

# Racket Programming Assignment #4: RLP and HOF's

**Learning Abstract:** Throughout the first five tasks we use straightforward recursive methods for list processing but the following five require us to use high order functions. The methods use were relatively straightforward and some borrowed code from a previous task to complete the newer task.

# Task 1: Generate Uniform List

```
1 | #lang racket
2 |
3 |
4 | (define(generate-uniform-list num obj)
5 |   (cond
6 |     ((= num 0)
7 |      '())
8 |     )
9 |     ((> num 0)
10 |      (cons obj (generate-uniform-list (- num 1) obj))) ) )
```

```
Welcome to DrRacket, version 8.5 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (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))
> (generate-uniform-list 5 'brr)
```

Task 1 code and demo

## Task 2: Association List Generator

```
1: gen-uni-list.rkt

1 | #lang racket
2 |
3 | (define (a-list obj objclone)
4 |   (cond
5 |     ((empty? obj)
6 |      '())
7 |     )
8 |   (else
9 |     (cons (cons (car obj) (car objclone))
10 |           (a-list (cdr obj) (cdr objclone))
11 |           ) ) )
12 |

Welcome to DrRacket, version 8.5 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( 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 2 code and demo

## Task 3: Assoc

```
1 #lang racket
2
3 ;Used code from task 2
4 (define (a-list obj objclone)
5   (cond
6     ((empty? obj)
7      '())
8     )
9   (else
10    (cons (cons (car obj) (car objclone))
11          (a-list (cdr obj) (cdr objclone))
12          ) ) ) )
13
14 (define (assoc obj assoclist)
15   (cond
16     ((empty? assoclist)
17      '())
18     ((equal? (caar assoclist) obj)
19      (car assoclist) )
20     (else
21      (assoc obj (cdr assoclist) ) ) ) ) )
```

Welcome to [DrRacket](#), version 8.5 [cs].  
Language: [racket](#), with [debugging](#); memory limit: 128 MB.

```
> ( define al1 ( a-list '(one two three four) '(un duex trois quatre) ) )
> ( define al2 ( a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
> al1
'((one . un) (two . duex) (three . trois) (four . quatre))
> (assoc 'five al1)
'()
> (assoc 'two al1)
'(two . duex)
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (assoc 'three al2)
'(three 3 3 3)
> (assoc 'four al2)
'()
>
```

Task 3 code and demo

## Task 4: Rassoc

```
1 | #lang racket
2 |
3 | ;Used code from task 2
4 | (define (a-list obj objclone)
5 |   (cond
6 |     ((empty? obj)
7 |      '())
8 |     )
9 |     (else
10 |      (cons (cons (car obj) (car objclone))
11 |            (a-list (cdr obj) (cdr objclone))
12 |            ) ) ) )
13 |
14 | (define (rassoc obj rassoclist)
15 |   (cond
16 |     ((empty? rassoclist)
17 |      '())
18 |     ((equal? (cdr (car rassoclist)) obj)
19 |      (car rassoclist) )
20 |     (else
21 |      (rassoc obj (cdr rassoclist) ) ) ) ) )
```



Welcome to [DrRacket](#), version 8.5 [cs].

Language: [racket](#), with [debugging](#); memory limit: 128 MB.

```
> (define al1 (a-list '(one two three four) '(un duex trois quatre) ) )
> (define al2 (a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
> al1
'((one . un) (two . duex) (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: Los->s

```
1 #lang racket
2
3 ;Code used from task 1
4 (define(generate-uniform-list num obj)
5   (cond
6     ((= num 0)
7      (list)
8     )
9     ((> num 0)
10      (cons obj (generate-uniform-list (- num 1) obj)) ) ) )
11
12
13 (define (los->s stringlist)
14   (cond
15     ((empty? stringlist) "")
16     ((= (length stringlist) 1)
17      (car stringlist))
18     (else
19      (string-append (car stringlist) " " (los->s (cdr stringlist))) ) ) )
20
```

```
Welcome to DrRacket, version 8.5 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( los->s '( "red" "yellow" "blue" "purple" ) )
"red yellow blue purple"
> (los->s (generate-uniform-list 20 "-"))
"-----"
> (los->s '() )
""
> (los-> '("whatever") )
  los->: undefined;
cannot reference an identifier before its definition
> (los->s'("whatever") )
"whatever"
> |
```

## Task 6: Generate List

```
1 #lang racket
2 (require 2htdp/image)
3
4 (define (roll-die) (random 1 7))
5
6 (define (dot) (circle (+ 10 (random 41)) "solid" (random-color)))
7
8 (define (big-dot) (circle (+ 10 (random 150)) "solid" (random-color)))
9
10 (define (random-color) (color (rgb-value) (rgb-value) (rgb-value)))
11
12 (define (rgb-value) (random 256))
13
14 (define (sort-dots loc) (sort loc #:key image-width <))
15
16 (define (generate-list num obj)
17   (cond
18     ((= num 0) '())
19     (else
20      (cons (obj) (generate-list (- num 1) obj))))))
```

Welcome to [DrRacket](#), version 8.5 [cs].

Language: **racket**, with **debugging**; memory limit: 128 MB.

> (generate-list 10 roll-die)

'(4 4 5 1 3 1 4 6 6 2)

> (generate-list 20 roll-die)

'(2 2 1 5 1 1 1 6 3 4 5 5 4 5 3 4 6 5 4 3)

> (generate-list 12 (lambda () (list-ref '(red yellow blue) (random 3))))

'(red red blue blue yellow blue blue yellow blue red yellow blue)

Welcome to [DrRacket](#), version 8.5 [cs].

Language: **racket**, with **debugging**; memory limit: 128 MB.

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

> dots

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

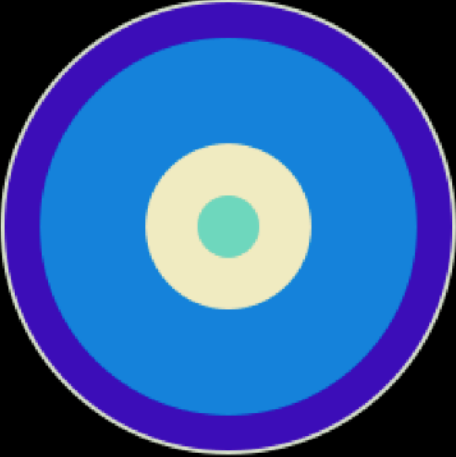
  
> (sort-dots dots)

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

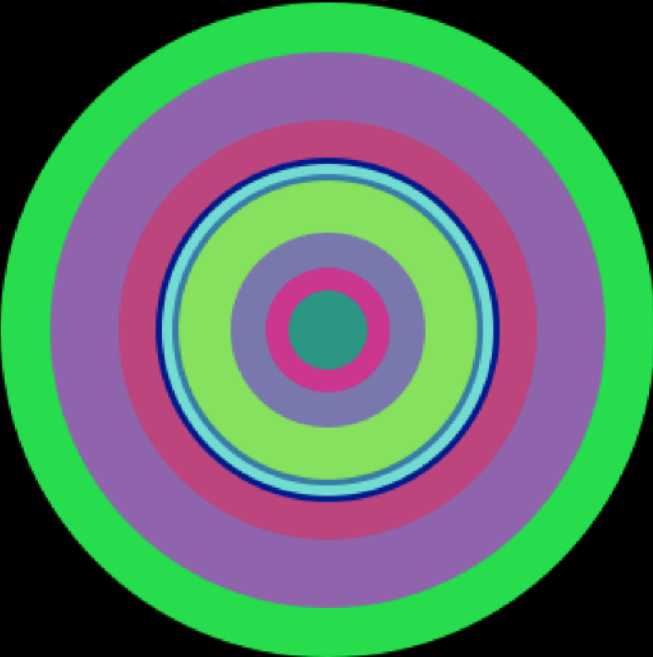


> |

Welcome to [DrRacket](#), version 8.5 [cs].  
Language: racket, with debugging; memory limit: 128 MB.  
> (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) )



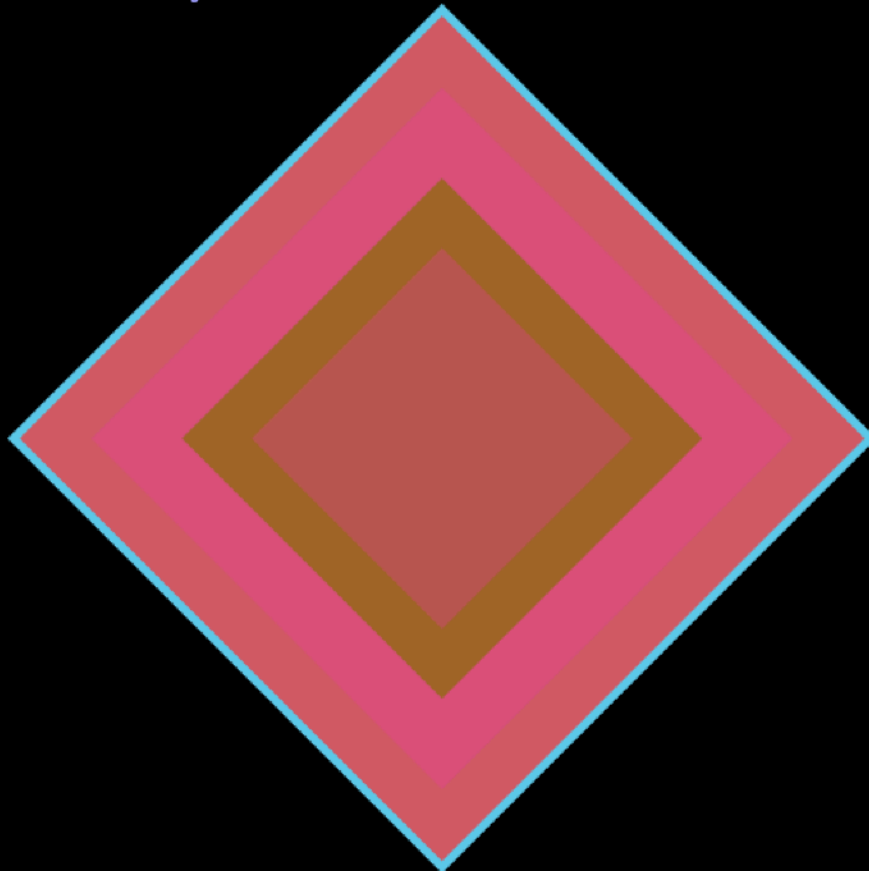
Task 6 Demos 1-3



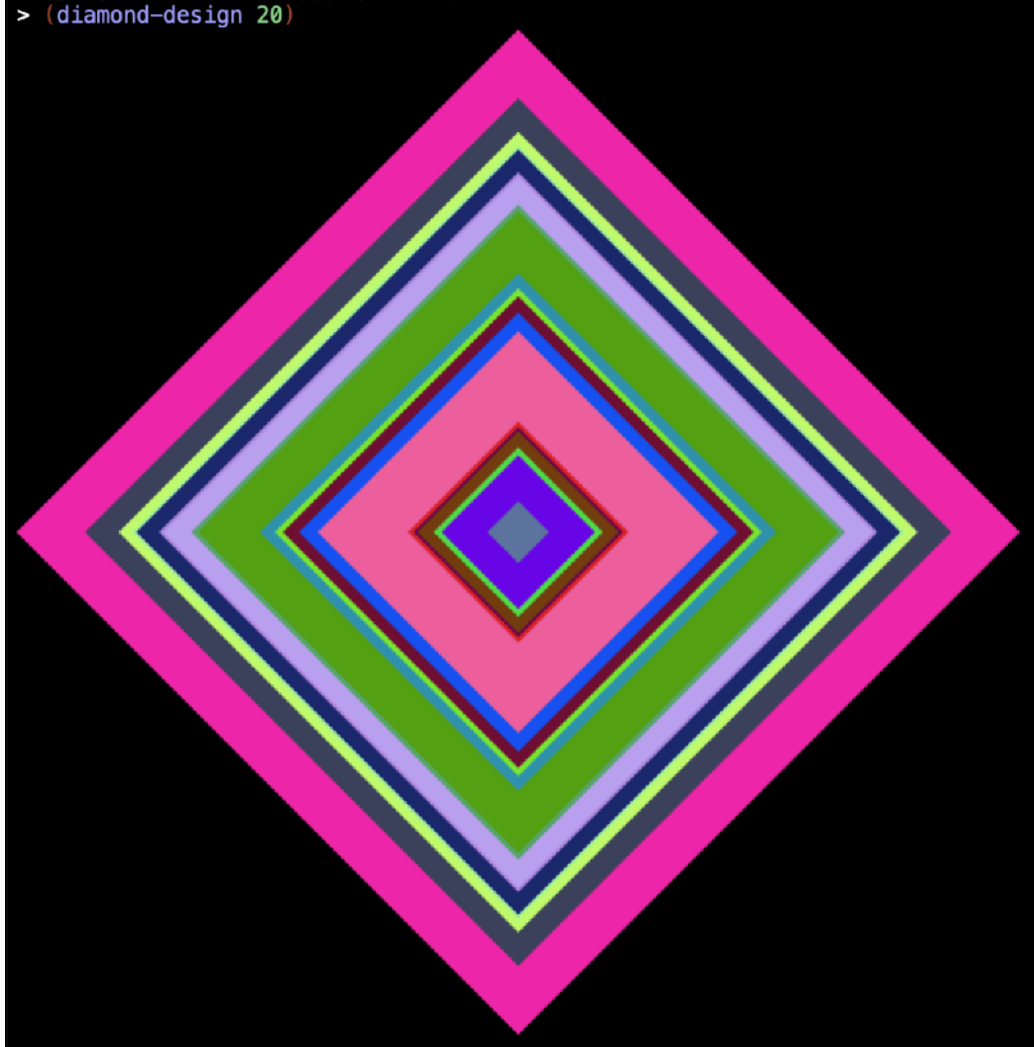
## Task 7: The Diamond

```
1 #lang racket
2 (require 2http/image)
3
4 (define (diamond-design num)
5   (define dots (recursive-diamond num) )
6   (foldr overlay empty-image (sort dots #:key image-width < ) ) )
7
8 (define (recursive-diamond num)
9   (define col (color (random 256) (random 256) (random 256) ) )
10  (define rotsqua (rotate 45 (square (random 20 400) "solid" col) ) )
11  (cond
12    ((= num 0) '() )
13    (else
14     (cons rotsqua (recursive-diamond (- num 1) ) ) ) ) ) )
```

Welcome to [DrRacket](#), version 8.5 [cs].  
Language: [racket](#), with [debugging](#); memory limit: 128 MB.  
> (diamond-design 5)



> |



Task 7 code and demos

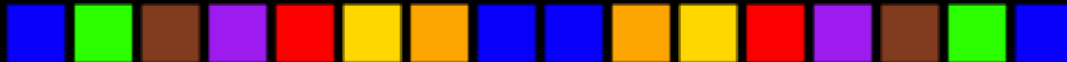
## Task 8: Chromesthetic Renderings

```
15 ( define pitch-classes '( c d e f g a b ) )
16
17 ( define color-names '( blue green brown purple red yellow orange ) )
18
19 ( define ( box color )
20   ( overlay
21     ( square 30 "solid" color )
22     ( square 35 "solid" "black" ) ) )
23
24 ( define boxes
25   ( list
26     ( box "blue" )
27     ( box "green" )
28     ( box "brown" )
29     ( box "purple" )
30     ( box "red" )
31     ( box "gold" )
32     ( box "orange" ) ) )
33
34 ( define pc-a-list ( a-list pitch-classes color-names ) )
35
36 ( define cb-a-list ( a-list color-names boxes ) )
37
38 ( define ( pc->color pc ) ( cdr ( assoc pc pc-a-list ) ) )
39
40 ( define ( color->box color ) ( cdr ( assoc color cb-a-list ) ) )
41
42 (define (play pitch-list)
43   (define color-list (map pc->color pitch-list) )
44   (define box-list (map color->box color-list) )
45   (foldr beside empty-image box-list) )
46
```

Welcome to [DrRacket](#), version 8.5 [cs].

Language: [racket](#), with [debugging](#); memory limit: 128 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 ) )



> 6

## Task 9: Diner

```
> menu
'((fettucini . 26.9)
  (icetea . 6)
  (blt . 12)
  (cheesecake . 20.2)
  (orangejuice . 3)
  (milkshake . 9.7))
> sales
'(fettucini
  icetea
  blt
  milkshake
  blt
  icetea
  cheesecake
  orangejuice
  fettucini
  fettucini
  icetea
  fettucini
  milkshake
  fettucini
  milkshake
  cheesecake
  orangejuice
  cheesecake
  orangejuice
  blt
  milkshake
  fettucini
  fettucini
  icetea
  cheesecake
  orangejuice
  cheesecake
  orangejuice
  fettucini
  milkshake
  fettucini
  milkshake)
> (total sales 'fettucini )
242.10000000000002
> (total sales 'hotdog)
0
> (total sales 'icetea)
24
> (total sales 'blt)
36
> (total sales 'cheesecake)
101.0
> (total sales 'orangejuice)
15
> (total sales 'milkshake)
58.2
>
```

```

;Used code from task 2
(define (a-list obj objclone)
  (cond
    ((empty? obj)
     '())
    )
  (else
   (cons (append (list (car obj)) (car objclone))
         (a-list (cdr obj) (cdr objclone))
         ) ) ) )

;Used code from task 3
(define (assoc obj assoclist)
  (cond
    ((empty? assoclist)
     '())
    ((equal? (caar assoclist) obj)
     (car assoclist) )
    (else
     (assoc obj (cdr assoclist) ) ) ) ) )

(define foods '( fettucini icetea blt cheesecake orangejuice milkshake) )
(define prices '( 26.9 6 12 20.2 3 9.7 ) )

(define menu (a-list foods prices) )

(define sales '(fettucini icetea blt milkshake blt icetea cheesecake orangejuice
               fettucini fettucini icetea fettucini milkshake fettucini
               milkshake cheesecake orangejuice
               cheesecake orangejuice blt
               milkshake fettucini fettucini icetea
               cheesecake orangejuice cheesecake orangejuice
               fettucini milkshake fettucini milkshake) )

(define (food->price i)
  (cdr (assoc i menu) ) )

(define (total sales-list food)
  (define food-sales (filter (lambda (search) (equal? search food ) ) sales-list) )
  (define price-sales (map food->price food-sales) )
  (foldr + 0 price-sales ) )

```

Task 9 code

# Task 10: Grapheme Color Synesthesia

```
Welcome to DrRacket, version 8.5 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> alphabet
'(A B C)
> alphapic
(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 10 demo 1

```
> (gcs '(A L P H A B E T) )  
ALPHABET  
> (gcs '(D A N D E L I O N) )  
DANDELION  
> (gcs '(T I R E D) )  
TIRED  
> (gcs '(L A W S N I G H T M A R E) )  
LAWSNIGHTMARE  
> (gcs '(M A D A R A) )  
MADARA  
> (gcs '(N O T T I B O P) )  
NOTTIBOP  
> (gcs '(E X I S T E N T I A L) )  
EXISTENTIAL  
> (gcs '(C R I S I S) )  
CRISIS  
> (gcs '(K A N Y E) )  
KANYE  
> (gcs '(C U N N Y) )  
CUNNY  
>
```

Task 10 demo 2

```

( define AI (text "A" 36 "ORANGE") )
( define BI (text "B" 36 "RED") )
( define CI (text "C" 36 "BLUE") )
( define DI (text "D" 36 "YELLOW") )
( define EI (text "E" 36 "VIOLETRED") )
( define FI (text "F" 36 "PINK") )
( define GI (text "G" 36 "SPRINGGREEN") )
( define HI (text "H" 36 "LIGHTSEAGREEN") )
( define II (text "I" 36 "PALETURQUOISE") )
( define JI (text "J" 36 "DARKORCHID") )
( define KI (text "K" 36 "DARKGRAY") )
( define LI (text "L" 36 "SLATEBLUE") )
( define MI (text "M" 36 "THISTLE") )
( define NI (text "N" 36 "DARKBLUE") )
( define OI (text "O" 36 "AZURE") )
( define PI (text "P" 36 "CORAL") )
( define QI (text "Q" 36 "HOTPINK") )
( define RI (text "R" 36 "FIREBRICK") )
( define SI (text "S" 36 "SADDLEBROWN") )
( define TI (text "T" 36 "LIGHTPINK") )
( define UI (text "U" 36 "LIME") )
( define VI (text "V" 36 "LIGHTSKYBLUE") )
( define WI (text "W" 36 "DARKMAGENTA") )
( define XI (text "X" 36 "MIDNIGHTBLUE") )
( define YI (text "Y" 36 "GOLD") )
( define ZI (text "Z" 36 "VIOLET") )

( 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 letter) (cdr (assoc letter a->i) ) )

(define (gcs letters)
  (cond
    ((empty? letters) (empty-image) )
    (else
     (define piclist (map letter->image letters) ) (foldr beside empty-image piclist) ) ) )

```

Task 10 code