Racket Programming Assignment #5: RLP and HoFs

Learning Abstract

This assignment helped us understand RLP and HoFs in the Racket programming language. There were certain parts in this assignment where my problem-solving skills were tested and it was really tough and fun finding the solutions. I now understand and appreciate the usefulness and importance of RLP and HoFs in programming. Overall, this was an important assignment since we definitely felt challenged and learned a lot.

Task 1 - Simple List Generators

Task 1a - iota

Function Definition

```
#lang racket
( define ( iota n )
   ( define ( inc i )
      ( if ( <= i n )
           ( cons i ( inc ( + i 1 ) ) ) '()
           )
      )
   (inc 1 )
   )
```

Demo

```
> ( iota 10 )
'(1 2 3 4 5 6 7 8 9 10)
> ( iota 1 )
'(1)
> ( iota 12 )
'(1 2 3 4 5 6 7 8 9 10 11 12)
> 
```

Task 1b - Same

Function Definition

```
( define ( same n list )
   ( cond ( ( <= n 0 ) '() )
          ( ( > n 0 ) ( cons list ( same (- n 1 ) list ) ) )
          )
   )
```

Demo

```
> ( same 5 'five )
'(five five five five five)
> ( same 10 2 )
'(2 2 2 2 2 2 2 2 2 2)
> ( same 0 'whatever )
'()
> ( same 2 '(racket prolog haskell rust ) )
'((racket prolog haskell rust) (racket prolog haskell rust))
> |
```

Task 1c - Alternator

Function Definition

```
( define ( alternator i l )
   ( cond
      ( ( <= i 0 ) '() )
      ( ( > i 0 ) ( cons ( car l )
                         ( alternator ( - i 1 ) ( append ( cdr l ) ( list ( car l ) ) ) )
                         )
                  )
      )
   )
```

Demo

```
> ( alternator 7 '( black white ) )
'(black white black white black white black)
> ( alternator 12 '( red yellow blue ) )
'(red yellow blue red yellow blue red yellow blue red yellow blue)
> ( alternator 9 '( 1 2 3 4 ) )
'(1 2 3 4 1 2 3 4 1)
> ( alternator 15 '( x y ) )
'(x y x y x y x y x y x y x y x)
>
```

Task 1d - Sequence

Function Definition

```
( define ( sequence i num )
   ( cond
      ( ( <= num 0 ) '() )
      ( else ( map ( lambda (x) ( * x num ) ) ( iota i ) ) )
      )
   )
```

Demo

```
> ( sequence 5 20 )
'(20 40 60 80 100)
> ( sequence 10 7 )
'(7 14 21 28 35 42 49 56 63 70)
> ( sequence 8 50 )
'(50 100 150 200 250 300 350 400)
>
```

Task 2 - Counting

Task 2a - Accumulation Counting

Function Definition

```
( define ( a-count l )
   ( cond ( ( null? l ) '())
          ( else ( append ( iota ( car l ) ) ( a-count ( cdr l ) ) ) )
          )
   )
```

Demo

```
> ( a-count '( 1 2 3 ) )
'(1 1 2 1 2 3)
> ( a-count '( 4 3 2 1 ) )
'(1 2 3 4 1 2 3 1 2 1)
> ( a-count '( 1 1 2 2 3 3 2 2 1 1 ) )
'(1 1 1 2 1 2 1 2 3 1 2 3 1 2 1 2 1 1)
>
```

Task 2b - Repetition Counting

Function Definition

```
( define ( r-count l )
   ( cond ( ( null? l ) '() )
          ( else ( append ( same ( car l ) ( car l ) ) ( r-count ( cdr l ) ) ) )
          )
   )
```

Demo

```
> ( r-count '( 1 2 3 ) )
'(1 2 2 3 3 3)
> ( r-count '( 4 3 2 1 ) )
'(4 4 4 4 3 3 3 2 2 1)
> ( r-count '( 1 1 2 2 3 3 2 2 1 1 ) )
'(1 1 2 2 2 2 3 3 3 3 3 3 2 2 2 2 1 1)
>
```

Task 2c - Mixed Counting Demo

Demo

```
> ( a-count '( 1 2 3 ) )
'(1 1 2 1 2 3)
> ( r-count '( 1 2 3 ) )
'(1 2 2 3 3 3)
> ( r-count ( a-count '( 1 2 3 ) ) )
'(1 1 2 2 1 2 2 3 3 3)
> ( a-count ( r-count '( 1 2 3 ) ) )
'(1 1 2 1 2 1 2 3 1 2 3 1 2 3)
> ( a-count '( 2 2 5 3 ) )
'(1 2 1 2 1 2 3 4 5 1 2 3)
> ( r-count '( 2 2 5 3 ) )
'(2 2 2 2 5 5 5 5 5 3 3 3)
> ( r-count ( a-count '( 2 2 5 3 ) ) )
'(1 2 2 1 2 2 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 1 2 2 3 3 3)
> ( a-count ( r-count '(2 2 5 3 ) ) )
'(1 2 1 2 1 2 1 2 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 1 2 3 1 2 3)
>
```

Task 3a - Zip

Function Definition

```
( define ( zip list-a list-b )
   ( cond ( ( null? list-a ) '() )
          ( else ( cons ( cons ( car list-a ) ( car list-b ) ) ( zip ( cdr list-a ) ( cdr list-b ) ) )
                 )
          )
   )
```

Demo

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

Task 3b - Assoc

Function Definition

```
( define ( assoc objl assl )
   ( cond ( ( null? assl ) '() )
          ( ( eq? ( caar assl ) objl ) ( car assl ) )
          ( else ( assoc objl ( cdr assl ) ) )
         )
   )
```

Demo

```
>  ( define al1
     ( zip '( one two three four ) '( un deux trois quatre ) )
     )
> ( define al2
     ( zip '( 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 )
'()
>
```

Task 3c - Establishing some Association Lists

Code

```
( define scale-zip-CM
( zip ( iota 7 ) '("C" "D" "E" "F" "G" "A" "B") )
)

( define scale-zip-short-Am
( zip ( iota 7 ) '("A/2" "B/2" "C/2" "D/2" "E/2" "F/2" "G/2") )
)

( define scale-zip-short-low-Am
( zip ( iota 7 ) '("A,/2" "B,/2" "C,/2" "D,/2" "E,/2" "F,/2" "G,/2") )
)

( define scale-zip-short-low-blues-Dm
( zip ( iota 7 ) '( "D,/2" "F,/2" "G,/2" "_A,/2" "A,/2" "c,/2" "d,/2" ) )
)

( define scale-zip-wholetone-C
( zip ( iota 7 ) '("C" "D" "E" "^F" "^G" "^A" "c") )
)
```

Demo

```
> scale-zip-short-low-Am
'((1 . "A,/2") (2 . "B,/2") (3 . "C,/2") (4 . "D,/2") (5 . "E,/2") (6 . "F,/2") (7 . "G,/2"))
> scale-zip-CM
'((1 . "C") (2 . "D") (3 . "E") (4 . "F") (5 . "G") (6 . "A") (7 . "B"))
> scale-zip-short-Am
'((1 . "A/2") (2 . "B/2") (3 . "C/2") (4 . "D/2") (5 . "E/2") (6 . "F/2") (7 . "G/2"))
> scale-zip-short-low-Am
'((1 . "A,/2") (2 . "B,/2") (3 . "C,/2") (4 . "D,/2") (5 . "E,/2") (6 . "F,/2") (7 . "G,/2"))
> scale-zip-short-low-blues-Dm
'((1 . "D,/2") (2 . "F,/2") (3 . "G,/2") (4 . "_A,/2") (5 . "A,/2") (6 . "c,/2") (7 . "d,/2"))
> scale-zip-wholetone-C
'((1 . "C") (2 . "D") (3 . "E") (4 . "^F") (5 . "^G") (6 . "^A") (7 . "c"))
>
```

Task 4a - nr->note

Function Definition

```
( define ( nr->note n assl )
   ( cond ( ( null? assl ) "" )
          ( ( eq? n ( caar assl ) ) ( cdar assl ) )
          ( else ( nr->note n ( cdr assl ) ) )
          )
   )
```

Demo

```
>  ( nr->note 1 scale-zip-CM )
"C"
>  ( nr->note 1 scale-zip-short-Am )
"A/2"
> ( nr->note 1 scale-zip-short-low-Am )
"A,/2"
> ( nr->note 3 scale-zip-CM )
"E"
>  ( nr->note 4 scale-zip-short-Am )
"D/2"
> ( nr->note 5 scale-zip-short-low-Am )
"E,/2"
> ( nr->note 4 scale-zip-short-low-blues-Dm )
"_A,/2"
> ( nr->note 4 scale-zip-wholetone-C )
"^F"
>
```

Task 4b - nrs->notes

Function Definition

```
( define ( nrs->notes n assl )
   ( cond ( ( null? assl ) "" )
          ( else ( map ( lambda (x) ( nr->note x assl ) ) n ) )
          )
   )
```

Demo

```
> ( nrs->notes '(3 2 3 2 1 1) scale-zip-CM )
'("E" "D" "E" "D" "C" "C")
> ( nrs->notes '(3 2 3 2 1 1) scale-zip-short-Am )
'("C/2" "B/2" "C/2" "B/2" "A/2" "A/2")
>  ( nrs->notes ( iota 7 ) scale-zip-CM )
'("C" "D" "E" "F" "G" "A" "B")
> ( nrs->notes ( iota 7 ) scale-zip-short-low-Am )
'("A,/2" "B,/2" "C,/2" "D,/2" "E,/2" "F,/2" "G,/2")
> ( nrs->notes ( a-count '(4 3 2 1) ) scale-zip-CM )
'("C" "D" "E" "F" "C" "D" "E" "C" "D" "C")
> ( nrs->notes ( r-count '(4 3 2 1) ) scale-zip-CM )
'("F" "F" "F" "F" "E" "E" "E" "D" "D" "C")
>  ( nrs->notes ( a-count ( r-count '(1 2 3) ) ) scale-zip-CM )
'("C" "C" "D" "C" "D" "C" "D" "E" "C" "D" "E" "C" "D" "E")
>  ( nrs->notes ( r-count ( r-count '(1 2 3) ) ) scale-zip-CM )
'("C" "D" "D" "D" "D" "E" "E" "E" "E" "E" "E" "E" "E" "E")
>
```

Task 4c - nrs->abc

Function Definition

```
( define ( nrs->abs n assl )
   ( string-join ( nrs->notes n assl )
                 )
   )
```

Demo

```
> ( nrs->abc ( iota 7 ) scale-zip-CM )
"C D E F G A B"
> ( nrs->abc ( iota 7 ) scale-zip-short-Am )
"A/2 B/2 C/2 D/2 E/2 F/2 G/2"
> ( nrs->abc ( a-count '( 3 2 1 3 2 1 ) ) scale-zip-CM )
"C D E C D C C D E C D C"
> ( nrs->abc ( r-count '( 3 2 1 3 2 1 ) ) scale-zip-CM )
"E E E D D C E E E D D C"
> ( nrs->abc ( r-count ( a-count '(4 3 2 1) ) ) scale-zip-CM )
"C D D E E E F F F F C D D E E E C D D C"
>  ( nrs->abc ( a-count ( r-count '(4 3 2 1) ) ) scale-zip-CM )
"C D E F C D E F C D E F C D E F C D E C D E C D E C D C D C"
> |
```

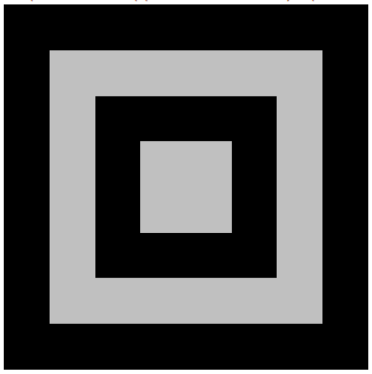
Task 5 - Stella

Function Definition

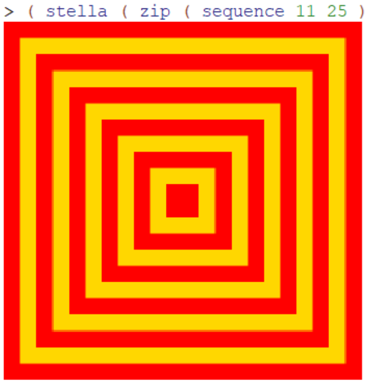
```
( define ( stella assl )
   ( foldr overlay empty-image
           ( map ( lambda (x) ( square ( car x ) "solid" ( cdr x ) ) ) assl )
          )
   )
```

The Five Demos

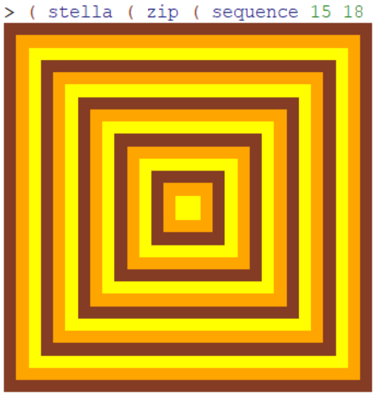
```
> ( stella '(( 70 . silver ) (140 . black) ( 210 . silver ) (280 . black ) ) )
```



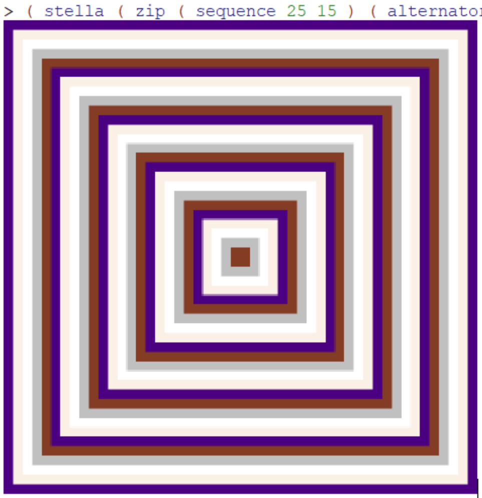
```
> ( stella ( zip ( sequence 11 25 ) (alternator 11 '( red gold ) ) ) )
```



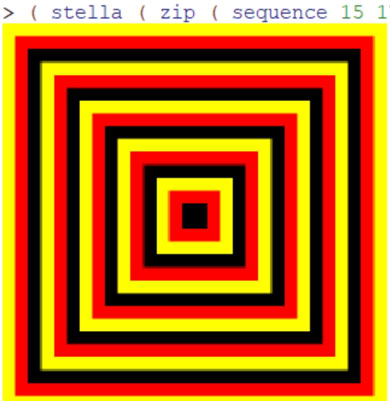
```
> ( stella ( zip ( sequence 15 18 ) (alternator 15 '( yellow orange brown ) ) ) )
```



```
> ( stella ( zip ( sequence 25 15 ) ( alternator 25 '( brown silver white linen indigo ) ) ) )
```



```
> ( stella ( zip ( sequence 15 17 )  ( alternator 15 '( black red yellow ))))
```



```
>
```

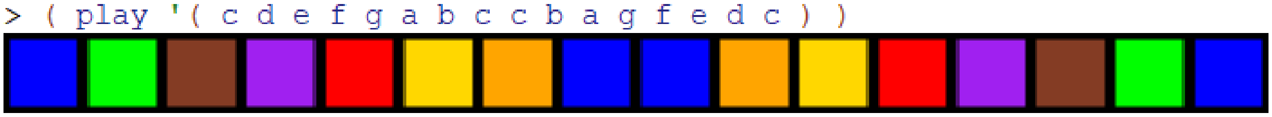
Task 6 - Chromesthetic Renderings

Code

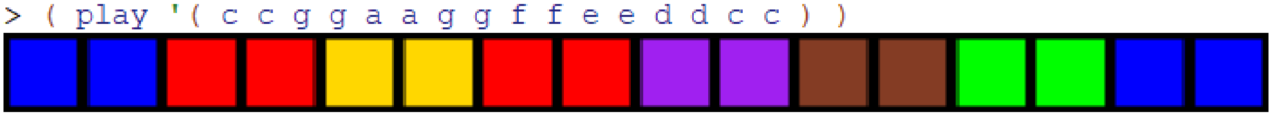
```
( define pitch-classes '( c d e f g a b ) )
( define color-names '( blue green brown purple red yellow orange ) )
( define ( box color )
   ( overlay
     ( square 30 "solid" color )
     ( square 35 "solid" "black" )
     )
   )
( define boxes
   ( list
     ( box "blue" )
     ( box "green" )
     ( box "brown" )
     ( box "purple" )
     ( box "red" )
     ( box "gold" )
     ( box "orange" )
     )
   )

( define pc-a-list ( zip pitch-classes color-names ) )
( define cb-a-list ( zip color-names boxes ) )

( define ( pc->color pc )
   ( cdr ( assoc pc pc-a-list ) )
   )
( define ( color->box color )
   ( cdr ( assoc color cb-a-list ) )
   )

( 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 )
   )
```

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 7 - Grapheme to 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 "firebrick") )
( define EI (text "E" 36 "crimson") )
( define FI (text "F" 36 "orange red") )
( define GI (text "G" 36 "deep pink") )
( define HI (text "H" 36 "chocolate") )
( define II (text "I" 36 "maroon") )
( define JI (text "J" 36 "indian red") )
( define KI (text "K" 36 "snow") )
( define LI (text "L" 36 "sienna") )
( define MI (text "M" 36 "indigo") )
( define NI (text "N" 36 "gold") )
( define OI (text "O" 36 "olive") )
( define PI (text "P" 36 "peach puff") )
( define QI (text "Q" 36 "lawn green") )
( define RI (text "R" 36 "khaki") )
( define SI (text "S" 36 "royal blue") )
( define TI (text "T" 36 "aqua") )
( define UI (text "U" 36 "turquoise") )
( define VI (text "V" 36 "cadet blue") )
( define WI (text "W" 36 "midnight blue") )
( define XI (text "X" 36 "brown") )
( define YI (text "Y" 36 "dark violet") )
( define ZI (text "Z" 36 "magenta") )

( 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 ( zip alphabet alphapic ) )

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

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

Demo 1

```
> 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
>
```

Demo 2

```
> ( gcs '(A L P H A B E T ) )
ALPHABET
> (gcs '( D A N D E L I O N ) )
DANDELION
> (gcs '( K R I T I K A ))
KRITIKA
> (gcs '(C O N S E Q U E N C E S ) )
CONSEQUENCES
> ( gcs '( Y U N I L ) )
YUNIL
> (gcs '( U N I T E D ) )
UNITED
> ( gcs '(L A N A ) )
LANA
> ( gcs '( D E L R E Y ) )
DELREY
> ( gcs '( T A Y L O R ))
TAYLOR
> (gcs '( N I R V A N A ) )
NIRVANA
>
```