Assignment #5

Trevor Strickland, CSC344

October 23, 2022

Learning Abstract: This assignment dives into the more advanced parts of list processing and manipulation in Racket. This involved creating a lot of utility functions to complete certain objectives, such as constructing a long string out of a list, or combining two lists of an equal length together, such that each element of each respective index is paired with each other. Finally, this assignment dives into some more visual representations of these list manipulation techniques.

Task 1: Generating Uniform Lists

Code:

```
(define (generate-uniform-list size token)
  (cond
    ((> size 1)
     (append (list token) (generate-uniform-list (- size 1) token))
     )
    ((= size 1)
     (list token)
     )
    ((= size 0) '())
  )
)
```

Demo:

```
> (generate-uniform-list 5 'cat)
'(cat cat cat cat cat)
> (generate-uniform-list 10 3)
'(3 3 3 3 3 3 3 3 3 3)
> (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 la lb)
  (define len (length la))
  (define (a-cons la lb index limit l)
    (cond
      ((< index len)
       (list* (cons (list-ref la index) (list-ref lb index))
              (a-cons la lb (+ index 1) limit l))
       )
      ((= index len)
       l
      )
     )
   )
 (a-cons la lb 0 len '())
)
```

Demo:

```
> (a-list '(one two three four five) '(uno dos tres cuatro cinco))
'((one . uno) (two . dos) (three . tres) (four . cuatro) (five . cinco))
> (a-list '() '())
'()
> (a-list '(this) '(thing))
'((this . thing))
> (a-list '(one two three) '((1) (2 2) (3 3 3)))
'((one 1) (two 2 2) (three 3 3 3))
```

Task 3: Assoc

Code:

```
(define (assoc x l)
  (define len (length l))
  (define (a-comp vx vl index limit)
    (cond
      ((eq? vx (car (list-ref vl index)))
       (list-ref vl index))
      ((= index (- len 1)) '())
      (else (a-comp vx vl (+ index 1) limit))
      )
    )
  (a-comp x l 0 len)
)
```

Demo:

```
> (define al1 (a-list '(one two three four) '(uno dos tres quatro)))
> (define al2 (a-list '(one two three) '((1) (2 2) (3 3 3))))
> (assoc two al1)
two: undefined;
 cannot reference an identifier before its definition
> (assoc 'two al1)
'(two . dos)
> (assoc 'five al1)
'()
> (assoc 'three al2)
'(three 3 3 3)
> (assoc 'four al2)
'()
```

Task 4: Rassoc

Code:

```
(define (rassoc x l)
  (define len (length l))
  (define (a-comp vx vl index limit)
    (cond
      ((equal? vx (cdr (list-ref vl index)))
       (list-ref vl index))
      ((= index (- len 1)) '())
      (else (a-comp vx vl (+ index 1) limit))
      )
    )
  (a-comp x l 0 len)
)
```

Demo:

```
> (rassoc '(1) al2)
'(one 1)
> (rassoc 'three al1)
'()
> (rassoc 'tres al1)
'(three . tres)
> (rassoc '(1) al2)
'(one 1)
> (rasso '(3 3 3) al2)
rasso: undefined;
 cannot reference an identifier before its definition
> (rassoc '(3 3 3) al2)
'(three 3 3 3)
```

Task 5: Los→s

Code:

```
(define (los→s l)
  (define len (length l))
  (define (s-app vl index limit)
    (cond
      ((< index limit)
       (string-append (string-append (list-ref vl index)
                                     (cond ((< index (- limit 1)) " ")
                                           ((= index (- limit 1)) "")))
                      (s-app vl (+ index 1) limit)))
      ((= index limit) "")
      )
    )
  (s-app l 0 len)
)
```

Demo:

```
> (los->s (generate-uniform-list 20 "|"))
"| | | | | | | | | | | | | | | | | | |"
> (los->s '())
""
> (los->s '("whatever"))
"whatever"
```

Task 6: Generate List

Code:

```
(define (generate-list size func)
  (cond
    ((> size 0)
     (append (list (func)) (generate-list (- size 1) func))
    )
    ((= size 0)
     '()
    )
  )
)
```

Demo:

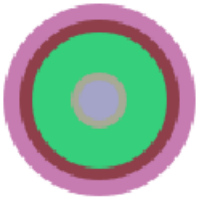
```
> (generate-list 10 roll-die)
'(4 6 1 5 4 4 1 1 1 4)
> (generate-list 20 roll-die)
'(4 1 6 6 1 2 1 3 6 6 2 2 5 3 3 2 3 1 6 1)
> (generate-list 12 (lambda () (list-ref '(red yellow blue) (random 3))))
'(yellow red red yellow blue yellow red yellow blue red blue red)
> (define dots (generate-list 3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

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

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

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

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



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

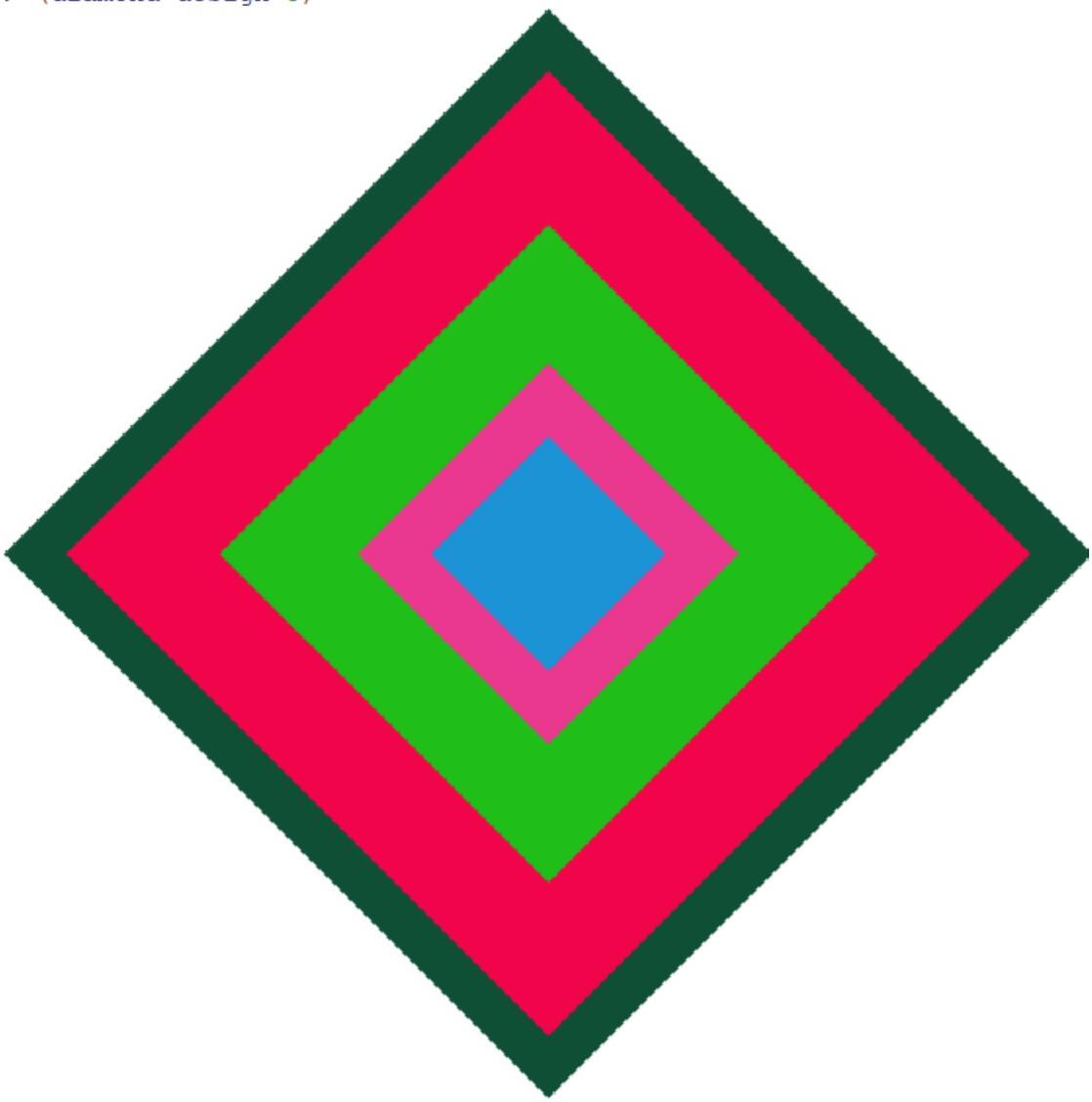
Task 7: The Diamond

Code:

```
(define (diamond-design x)
  (define (d) (rotate 45 (square (+ (random 380) 20) "solid"
                (make-color (random 256) (random 256) (random 256)))))
  (define (sort-d l) (sort l #:key image-width <))
  (define d-list (generate-list x d))
  (foldr overlay empty-image (sort-d d-list))
)
```

Demo:

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



```
> (diamond-design 50)
```

Task 8: Chromesthetic Renderings

Code:

```
(define (play l)
  (define m1 (map (lambda (x) (pc→color x)) l))
  (define m2 (map (lambda (x) (color→box x)) m1))
  (foldr beside empty-image m2)
)
```

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
  '((bread . 2) (cake . 10) (cupcakes . 8) (rolls . 3) (bagels . 4.5)))

(define sales
  '(bread cake cake bread cupcakes rolls rolls rolls rolls bagels bagels
bagels cake bread bread cupcakes bagels rolls cake bread bread bread))

(define (total l x)
  (define pr (assoc x menu))
  (cond
    ((eq? pr '())
      0
    )
    (else
      (define price (cdr pr))
      (define requirement-fulfillment1 (map (lambda (x) (+ x 1)) '(0 1 2)))
      (define requirement-fulfillment2 (foldr cons '(0 1 2) '(3 4 5)))
      (define fl (filter (lambda (i) (eq? x i)) l))
      (* (length fl) price)
  ))
)
```

Demo:

```
> menu
'((bread . 2) (cake . 10) (cupcakes . 8) (rolls . 3) (bagels . 4.5))
> sales
'(bread
  cake
  cake
  bread
  cupcakes
  rolls
  rolls
  rolls
  rolls
  bagels
  bagels
  bagels
  cake
  bread
  bread
  cupcakes
  bagels
  rolls
  cake
  bread
  bread
  bread)
> (total sales 'bread)
14
> (total sales 'cake)
40
> (total sales 'cupcakes)
16
> (total sales 'rolls)
15
> (total sales 'bagels)
18.0
> (total sales 'fish)
0
```

Task 10: Grapheme Color Synthesia

Code:

```
(define AI (text "A" 36 "orange"))
(define BI (text "B" 36 "red"))
(define CI (text "C" 36 "blue"))
(define DI (text "D" 36 "purple"))
(define EI (text "E" 36 "yellow"))
(define FI (text "F" 36 "green"))
(define GI (text "G" 36 "orange"))
(define HI (text "H" 36 "red"))
(define II (text "I" 36 "blue"))
(define JI (text "J" 36 "purple"))
(define KI (text "K" 36 "yellow"))
(define LI (text "L" 36 "green"))
(define MI (text "M" 36 "orange"))
(define NI (text "N" 36 "red"))
(define OI (text "O" 36 "blue"))
(define PI (text "P" 36 "purple"))
(define QI (text "Q" 36 "yellow"))
(define RI (text "R" 36 "green"))
(define SI (text "S" 36 "orange"))
(define TI (text "T" 36 "red"))
(define UI (text "U" 36 "blue"))
(define VI (text "V" 36 "purple"))
(define WI (text "W" 36 "yellow"))
(define XI (text "X" 36 "green"))
(define YI (text "Y" 36 "orange"))
(define ZI (text "Z" 36 "red"))

(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 x)
  (cdr (assoc x a→i))
)

(define (gcs l)
  (define m (map (lambda (x) (letter→image x)) l))
  (foldr beside empty-image m)
)
```

Demo:

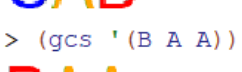
```
> alphabet
'(A B C D E F G H I J K L M N O P Q R S T U V W X Y Z)
> alphapic
```


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

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

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

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



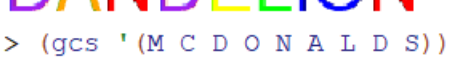
```
> (gcs '(B A B A))
```

```
> (gcs '(A L P H A B E T))
```

```
> (gcs '(D A N D E L I O N))
```

```
> (gcs '(M C D O N A L D S))
```

```
> (gcs '(H A M B U R G E R))
```



```
> (gcs '(C H E E S E S T E A K))
```

```
> (gcs '(U N F O R T U N A T E))
```

```
> (gcs '(S Y N T H E S I A))
```

```
> (gcs '(E C L I P S E D))
```

```
> (gcs '(D I S C O V E R Y))
```

```
> (gcs '(A N T A R C T I C A))
```