Racket Assignment #5: RLP and HoFs

Learning Abstract

The abstract of this assignment is to be fluent in the use of recursion methods, generate number sequences by performing some interesting sorts of “counting”, be acquainted with “association lists,” transformation of number sequences to musical notes represented in ABC notation, make Frank Stella, simulate a cognitive phenomenon known as chromesthesia and finally stimulate grapheme to color synesthesia, in which letters are mapped to colors.

Task 1: Simple List Generators

Task 1a - iota

Function Definition

```
(define (snoc o lis)
  (cond
    ((empty? lis)
     (list o)
     )
    (else
     (cons (car lis)(snoc o (cdr lis)))
     )
    )
  )

(define (iota n)
  (cond
   ((= n 1) '(1))
   (else
    (snoc n (iota (- n 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 o)
  (cond
    ((zero? n) '())
    (else
     (snoc o (same (- n 1) o) )
    )
  )
  )
```

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 n l)
  (cond
    ((zero? n) '())
    (else
     (cons (car l) (alternator (- n 1) (snoc (car l) (cdr 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 n a)
  (cond
    ((zero? n)'())
    (else
      (snoc (* n a) (sequence (- n 1) a))
     )
    )
  )
```

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 (repeat x y)
  (cond
    ((zero? y)'())
    (else
 (append (cons x (repeat x (- y 1)))
  )
    )
  )
  )

(define (r-count l)
  (cond
    ((null? l)'())
    (else
     (append (repeat(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 3 - Association Lists

Task 3a - Zip

Function Definition

```
(define (zip l s)
  (cond
```

```
  ( (null? l) '())
   (else
    (cons (cons (car l) (car s)) (zip (cdr l) (cdr s))))
    )
   )
```

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 o list)
  (cond
    ((null? list)'())
    ((eq? o (caar list))(car list))
    (else (assoc o (cdr list)))))
```

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-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 4 - Numbers to Notes to ABC

Task 4a - nr- > note

Function Definition

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

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 l scale)
  (cond
    ((null? l)'())
    (else
     (cons (nr->note (car l) scale)(nrs->notes (cdr l) scale)))
    ))
```

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 (a-count ’(1 2 3))) scale-zip-CM)
’("C" "C" "D" "D" "C" "D" "D" "E" "E" "E")

Task 4c - nrs- > abc

Function Definition

```
(define (nrs->abc l scale)
  (cond
    ((null? l)'())
    (else
     (string-join (nrs->notes l scale))
     )
    )
  )
```

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"

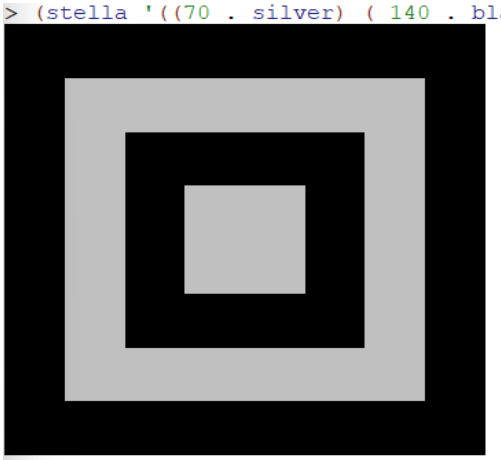
Task 5 - Stella

Function Definition

```
(define (sq n c)
  (square n "solid" c))
(define (stella spec)
  (cond
    ((null? spec) empty-image)
    (else
     (foldr overlay empty-image
         (map sq (map car spec) (map cdr spec))
        )
     )
    ))
```

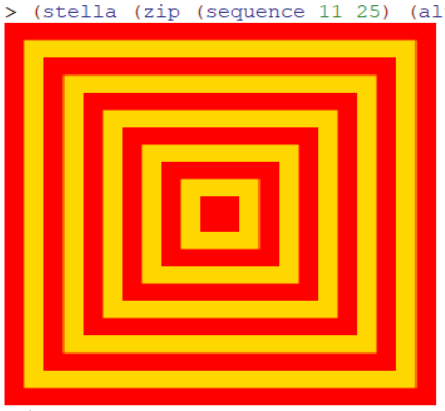
The Five Demos

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



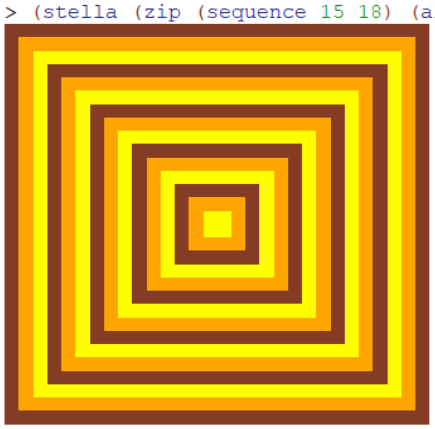
```
>
```

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



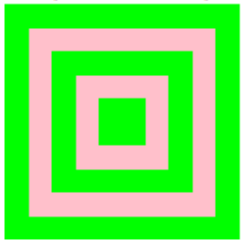
```
>
```

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



```
>
```

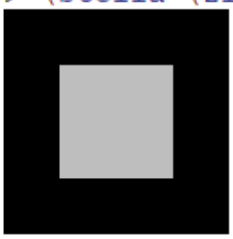
```
> (stella (zip (sequence 5 20)(alternator 5 '(green pink))))
```



2 bound occurrences

```
>
```

```
> (stella (zip (sequence 2 50) (alternator 2 '(gray black))))
```



```
> |
```

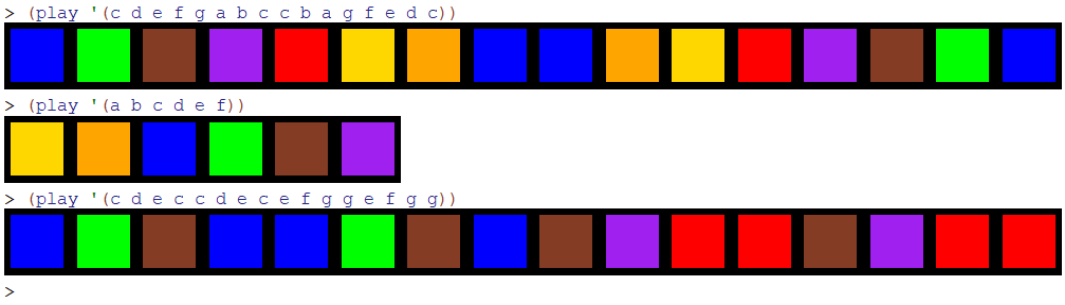
Task 6 - Chromesthetic Renderings

Code

```
(define (sq1 c)
  (overlay
  (square 40 "solid" c)
  (square 50 "solid" "black")
  )
  )
(define pitch-classes '(c d e f g a b))
```

```
(define color-names '(blue green brown purple red yellow orange))
(define pitch-colors
   '((c . (blue))
     (d . (green))
     (e . (brown))
     (f . (purple))
     (g . (red))
     (a . (gold))
     (b . (orange))))
(define (play l)
(cond
((null? l) empty-image)
(else
(foldr beside empty-image
(map sq1 (map car (nrs->notes l pitch-colors)))
))))
```

Demo



Task 7 - Grapheme to Color Synthesia

Function Definition

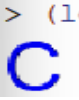
```
( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
( define alphabet '(A B C) )
( define alphapic ( list AI BI CI ) )
(define a->i ( zip alphabet alphapic ))
(define (letter->image a)
  (nr->note a a->i)
  )
```

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

Demo

```
> (letter->image 'A)
```

```
> (letter->image 'B)
```

```
> (letter->image 'C)
```


Demo

```
> (gcs '(A B A B A C A B A C A B C))
```

```
> (gcs '(B A B A B A B A B A B A))
```

```
>
```