else ( cons ( cons ( car list1 ) ( car list2 ) )( a-list ( cdr list1 ) ( cdr list2 ) ) )]
   )
)
```

Task 3: Assoc

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define al1
( a-list '(one two three four ) '(un deux trois quatre ) )
)
>  ( define al2
( a-list '(one two three) '( (1) (2 2) (3 3 3) ) )
)
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( assoc 'two al1 )
'(two . deux)
> ( assoc 'five al1 )
'()
> al2
'((one 1) (two 2 2) (three 3 3 3))
> ( assoc 'three al2 )
'(three 3 3 3)
> ( assoc 'four al2 )
'()
>
```

Code ...

```
( define ( assoc obj l )
   ( cond
      [( empty? l ) ( append '() )]
      [( equal? obj ( car ( car l ) ) )(car l)]
      [else ( assoc obj ( cdr l ) )]
   )
)
```

Task 4: Rassoc

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define al1
( a-list '(one two three four ) '(un deux trois quatre ) )
)
> ( define al2
( a-list '(one two three) '( (1) (2 2) ( 3 3 3 ) ) )
)
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( rassoc 'three al1 )
'()
> ( rassoc 'trois al1 )
'(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 4 – Code

```
( define ( rassoc obj l )
   ( cond
      [( empty? l ) ( append '() )]
      [( equal? obj ( cdr ( car l ) ) )(car l)]
      [else ( rassoc obj ( cdr l ) )]
   )
)
```

Task 5: Los->s

```
( define ( los->s l )
   ( cond
      [( empty? l ) ( append "" )]
      [else ( string-append ( car l ) " " ( los->s ( cdr l ) ) )]
   )
)
```

Task 5 – Code

```
Welcome to DrRacket, version 8.6 [cs].
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 "
>
```

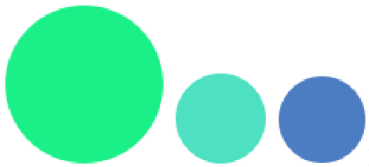
Task 6: Generate list

Demo 1…

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

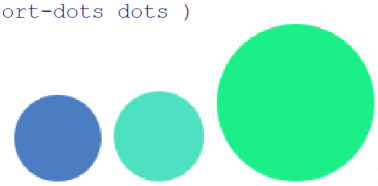
Demo 2…

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



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

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



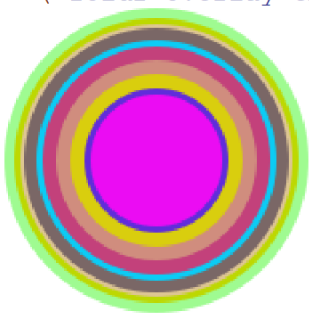
```
(list                    )
> ( foldr overlay empty-image ( sort-dots dots ))
```

```
>
```

Demo 3…

```
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 ))
```



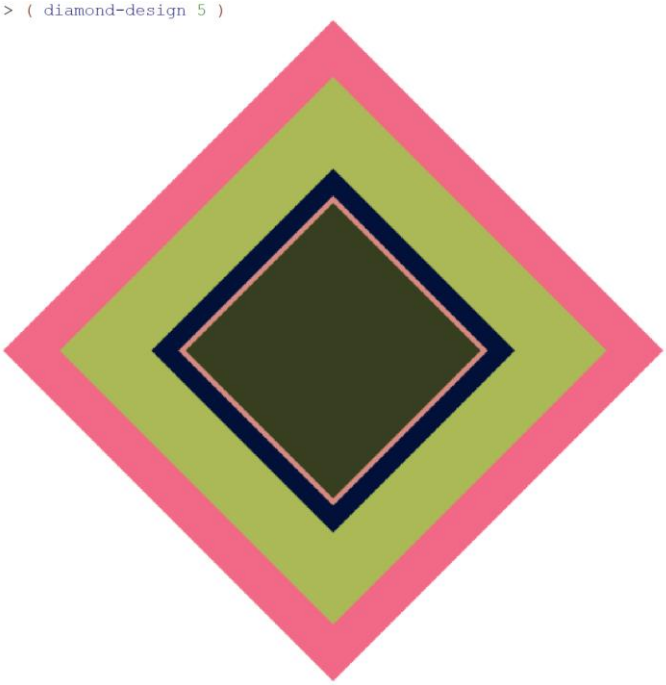
```
>
```

Task 6 – Code

```
( define ( big-dot )
( circle ( + 20 ( random 81 ) ) "solid" ( random-color ) )
)
( define ( generate-list n foo )
   ( cond
      [( = n 0 ) ( append '() )]
      [else ( cons ( foo ) ( generate-list ( - n 1 ) foo ))]
   )
)
```

Task 7: The Diamond

```
> ( diamond-design 5 )
```



```
> ( diamond-design 20 )
```

Task 7 – Code

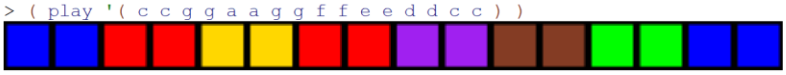
```
( define ( diamond )
   ( rotate 45 ( square ( + 20 ( random 381 ) ) 'solid ( random-color ) ) )
)
( define ( sort-diamond loc )
   ( sort loc #:key image-width < )
)
( define ( diamond-design n )
   ( define d ( generate-list n diamond ) )
   ( foldr overlay empty-image ( sort-diamond d ) )
)

```

Task 8: Chromesthetic renderings

```
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 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 8 – Code

```
;STEP.1 - For each note in notes <-- use map
;find the match in the pc-a-list and return the tail of the of the reulting pair <-- use cdr ( assoc )
;STEP.2 - For each result in Step.1
;find the match in cb-a-list and return the tail of the of the reulting pair
;STEP.3 - Print results from STEP.2 side-by-side <-- use foldr
;Need helper funtions to assoc to a note grabed from map on note
( define ( play notes )
   ( cond
      [( empty? notes )( empty-image )]
      [else ( foldr beside empty-image
                  ( map getBox
                        ( map getColor notes )
                  )
            )]
   )
)
( define ( getColor n )
   ( cdr ( assoc n pc-a-list ) )
)
( define ( getBox n )
   ( cdr ( assoc n cb-a-list ) )
)
```

Task 9: Diner

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( total sales 'Croissant )
45
> ( total sales 'Macaroon )
25.8
> ( total sales 'CheeseCake )
36
> ( total sales 'Muffin )
24.8
> ( total sales 'Tiramisu )
28
> ( total sales 'Scone )
11.7
>
```

Task 9 – Code

```
( define ( total sales item )
   ( foldr + 0
    ( map getPrice
     ( filter ( lambda (s) ( cond
                              [(equal? item s ) #t ]
                              [else #f ]
                            ) ) sales ) ) )
)
( define menue ( a-list '( Croissant Macaroon CheeseCake Muffin Tiramisu Scone ) '( 5 4.3 9 6.2 7 3.9 ) ) )
( define sales '( Croissant
                  Muffin
                  Macaroon
                  Scone
                  Scone
                  Muffin
                  Macaroon
                  Croissant
                  Croissant
                  CheeseCake
                  Macaroon
                  Croissant
                  Muffin
                  Tiramisu
                  Croissant
                  CheeseCake
                  Scone
                  Croissant
                  Macaroon
                  Muffin
                  Croissant
                  Macaroon
                  CheeseCake
                  Croissant
                  Croissant
                  Macaroon
                  Tiramisu
                  Tiramisu
                  CheeseCake
                  Tiramisu
                )
)
( define ( getPrice item )
   ( cdr ( assoc item menue ) )
)
```

Task 10: Grapheme Color Synesthesia

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

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( 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 '( O R C H I D ) )
ORCHID
> ( gcs '( P R O L O G ) )
PROLOG
> ( gcs '( J A V A ) )
JAVA
> ( gcs '( C H E F ) )
CHEF
> ( gcs '( X Y L O P H O N E ) )
XYLOPHONE
> ( gcs '( M O U N T A I N ) )
MOUNTAIN
> ( gcs '( S U P E R C A L I F R A G I L I S T I C E X P I A L I D O C I O U S ) )
SUPERCALIFRAGILISTICEXPIALIDOCIOUS
>
```

Task 10 – Code

```
;----------Task.10-------------------------------------------------------
( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
;--------------------------------------------
( define DI (text "D" 36 "Medium Violet Red") )
( define EI (text "E" 36 "Light Coral") )
( define FI (text "F" 36 "Orchid") )
( define GI (text "G" 36 "Lavender Blush") )
( define HI (text "H" 36 "Chocolate") )
( define II (text "I" 36 "Saddle Brown") )
( define JI (text "J" 36 "Coral") )
( define KI (text "K" 36 "Salmon") )
( define LI (text "L" 36 "Yellow") )
( define MI (text "M" 36 "Olive") )
( define NI (text "N" 36 "Burlywood") )
( define OI (text "O" 36 "Khaki") )
( define PI (text "P" 36 "Dark Khaki") )
( define QI (text "Q" 36 "Green") )
( define RI (text "R" 36 "Olive Drab") )
( define SI (text "S" 36 "Turquoise") )
( define TI (text "T" 36 "Light Sea Green") )
( define UI (text "U" 36 "Dark Blue") )
( define VI (text "V" 36 "Indigo") )
( define WI (text "W" 36 "Blue Violet") )
( define XI (text "X" 36 "Medium Purple") )
( define YI (text "Y" 36 "Cornflower Blue") )
( define ZI (text "Z" 36 "Dark Gray") )
;--------------------------------------------
( 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 ) )
;---^ Given ^-------------------------------
( define ( letter->image x )
   ( cdr ( assoc x a->i ) )
)
( define ( gcs l )
   ( cond
      [( empty? l )( empty-image )]
      [else( foldr beside empty-image ( map letter->image l ) )]
   )
)
```