Joseph Scollo

Abstract:

In this assignment, various problem sets are solved in racket by use of higher order functions. In addition, the use of various functions to solve these tasks such as: Foldr, Foldl, map and lambda will be showcased.

Task 1: Simple List Generators

Task 1a – Iota

Code

```
1  #lang racket
2
3  (define(snoc k l)
4    (cond
5      ((empty? l)
6        (list k)
7        )
8      (else
9        ( cons(car l)(snoc k( cdr l)))
10       )
11     )
12   )
13
14 (define(iota n)
15   (cond
16     (( = n 1)'( 1 ) )
17     (else
18      (snoc n(iota( - n 1 )))
19     )
20   )
21 )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (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

Code

```
1  #lang racket
2
3
4  (define(snoc k l)
5    (cond
6      ((empty? l)
7        (list k)
8        )
9      (else
10       ( cons(car l)(snoc k( cdr l)))
11       )
12     )
13   )
14
15 (define(same n obj)
16   (cond
17     (( = n 0)'() )
18     (else
19      (snoc obj(same( - n 1 ) obj))
20     )
21   )
22 )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (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

Code

```
#lang racket

(define(snoc k l)
  (cond
    ((empty? l)
      (list k)
      )
    (else
      (cons(car l)(snoc k( cdr l)))
      )
    )
  )

(define(alternator n obj)
  (cond
    (( = n 0)'() )
    (else
     (cons(car obj)(alternator( - n 1 )(snoc (car obj)(cdr obj))))
    )
  )
)
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (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

Code

```
#lang racket
(define(snoc k l)
  (cond
    ((empty? l)
      (list k)
      )
    (else
      ( cons(car l)(snoc k( cdr l)))
      )
    )
  )

(define(iota n)
  (cond
    (( = n 1)'( 1 ) )
    (else
     (snoc n(iota( - n 1 )))
    )
  )
)

(define(sequence n m)
  (map(lambda(x)(* x m))(iota n))
)
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (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

Code

```
29
30  (define (a-count n)
31    (cond
32      ((empty? n)
33       '() )
34      (else
35       (append(iota(car n))( a-count(cdr n))))
36      )
37     )
38   )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (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-Repition Counting

Code

```
40  (define (r-count n)
41    (cond
42      ((empty? n)
43       '() )
44      (else
45       (append(same(car n)(car n))( r-count(cdr n)))
46      )
47     )
48   )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (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

```
Language: racket, with debugging; memory limit: 2048 MB.
> (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 '(5 5 2 3)))
'(1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 1 2 2 1 2 2 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

Code

```
 1 | #lang racket
21 | (define (zip kl vl)
22 |   (cond
23 |     ((or(empty? kl) (empty? vl))
24 |      '()
25 |      )
26 |     (else
27 |      (cons(list*(car kl)(car vl))(zip(cdr kl)(cdr vl)))
28 |      )
29 |     )
30 |   )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (zip '(1 2 3 4 5) '(un deux trois quatre cinq))
'((1 . un) (2 . deux) (3 . trois) (4 . quatre) (5 . 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

Code

```
 1 | #lang racket
21 | (define (zip kl vl)
22 |   (cond
23 |     ((or(empty? kl) (empty? vl))
24 |      '()
25 |      )
26 |     (else
27 |      (cons(list*(car kl)(car vl))(zip(cdr kl)(cdr vl)))
28 |      )
29 |     )
30 |   )
31 |
32 | (define(assoc e l)
33 |   (cond
34 |     ((empty? l)
35 |      '() )
36 |     ((eq? e (car(car l)))
37 |      (car l))
38 |     (else
39 |      (assoc e(cdr l)))
40 |     )
41 |   )
```

Demo

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

> (define al2
    (zip '(one two three) '( (1) (2 2) (3 3 3) ) ) ; # a-list -> zip
  )
> 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

```
44
45  (define scale-zip-CM
46    (zip(iota 7)'("C" "D" "E" "F" "G" "A" "B"))
47    )
48
49  (define scale-zip-short-Am
50    (zip(iota 7)'("A/2" "B/2" "C/2" "D/2" "E/2" "F/2" "G/2"))
51    )
52
53  (define scale-zip-short-low-Am
54    (zip(iota 7)'("A,/2" "B,/2" "C,/2" "D,/2" "E,/2" "F,/2" "G,/2"))
55    )
56
57  (define scale-zip-short-low-blues-Dm
58    (zip(iota 7)'( "D,/2" "F,/2" "G,/2" "_A,/2" "A,/2" "c,/2" "d,/2"))
59    )
60
61  (define scale-zip-wholetone-C
62    (zip(iota 7)'("C" "D" "E" "^F" "^G" "^A" "c"))
63    )
64
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> 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

Code

```
65 (define (nr->note num note)
66   (cond
67     ((or(< num 1) (> num 7 ))
68      '()
69      )
70     (else
71      (cdr(assoc num note))
72     )
73   )
74 )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (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

Code

```
 96
 97  (define (nr->note num note)
 98    (cond
 99      ((or(< num 1) (> num 7 ))
100       '()
101       )
102      (else
103       (cdr(assoc num note))
104      )
105    )
106  )
107
108  (define(nrs->notes numList elements)
109
110    (map(lambda(x)(nr->note x elements))numList)
111  )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (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

Code

```
108  (define(nrs->notes numList elements)
109
110    (map(lambda(x)(nr->note x elements))numList)
111  )
112  (define(nrs->abc numfunc notefunc)
113    (string-join(nrs->notes numfunc notefunc))
114  )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
>  (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"
>
```

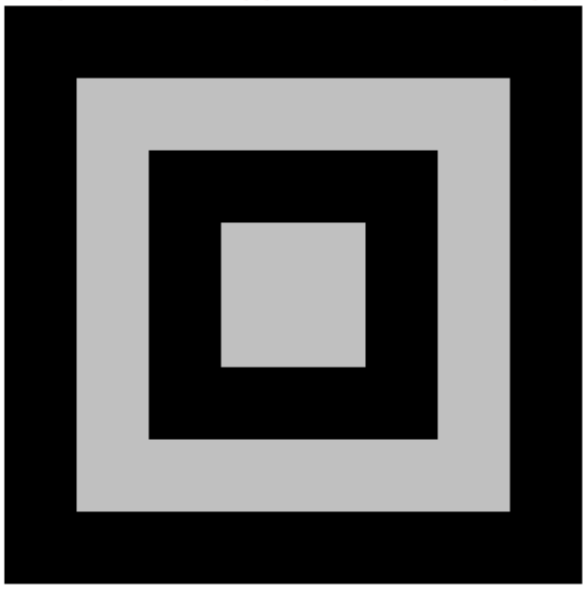
Task 5 – Stella:

Code

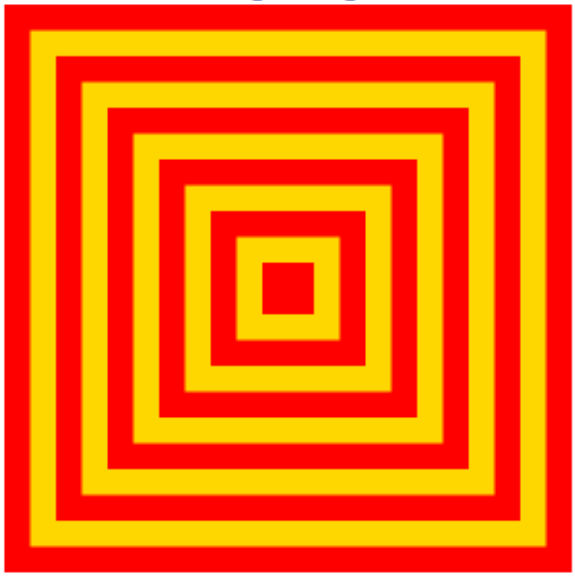
```
72 (define(sequence n m)
73   (map(lambda(x)(* x m))(iota n))
74 )
75 (define(stella spec)
76   (cond
77     ((empty? spec )empty-image)
78     (else
79       (foldr overlay empty-image
80       (map(lambda(x)(square(car x)'solid(cdr x)))spec))
81      )
82   )
83 )
84 (define(alternator n obj)
85   (cond
86     (( = n 0)'() )
87     (else
88      (cons(car obj)(alternator( - n 1 )(snoc (car obj)(cdr obj))))
89     )
90   )
91 )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (stella '((70 . silver)(140 . black)(210 . silver)(280 . black)))
```

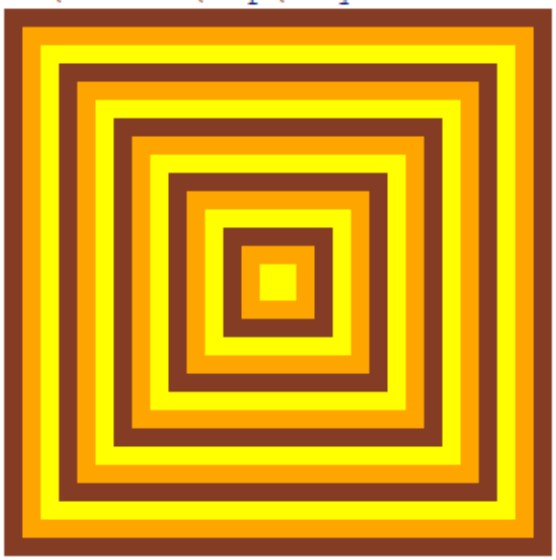


```
> (stella(zip(sequence 11 25)(alternator 11 '(red gold))))
```

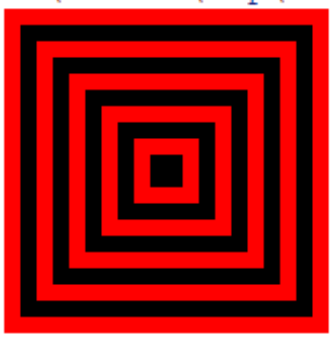


```
> (stella(zip(sequence 15 18)(alternator 15 '(yellow orange brown))))
```

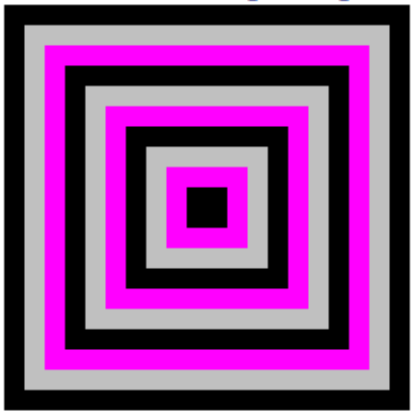
```
> (stella(zip(sequence 15 18)(alternator 15 '(yellow orange brown))))
```



```
> (stella(zip(sequence 10 16)(alternator 10 '(black red))))
```



```
> (stella(zip(sequence 10 20)(alternator 10 '(black magenta silver))))
```



```
>
```

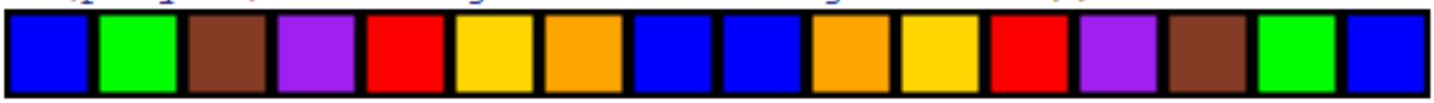
Task 6 – Chromesthetic Renderings:

Code

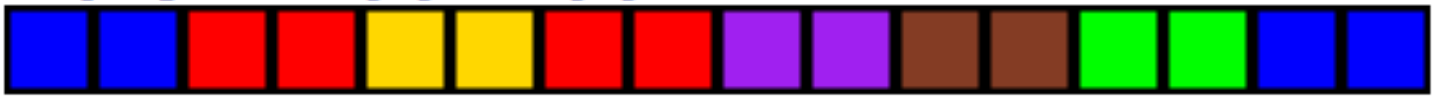
```
92  (define(play n)
93    (cond
94      ((empty? n )empty-image)
95      (
96       else
97        (foldr beside empty-image
98          (map
99            (lambda(y)(color->box y))
100           (map(lambda(x)(pc->color x))n)
101         )
102       )
103     )
104   )
105 )
```

Demo

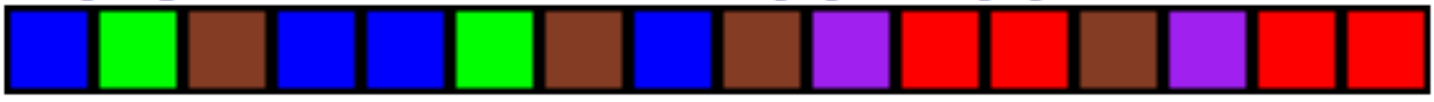
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (play '(c d e f g a b c c b a g f e d c))
```



```
> (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 7 – Grapheme to Color Synethesia:

Code

```
#lang racket

(define AI(text "A" 36 "orange"))
(define BI(text "B" 36 "red"))
(define CI(text "C" 36 "blue"))
(define DI(text "D" 36 "cyan"))
(define EI(text "E" 36 "magenta"))
(define FI(text "F" 36 "maroon"))
(define GI(text "G" 36 "lime"))
(define HI(text "H" 36 "indigo"))
(define II(text "I" 36 "salmon"))
(define JI(text "J" 36 "olive"))
(define KI(text "K" 36 "purple"))
(define LI(text "L" 36 "yellow"))
(define MI(text "M" 36 "gold"))
(define NI(text "N" 36 "silver"))
(define OI(text "O" 36 "violet"))
(define PI(text "P" 36 "pink"))
(define QI(text "Q" 36 "teal"))
(define RI(text "R" 36 "forestgreen"))
(define SI(text "S" 36 "skyblue"))
(define TI(text "T" 36 "fuchsia"))
(define UI(text "U" 36 "orange"))
(define VI(text "V" 36 "brown"))
(define WI(text "W" 36 "plum"))
(define XI(text "X" 36 "gray"))
(define YI(text "Y" 36 "green"))
(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 ( zip alphabet alphapic ) )



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

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

Demo 1

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> alphabet
'(A B C)
> alphapic
```


```
> (display a->i)
```


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


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


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


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


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


```
>
```

Demo 2

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (gcs '(A L P H A B E T))
ALPHABET
> (gcs '(D A N D E L I O N))
DANDELION
> (gcs '(Y O D A))
YODA
> (gcs '(L I G H T S A B O R))
LIGHTSABOR
> (gcs '(U N D E R T A K E R))
UNDERTAKER
> (gcs '(H A S K E L L))
HASKELL
> (gcs '(P R O L O G))
PROLOG
> (gcs '(R A C K E T))
RACKET
> (gcs '(J A V A))
JAVA
> (gcs '(E N C R Y P T I O N))
ENCRYPTION
>
```