

# Racket Assignment 4:

## RLP and HoF's

### Learning Abstract

This assignment focuses on programs with recursive list processing and list processing with higher order functions. Featuring programs which handle lists on their own, and lists which handle images.

---

### Task 1 - Generate Uniform List

Code:

```
(define (generate-uniform-list size object)
  (cond
    ((> size 1)
     (append (list object) (generate-uniform-list (- size 1) object))
    )
    ((= size 1)
     (list object)
    )
    ((= size 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

Code:

```
(define (a-list list1 list2)
  (cond
    ((empty? list1)
     '())
    (else
     (cons (cons (car list1) (car list2))
           (a-list (cdr list1) (cdr list2))
          )
    )
  )
)
```

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

Code:

```
(define (assoc object assoc-list)
  (cond
    ((null? assoc-list)
     '())
    ((equal? object (car (car assoc-list))) (car assoc-list))
    (else
     (assoc object (cdr assoc-list)))
  )
)
```

Demo:

```
> (define all
  (a-list '(one two three four) '(un deux trois quatre))
)
> (define al2
  (a-list '(one two three) '((1) (2 2) (3 3 3)))
)
> all
'((one . un) (two . deux) (three . trois) (four . quatre))
> (assoc 'two all)
'(two . deux)
> (assoc 'five all)
'()
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (assoc 'three al2)
'(three 3 3 3)
> (assoc 'four al2)
'()
>
```

## Task 4 - Rassoc

Code:

```
(define (rassoc object assoc-list)
  (cond
    ((null? assoc-list)
     '())
    ((equal? object (cdr (car assoc-list))) (car assoc-list))
    (else
     (rassoc object (cdr assoc-list)))
  )
)
```

Demo:

```
> (define all
  (a-list '(one two three four) '(un deux trois quatre))
)
> (define al2
  (a-list '(one two three) '((1) (2 2) (3 3 3)))
)
> all
'((one . un) (two . deux) (three . trois) (four . quatre))
> (rassoc 'three all)
'()
> (rassoc 'trois all)
'(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 - Los->s

Code:

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

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

Code:

```
(require 2htdp/image)





(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 (sorts-dots loc)
  (sort loc #:key image-width <)
)

(define (generate-list number object)
  (cond
    ((< number 1)
     '())
    (else
     (cons
      (object) (generate-list (- number 1) object)
     )
    )
  )
)
```

## Demo 1:

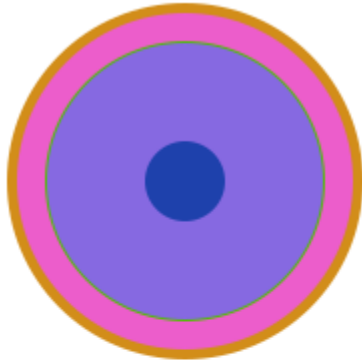
```
> (generate-list 10 roll-die)
'(2 1 5 5 2 5 3 6 6 2)
> (generate-list 20 roll-die)
'(1 2 2 2 5 3 5 3 1 4 2 4 6 5 6 1 3 4 6 1)
> (generate-list 12
  (lambda () (list-ref '(red yellow blue) (random 3))))
'(yellow red blue red yellow yellow red red red red red blue)
>
```

## 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))

  >
```

## Demo 3:

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



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



```
>
```



## Task 7 - The Diamond

Code:

```
(require 2htdp/image)

(define (random-color)
  (color (rgb-value) (rgb-value) (rgb-value))
)

(define (rgb-value)
  (random 256)
)

(define (generate-list number object)
  (cond
    ((< number 1)
     '())
    (else
     (cons
      (object) (generate-list (- number 1) object)
     )
    )
  )
)

(define (diamond-design number)
  (define (diamond)
    (rotate 45
      (square (+ (random 400) 20) "solid" (random-color))
    )
  )
  (define (sort-diamonds n) (sort n #:key image-width <))
  (define diamond-list (generate-list number diamond))
  (foldr overlay empty-image (sort-diamonds diamond-list))
)
```

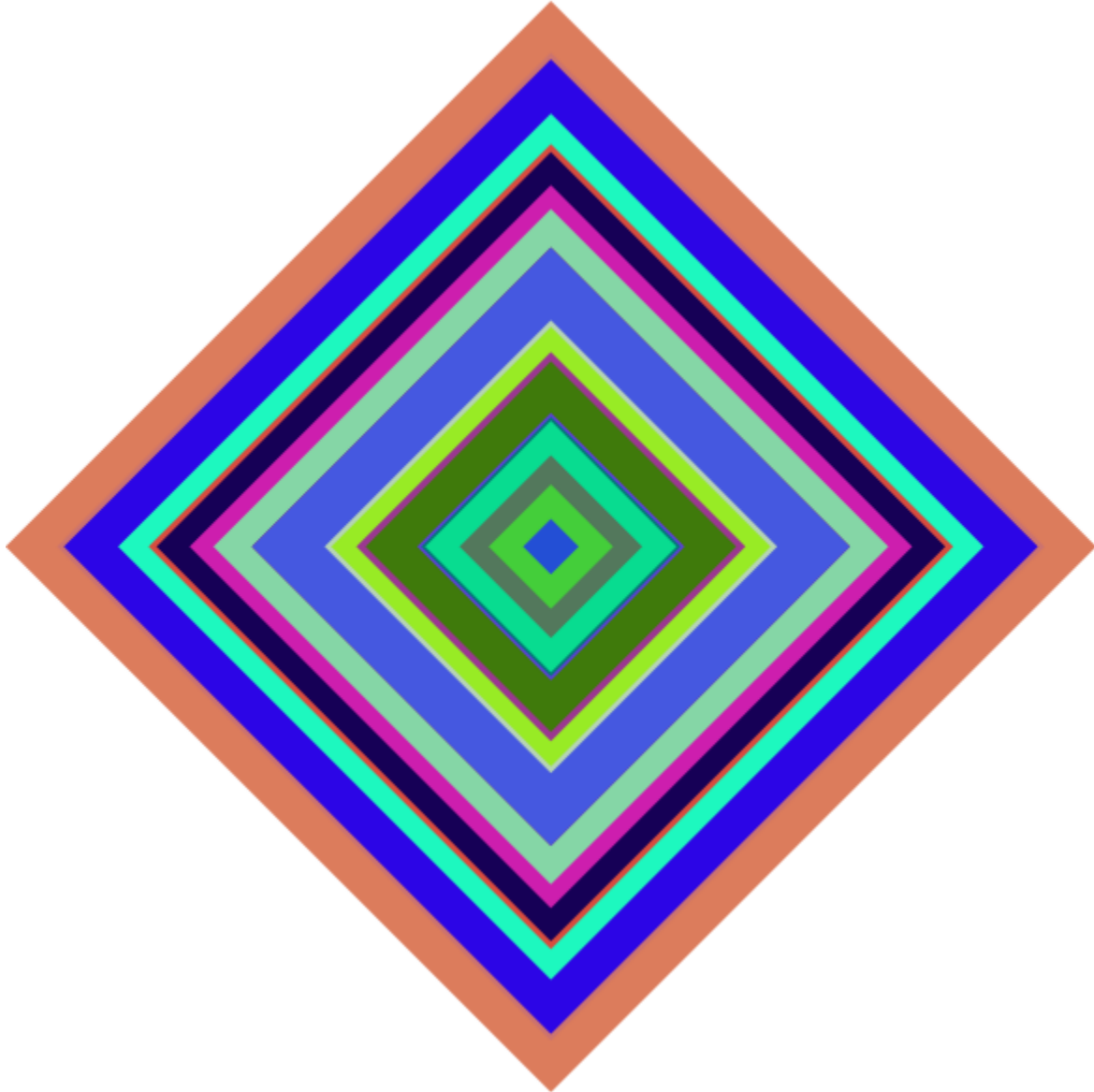
Demo 1:

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



Demo 2:

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



---

Task 8 - Chromesthetic Renderings

Code:

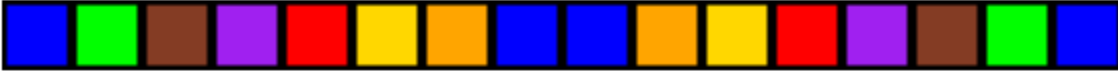
```

1 #lang racket
2 (require 2htdp/image)
3
4 (define pitch-classes '(c d e f g a b))
5 (define color-names '(blue green brown purple red yellow orange))
6
7 (define (box color)
8   (overlay
9     (square 30 "solid" color)
10    (square 35 "solid" "black")
11  )
12 )
13
14 (define boxes
15   (list
16     (box "blue")
17     (box "green")
18     (box "brown")
19     (box "purple")
20     (box "red")
21     (box "gold")
22     (box "orange")
23   )
24 )
25
26 (define (a-list list1 list2)
27   (define n (length list2))
28   (cond
29     ((= n 0)
30      '())
31     )
32   (else
33     (cons (cons (car list1) (car list2)) (a-list (cdr list1) (cdr list2)))
34   )
35 )
36 )
37
38 (define pc-a-list (a-list pitch-classes color-names))
39 (define cb-a-list (a-list color-names boxes))
40
41 (define (pc->color pc)
42   (cdr (assoc pc pc-a-list))
43 )
44
45 (define (color->box color)
46   (cdr (assoc color cb-a-list))
47 )
48
49 (define (play notes)
50   (define colors (map (lambda (x) (pc->color x)) notes))
51   (define box-list (map (lambda (x) (color->box x)) colors))
52   (foldr beside empty-image box-list)
53 )

```

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:

```
1  #lang racket
2
3  (define (a-list list1 list2)
4    (define n (length list2))
5    (cond
6      ((= n 0)
7        '())
8      )
9    (else
10     (cons (cons (car list1) (car list2)) (a-list (cdr list1) (cdr list2)))
11    )
12  )
13 )
14
15 (define (assoc object assoc-list)
16   (cond
17     ((null? assoc-list)
18       '())
19     )
20     ((equal? object (car (car assoc-list))) (car assoc-list))
21     (else
22      (assoc object (cdr assoc-list))
23     )
24   )
25 )
26
27 (define menu
28   '((cheeseburger . 6) (pizza . 4.5) (soda . 1.5) (cake . 3) (sandwich . 4) (pasta . 7))
29 )
30
31 (define sales
32   '(sandwich pizza soda cheeseburger soda cake pasta soda cheeseburger cheeseburger cake pizza pasta soda
33     sandwich sandwich soda cake pizza pizza pizza cheeseburger pasta soda cake cheeseburger pizza cake
34     soda soda sandwich pizza sandwich pasta pasta soda)
35 )
36
37 (define (total n x)
38   (define p (assoc x menu))
39   (cond
40     ((eq? p '())
41       0
42     )
43     (else
44      (define price (cdr p))
45      (define m (map (lambda (x) (+ x 1)) '(0 1 2)))
46      (define f (foldr cons '(0 1 2) '(3 4 5)))
47      (define filt (filter (lambda (i) (eq? x i)) n))
48      (* (length filt) price)
49     )
50   )
51 )
```

## Demo:

```
> menu
'((cheeseburger . 6) (pizza . 4.5) (soda . 1.5) (cake . 3) (sandwich . 4) (pasta . 7))
> sales
'(sandwich
  pizza
  soda
  cheeseburger
  soda
  cake
  pasta
  soda
  cheeseburger
  cheeseburger
  cake
  pizza
  pasta
  soda
  sandwich
  sandwich
  soda
  cake
  pizza
  pizza
  pizza
  cheeseburger
  pasta
  soda
  cake
  cheeseburger
  pizza
  cake
  soda
  soda
  sandwich
  pizza
  sandwich
  pasta
  pasta
  soda)
> (total sales 'cheeseburger)
30
> (total sales 'pizza)
31.5
> (total sales 'soda)
13.5
> (total sales 'cake)
15
> (total sales 'sandwich)
20
> (total sales 'pasta)
35
```

## Task 10 - Grapheme Color Synesthesia

Code:

```
1 | #lang racket
2 | (require 2htdp/image)
3 |
4 | (define (a-list list1 list2)
5 |   (define n (length list2))
6 |   (cond
7 |     ((= n 0)
8 |      '())
9 |     )
10 |   (else
11 |    (cons (cons (car list1) (car list2)) (a-list (cdr list1) (cdr list2)))
12 |    )
13 |   )
14 | )
15 |
16 | (define AI (text "A" 36 "orange"))
17 | (define BI (text "B" 36 "red"))
18 | (define CI (text "C" 36 "blue"))
19 | (define DI (text "D" 36 "green"))
20 | (define EI (text "E" 36 "purple"))
21 | (define FI (text "F" 36 "indigo"))
22 | (define GI (text "G" 36 "teal"))
23 | (define HI (text "H" 36 "cadet blue"))
24 | (define II (text "I" 36 "aqua"))
25 | (define JI (text "J" 36 "cornflower blue"))
26 | (define KI (text "K" 36 "royal blue"))
27 | (define LI (text "L" 36 "light sea green"))
28 | (define MI (text "M" 36 "dark green"))
29 | (define NI (text "N" 36 "gold"))
30 | (define OI (text "O" 36 "olive"))
31 | (define PI (text "P" 36 "violet red"))
32 | (define QI (text "Q" 36 "indian red"))
33 | (define RI (text "R" 36 "deep pink"))
34 | (define SI (text "S" 36 "aquamarine"))
35 | (define TI (text "T" 36 "cyan"))
36 | (define UI (text "U" 36 "crimson"))
37 | (define VI (text "V" 36 "orchid"))
38 | (define WI (text "W" 36 "firebrick"))
39 | (define XI (text "X" 36 "rosy brown"))
40 | (define YI (text "Y" 36 "maroon"))
41 | (define ZI (text "Z" 36 "salmon"))
42 |
43 | (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))
44 | (define alphabetic (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))
45 |
```



```

45 |
46 | (define a->i (a-list alphabet alphabetic))
47 |
48 | (define (letter->image letter)
49 |   (cdr (assoc letter a->i))
50 | )
51 |
52 | (define (gcs letters)
53 |   (cond
54 |     ((empty? letters)
55 |      (empty-image)
56 |    )
57 |     (else
58 |      (define list (map letter->image letters))
59 |      (foldr beside empty-image list)
60 |    )
61 |   )
62 | )

```

---

## Task 10b - ABC Demo:

```

> alphabet
'(A B C)
> alphabetic

(list A B C)
> (display a->i)

((A . A) (B . B) (C . C))
> (letter->image 'A)
A
> (letter->image 'B)
B
> (gcs '(C A B))
CAB
> (gcs '(B A A))
BAA
> (gcs '(B A B A))
BABA

```

## Task 10d - Alphabet 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)'
> alphabetic

(list 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)
> (gcs '(A L P H A B E T))
ALPHABET
> (gcs '(D A N D E L I O N))
DANDELION
> (gcs '(R O C K E T))
ROCKET
> (gcs '(O K A Y))
OKAY
> (gcs '(B R A N C H))
BRANCH
> (gcs '(S L O W L Y))
SLOWLY
> (gcs '(T H R O U G H))
THROUGH
> (gcs '(F I G H T))
FIGHT
> (gcs '(G R A N D))
GRAND
> (gcs '(C L O S E))
CLOSE
```