Fourth Racket Programming Assignment

Abstract

In this programming assignment we focus on constructing and manipulating lists by means of recursion and higher order functions. Recursion is a familiar topic by now but it is still important to show how useful and elegant recursive functions can be. Higher order functions are functions that can take other functions as parameters or return functions as a value. Higher order functions are useful in their ability to shorten code but are also relatively easy to maintain. An important aspect of higher order functions is their flexibility in terms of how they can be used. In this assignment we use higher order functions such as map, filter, and foldr to great effect.

Task 1

Code

```
(define (generate-uniform-list n k)
  (cond
    ((= n 0)
     '())
    ((> n 0)
     (cons k (generate-uniform-list (- n 1) k)))))
```

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

Code

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

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

Code

```
(define (assoc a b)
  (cond
    ((empty? b)
     '())
    ((equal? a (car(car b)))
     (car b))
    (else
     (assoc a (cdr b)))))
```

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

Code

```
(define (rassoc a b)
  (cond
    ((empty? b)
     '())
    ((equal? a (cdr(car b)))
     (car b))
    (else
     (rassoc a (cdr b)))))
```

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))
> (rassoc 'three al1)
'()
> (rassoc 'trois al1)
'(three . trois)
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (rassoc '(1) al2)
'(one 1)
> (rassoc '(3 3 3) al2)
'(three 3 3 3)
> (rassoc 1 al2)
'()
```

Task 5

Code

```
(define (los->s a)
  (cond
    ((empty? a) "")
    ((= (length a) 1)
     (car a))
    (else
     (string-append (car a) " " (los->s (cdr a))))))
```

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

Code

```
(define (generate-list a b)
  (cond
    ((= a 0)
     '())
    ((> a 0)
     (cons (b) (generate-list (- a 1) b)))))
```

Auxiliary Code

```
> (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 <))
```

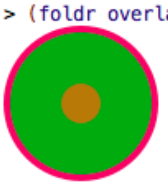
Demo 1

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

Demo 2

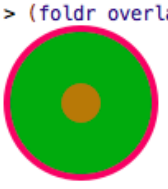
```
> (define dots (generate-list 3 dot))
> dots
```


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



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


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



Demo 3

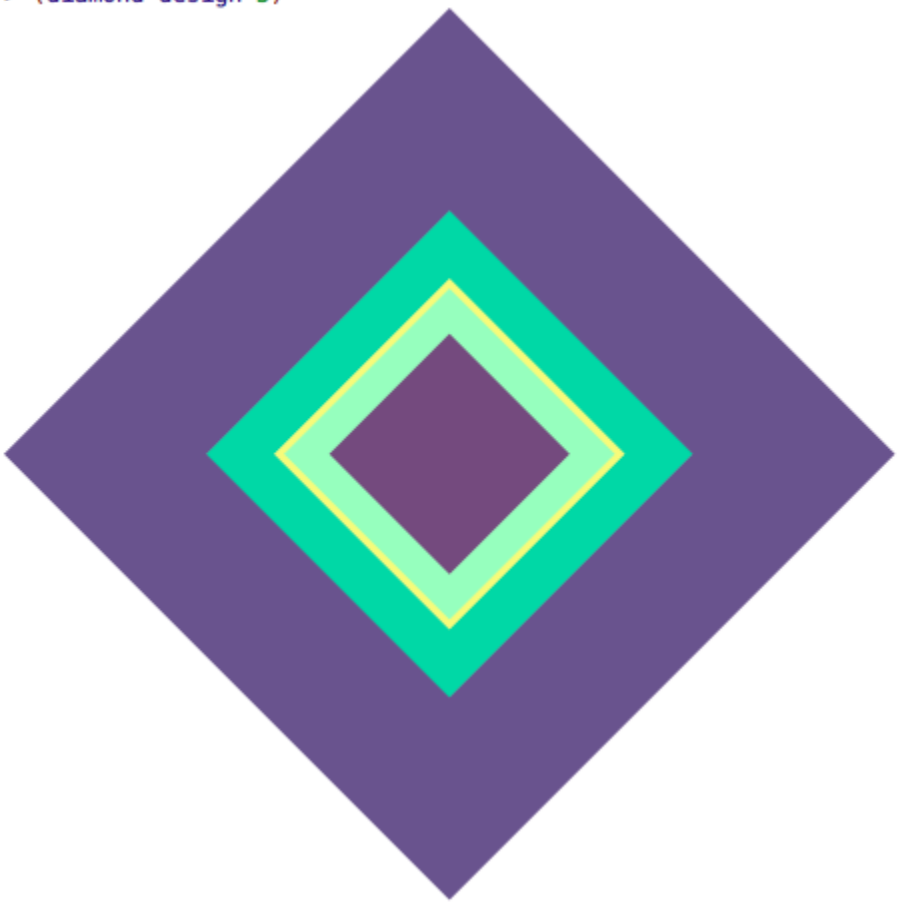
Task 7

Code

```
(define (sort-shapes a) (sort a #:key image-width <))

(define (diamond)
  (rhombus (random-side) 90 "solid" (random-color)))

(define (random-color) (color (random 256) (random 256) (random 256)))

(define (random-side) (random 20 401))

(define (diamond-design a)
  (define diamonds (generate-list a diamond))
  (foldr overlay empty-image(sort-shapes diamonds)))
```

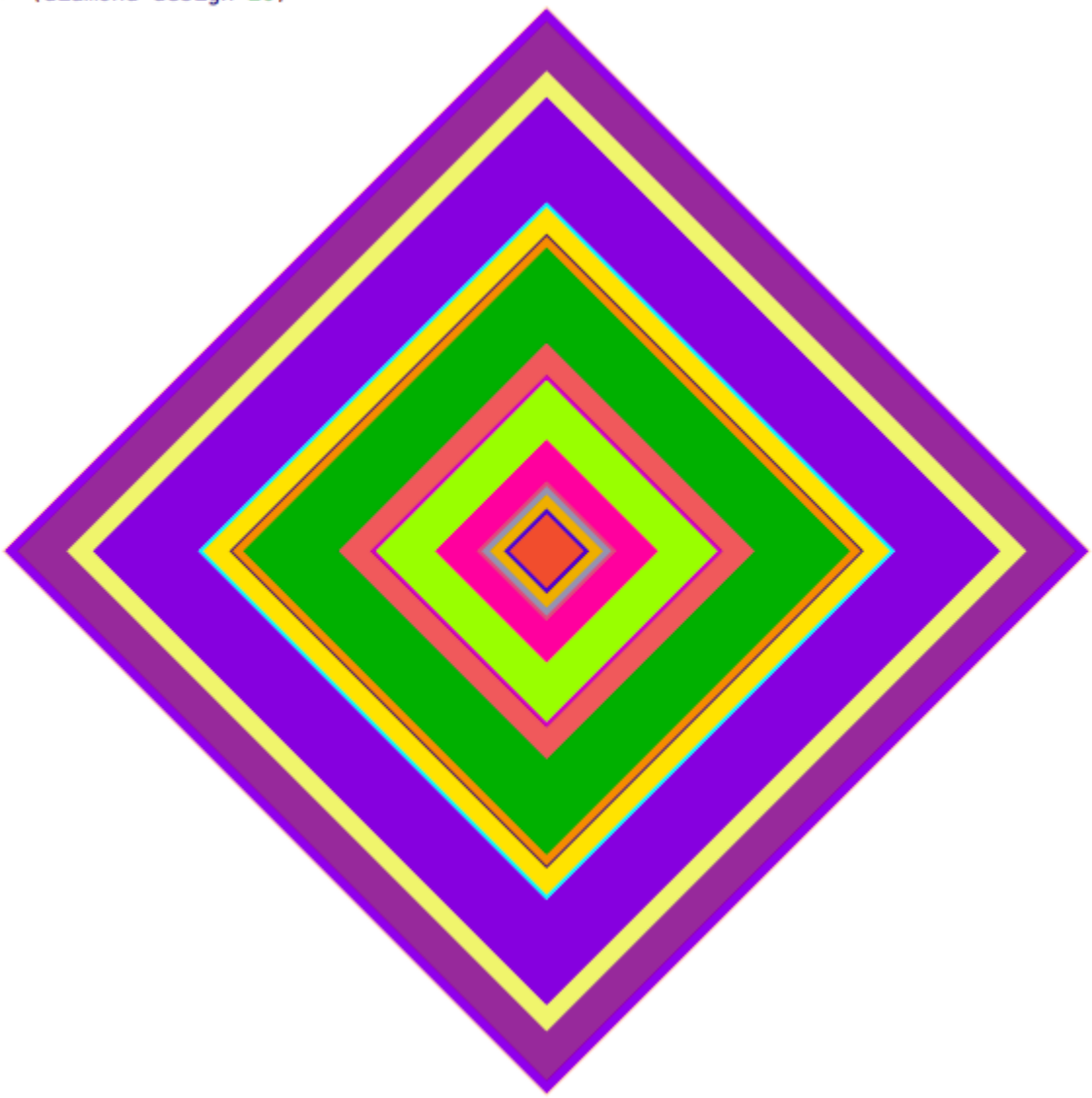
Demo 1

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



Demo 2

```
> (diamond-design 20)
```



Task 8

Code

```
(define (play a)
  (foldr beside empty-image (map color->box (map pc->color a))))
```

Auxiliary 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 ( 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 )))
```

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

Code

```
(define menu-items '(corndog pizza hotdog frieddough rootbeer water))

(define menu-prices '(3 2.5 2 4 1.5 1))

(define menu (a-list menu-items menu-prices))

(define (random-sales n)
  (cond
    ((= n 0) '())
    ((> n 0) (cons (list-ref menu-items (random 6)) (random-sales (- n 1))))
  ))

(define (price a)
  (cond
    ((eq? #f (member a menu-items)) 0)
    (else
     (cdr (assoc a menu)))))

(define sales (random-sales 30))

(define (total a b)
  (foldr + 0 (map price (filter (lambda (x) (eq? b x)) a))))
```

Demo

```
> menu
'((corndog . 3) (pizza . 2.5) (hotdog . 2) (frieddough . 4) (rootbeer . 1.5) (water . 1))
> sales
'(corndog
  hotdog
  pizza
  corndog
  water
  corndog
  rootbeer
  pizza
  frieddough
  rootbeer
  pizza
  water
  hotdog
  water
  frieddough
  hotdog
  pizza
  corndog
  water
  pizza
  pizza
  hotdog
  rootbeer
  frieddough
  frieddough
  hotdog
  frieddough
  frieddough
  rootbeer
  water)
> (total sales 'corndog)
12
> (total sales 'pizza)
15.0
> (total sales 'fish)
0
> (total sales 'frieddough)
24
> (total sales 'rootbeer)
6.0
> (total sales 'water)
5
```