Fourth Racket Programming Assignment Solution

This assignment focused on creating recursive functions, using the functions “foldr”, “map”, and “filter”, more list manipulations, and dealing with images. I used the functions “map” and “filter” to create a nearly useless function of my own to generate outlines on boxes.

Task 1 – Generate Uniform List

Source

```
(define (generate-uniform-list num object)
  (cond
    ((> num 0)
     (cons object (generate-uniform-list (- num 1) object))
     )
     ((= num 0)
     '()
     )
    )
  )
```

Demo

```
> (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 2 – Association List Generator

Source

```
(define (a-list list1 list2)
  (cond
    ((> (length list1) 0)
     (cons (cons (car list1) (car list2))
           (a-list (cdr list1) (cdr list2))
           )
     )
    ((= (length list1) 0)
     '()
     )
    )
  )
```

Demo

```
> (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 3 – Assoc

Source

```
(define (assoc object assoc_list)
  (cond
    ((= (length assoc_list) 0)
     '()
     )
    ((equal? object (car (car assoc_list)))
             (car assoc_list))
     (else (assoc object (cdr assoc_list))
        )
     )
  )
```

Demo

```
> (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)
'()
>
```

Task 4 – Rassoc

Source

```
(define (rassoc object assoc_list)
  (cond
    ((= (length assoc_list) 0)
     '()
     )
    ((equal? object (cdr (car assoc_list)))
             (car assoc_list))
     (else (rassoc object (cdr assoc_list))
        )
     )
  )
```

Demo

```
> (define a11 (a-list '(one two three four) '(un deux trois quatre)))
> (define a12 (a-list '(one two three) '( (1) (2 2) (3 3 3))))
> a11
'((one . un) (two . deux) (three . trois) (four . quatre))
> (rassoc 'three a11)
'()
> (rassoc 'trois a11)
'(three . trois)
> a12
'((one 1) (two 2 2) (three 3 3 3))
> (rassoc '(1) a12)
'(one 1)
> (rassoc '(3 3 3) a12)
'(three 3 3 3)
> (rassoc 1 a12)
'()
>
```

Task 5 – Los→s

Source

```
(define (los->s slist)
  (cond
    ((= (length slist) 1)
     (car slist)
     )
    ((> (length slist) 1)
     (string-append (car slist) " " (los->s(cdr slist)))
     )
    ((= (length slist) 0)
     "")
    )
  )
```

Demo

```
> (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

Source

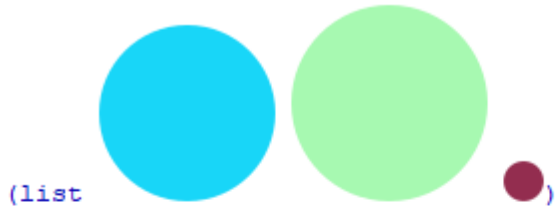
```
(define (roll-die) (+ (random 6) 1))
(define (dot)
  (circle (+ 10 (random 41)) "solid" (random-color))
  )
(define (random-color)
  (color (rgb-value) (rgb-value) (rgb-value))
  )
(define (rgb-value)
  (random 256)
  )
(define (sort-dots loc)
  (sort loc #:key image-width <)
  )

(define (generate-list num func)
  (cond
    ((> num 0)
     (cons (func) (generate-list (- num 1) func))
     )
    ((=  num 0)
     '()
     )
    )
  )
```

Demo 1

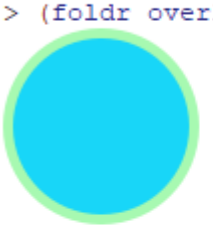
```
> (generate-list 10 roll-die)
'(6 2 2 1 2 1 5 4 5 2)
> (generate-list 20 roll-die)
'(4 6 2 6 2 1 2 6 3 3 4 4 5 1 5 4 3 4 5 1)
> (generate-list 12 (lambda () (list-ref '(red yellow blue) (random 3)))
                  )
'(yellow red blue blue yellow red blue yellow red yellow blue red)
>
```

Demo 2

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

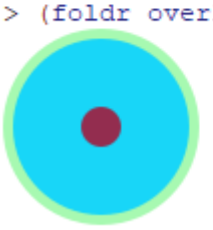


```
(list                                  )
> (foldr overlay empty-image dots)
```



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

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



```
>
```

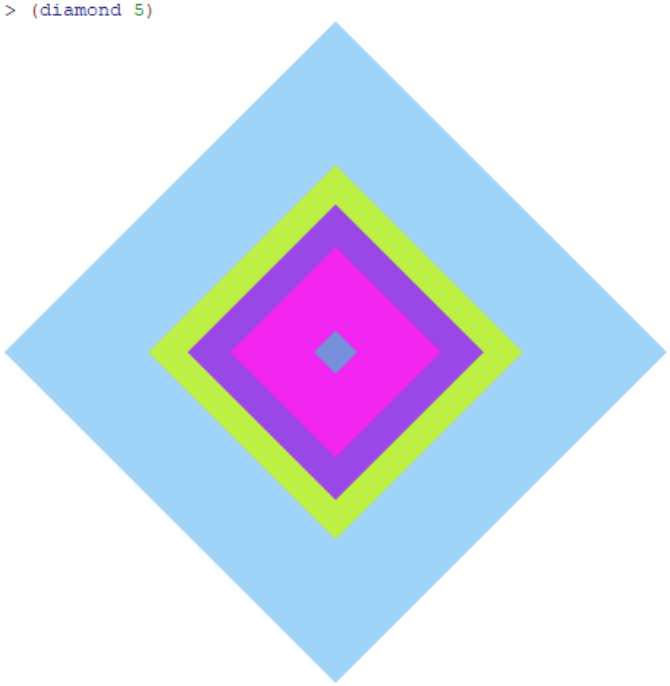
Task 7 – The Diamond

Source

```
(define (gen_diamond)
  (rotate 45 (square (random 20 400) "solid" (random-color)))
  )
(define (sort-diam loc)
  (sort loc #:key image-width <)
  )
(define (diamond num)
  (foldr overlay empty-image (sort-dots (generate-list num gen_diamond)))
  )
```

Demo

```
> (diamond 5)
```



```
>
```

```
> (diamond 20)
```

```
>
```

Task 8 – Chromesthetic renderings

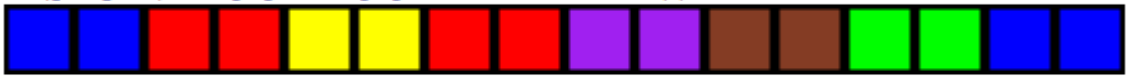
Source

```
(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 (a-list pitch-classes color-names))
(define cb-a-list (a-list 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 plist)
  (foldr beside empty-image (map box (map pc->color plist)))
  )
```

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

Source

```
(define items '(pancakes waffles muffin eggs bacon coffee))
(define prices '(3.0 3.5 1.25 4.0 6.75 2.0))
(define sales '(waffles eggs eggs coffee bacon pancakes coffee coffee coffee
                eggs eggs bacon waffles waffles waffles bacon eggs coffee
                muffin muffin eggs muffin waffles coffee bacon eggs coffee
                pancakes pancakes bacon))

(define menu (a-list items prices))


(define (item_price item)
  (cdr (assoc item menu))
  )

(define (total sales item)
  (foldr + 0 (map item_price (filter (lambda (i)(equal? i item)) sales)))
  )
```

Demo

```
> menu
'((pancakes . 3.0)
  (waffles . 3.5)
  (muffin . 1.25)
  (eggs . 4.0)
  (bacon . 6.75)
  (coffee . 2.0))
> sales
'(waffles
  eggs
  eggs
  coffee
  bacon
  pancakes
  coffee
  coffee
  coffee
  eggs
  eggs
  bacon
  waffles
  waffles
  waffles
  bacon
  eggs
  coffee
  muffin
  muffin
  eggs
  muffin
  waffles
  coffee
  bacon
  eggs
  coffee
  pancakes
  pancakes
  bacon)
> (total sales 'pancakes)
9.0
> (total sales 'waffles)
17.5
> (total sales 'muffin)
3.75
> (total sales 'eggs)
28.0
> (total sales 'bacon)
33.75
> (total sales 'coffee)
14.0
>
```

Task 10 – Wild Card

Specification

Function called *outline_boxes* takes two parameters:

1. First parameter is an integer and is the number of random squares to generate
2. Second parameter is an integer and is the maximum size of square to outline

The function will generate the number of squares equal to the first parameter, and add a black border to any squares below the value of parameter two. The black border fills any space to make the square size 35

Source

```
(define (generate-square-list num)
  (cond
    ((> num 0)
     (cons (box (random-color)) (generate-square-list (- num 1)))
     )
     ((= num 0)
     '()
     )
    )
  )

(define (outline_boxes num size)
  (define (boxes)
    (generate-square-list num))
  (define (overlay_on_box box1)
    (overlay box1 (square 35 "solid" "black")))
  (map overlay_on_box (filter (lambda (b) (> (image-height b) size)) (boxes)))
  )
```

Demo

```
> (outline_boxes 25 15)
(list
> (outline_boxes 35 30)
(list
> (outline_boxes 35 2)
(list
>
```