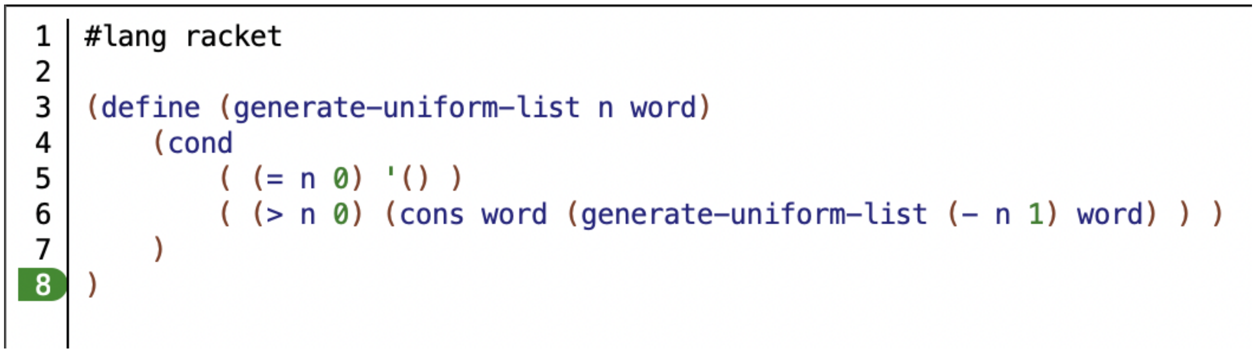
Racket Programming Assignment# 4: RLP and HoFs

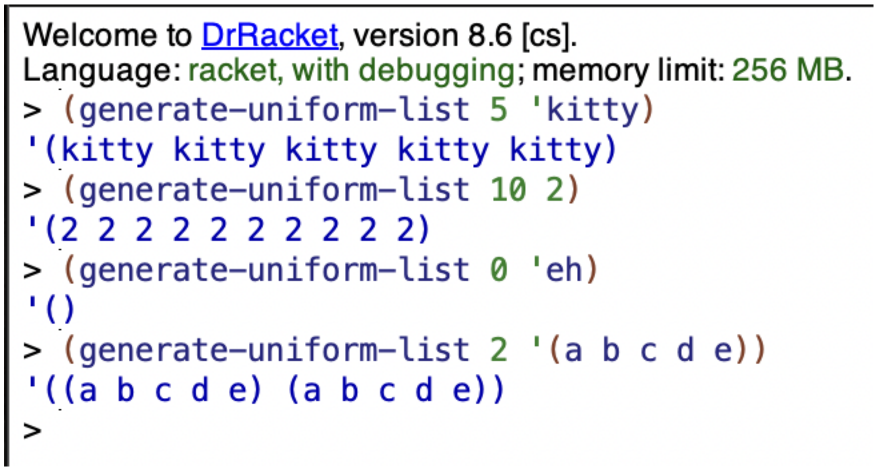
Learning Abstract

This assignment features programs working within the DrRacket PDE. The first five tasks pertain to straightforward recursive list processing. The remaining tasks pertain to list processing with higher order functions.

Task 1: Generate Uniform List

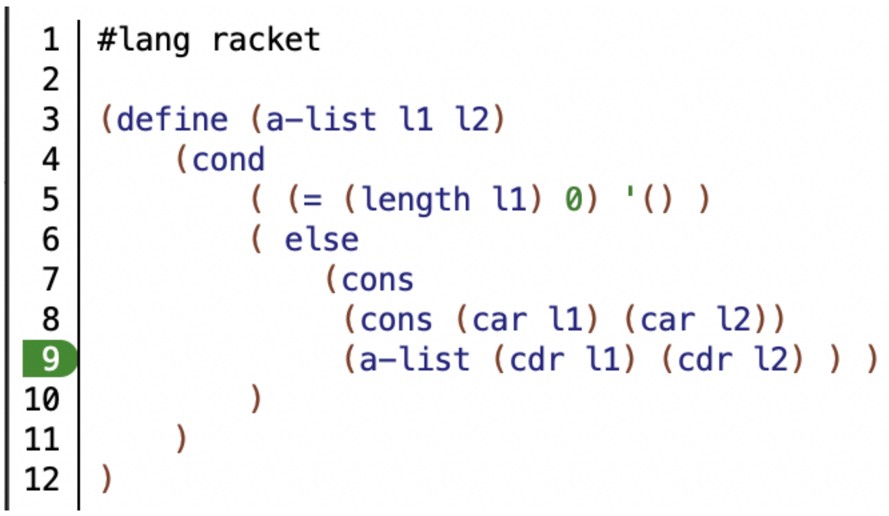


```
#lang racket

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

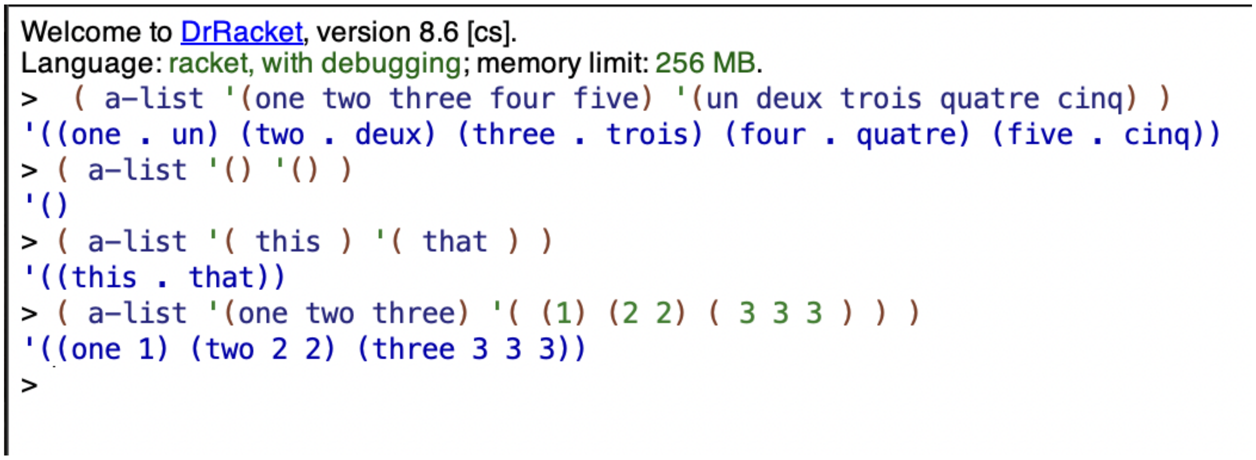


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (generate-uniform-list 5 'kitty)
'(kitty kitty kitty kitty kitty)
> (generate-uniform-list 10 2)
'(2 2 2 2 2 2 2 2 2 2)
> (generate-uniform-list 0 'eh)
'()
> (generate-uniform-list 2 '(a b c d e))
'((a b c d e) (a b c d e))
>
```

Task 2: Association List Generator



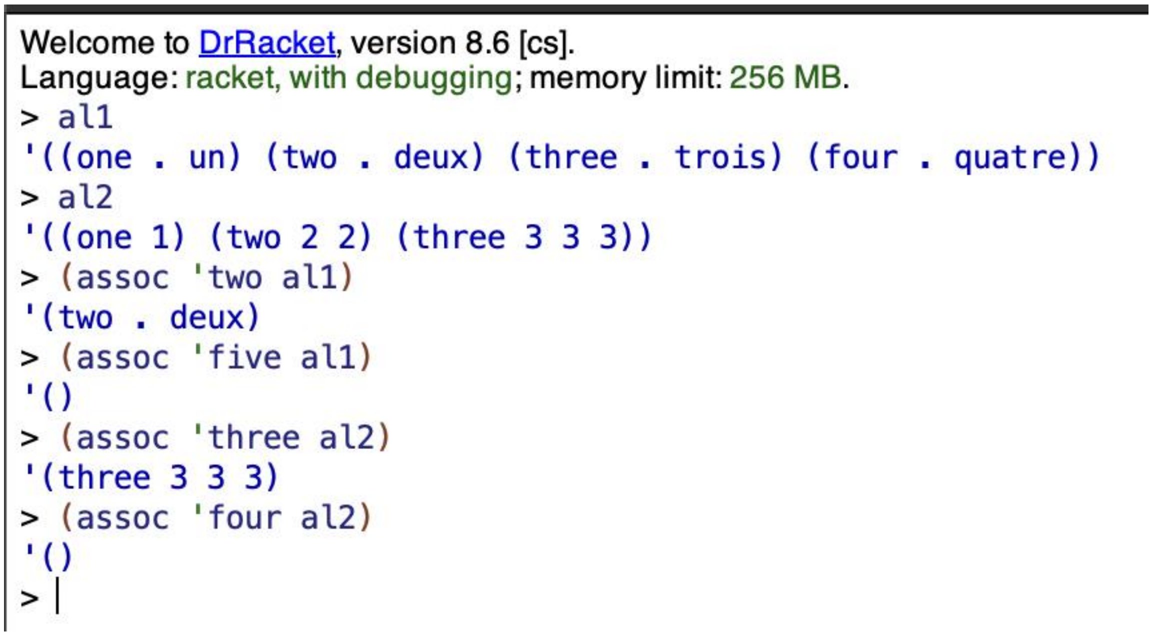
```
#lang racket

(define (a-list l1 l2)
    (cond
        ( (= (length l1) 0) '() )
        ( else
            (cons
             (cons (car l1) (car l2))
             (a-list (cdr l1) (cdr l2) ) )
        )
    )
)
```



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 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))
>
```

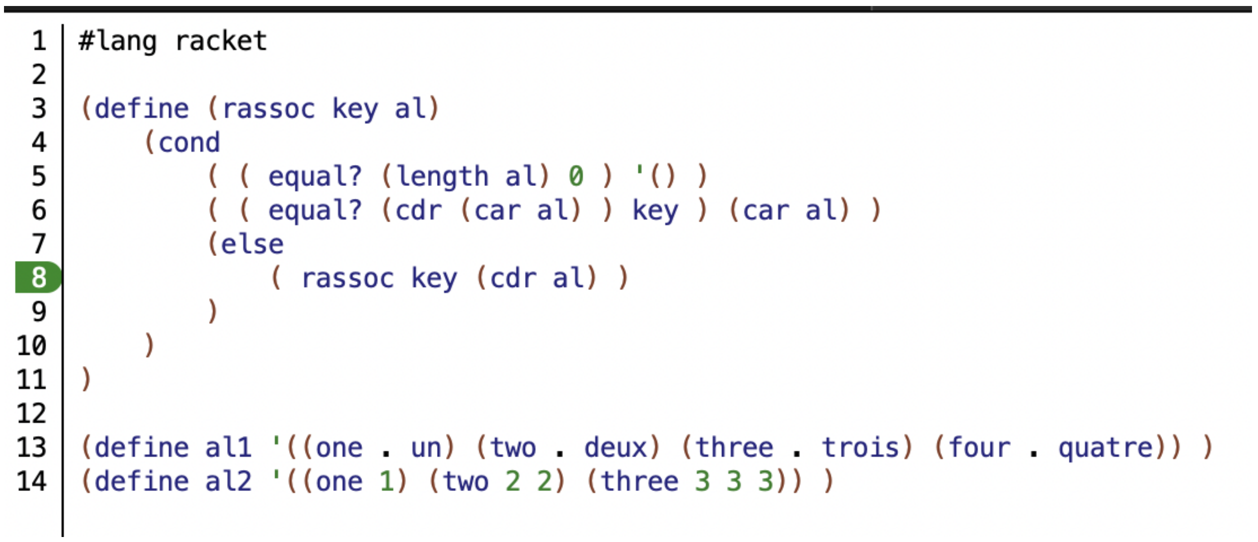
Task 3: Assoc

```racket
#lang racket

(define (assoc key al)
    (cond
        ( ( equal? (length al) 0 ) '() )
        ( ( equal? (car (car al) ) key ) (car al) )
        (else
            ( assoc key (cdr al) )
        )
    )
)

(define al1 '((one . un) (two . deux) (three . trois) (four . quatre)) )
(define al2 '((one 1) (two 2 2) (three 3 3 3)) )


```

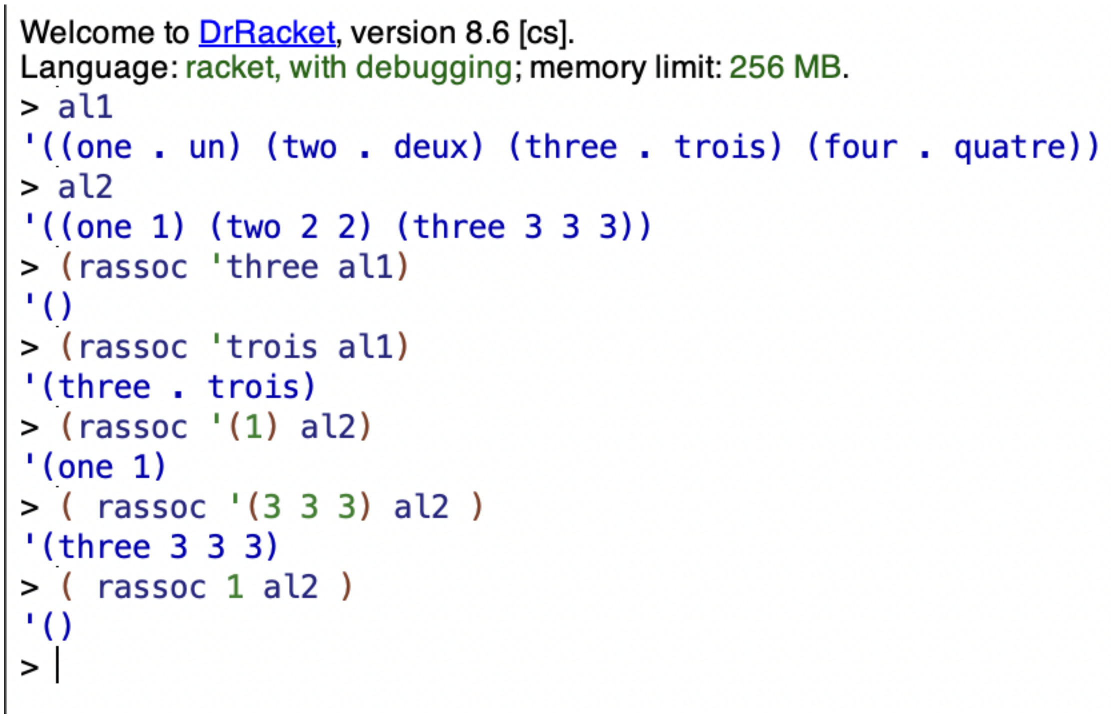


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (assoc 'two al1)
'(two . deux)
> (assoc 'five al1)
'()
> (assoc 'three al2)
'(three 3 3 3)
> (assoc 'four al2)
'()
> 
```

Task 4: Rassoc

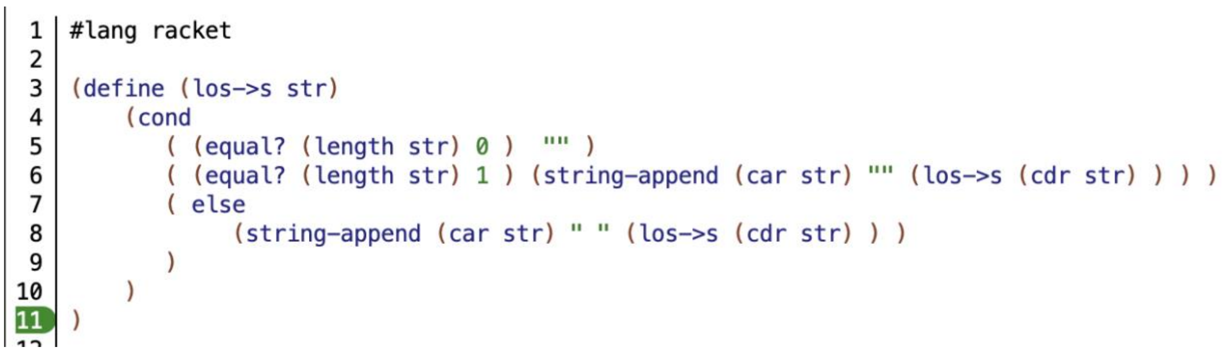


```
#lang racket

(define (rassoc key al)
    (cond
        ( ( equal? (length al) 0 ) '() )
        ( ( equal? (cdr (car al) ) key ) (car al) )
        (else
            ( rassoc key (cdr al) )
        )
    )
)

(define al1 '((one . un) (two . deux) (three . trois) (four . quatre)) )
(define al2 '((one 1) (two 2 2) (three 3 3 3)) )
```

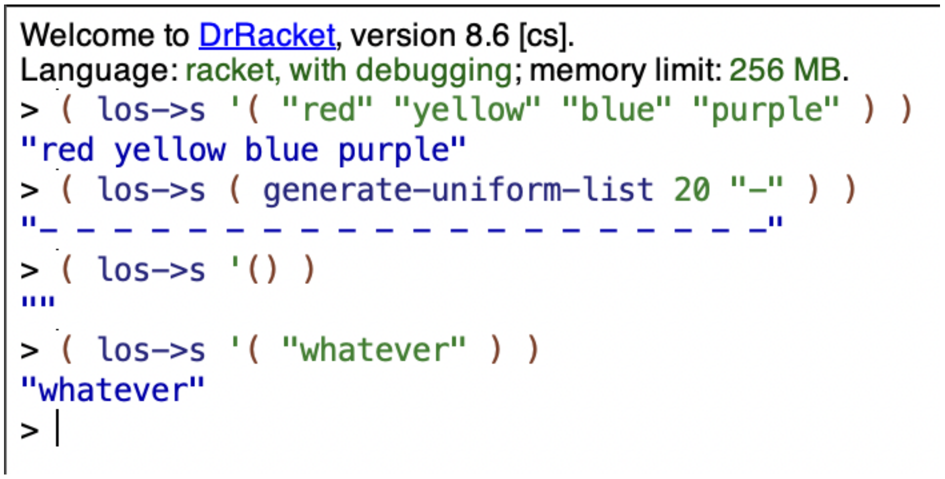


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (rassoc 'three al1)
'()
> (rassoc 'trois al1)
'(three . trois)
> (rassoc '(1) al2)
'(one 1)
> ( rassoc '(3 3 3) al2 )
'(three 3 3 3)
> ( rassoc 1 al2 )
'()
> 
```

Task 5 : Los->s

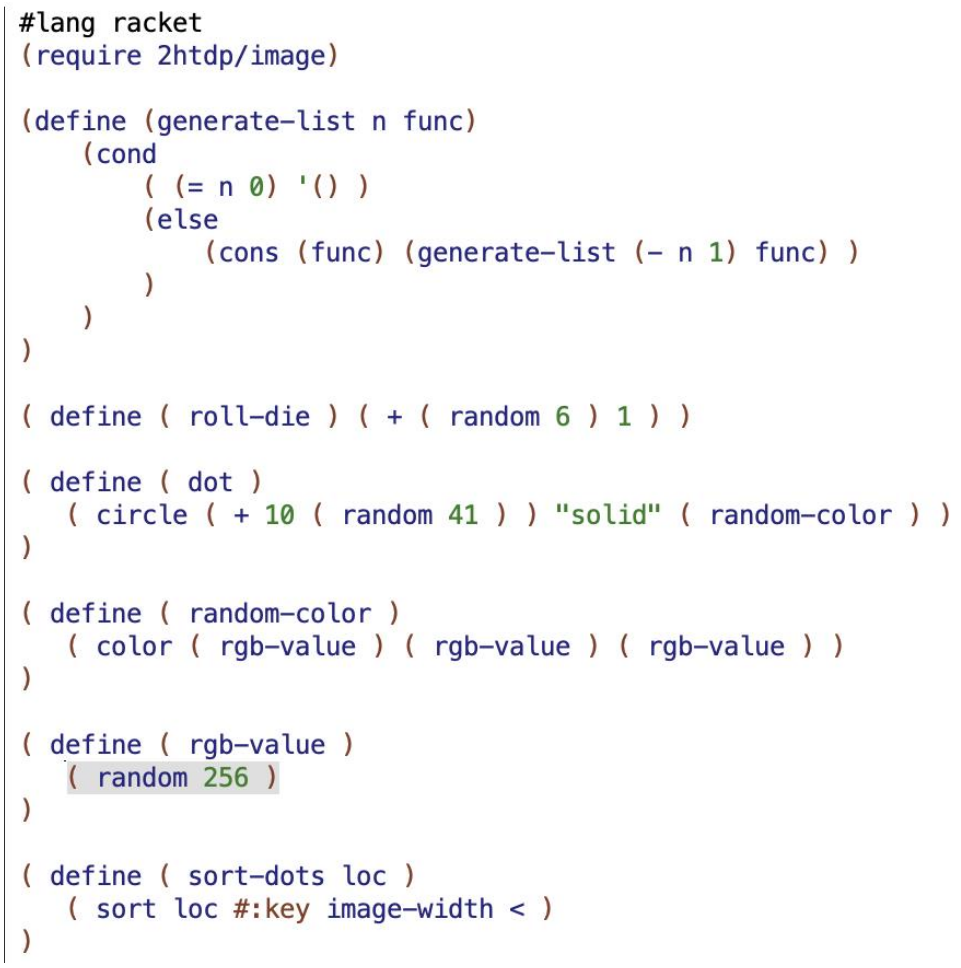


```
#lang racket

(define (los->s str)
    (cond
       ( (equal? (length str) 0 )  "" )
       ( (equal? (length str) 1 ) (string-append (car str) "" (los->s (cdr str) ) ) )
       ( else
            (string-append (car str) " " (los->s (cdr str) ) )
       )
    )
)
```

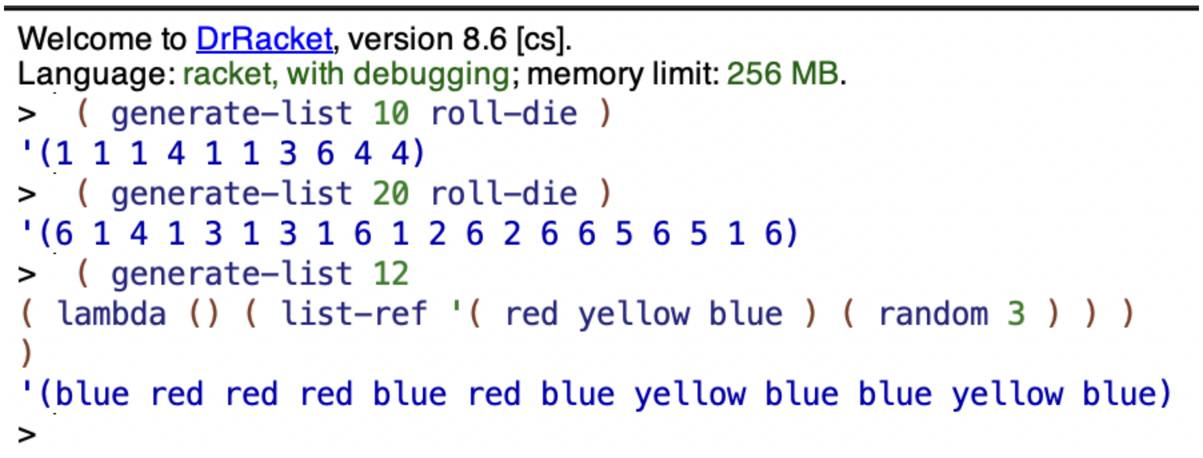


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> ( los->s '( "red" "yellow" "blue" "purple" ) )
"red yellow blue purple"
> ( los->s ( generate-uniform-list 20 "-" ) )
"- - - - - - - - - - - - - - - - - - - -"
> ( los->s '() )
""
> ( los->s '( "whatever" ) )
"whatever"
> 
```

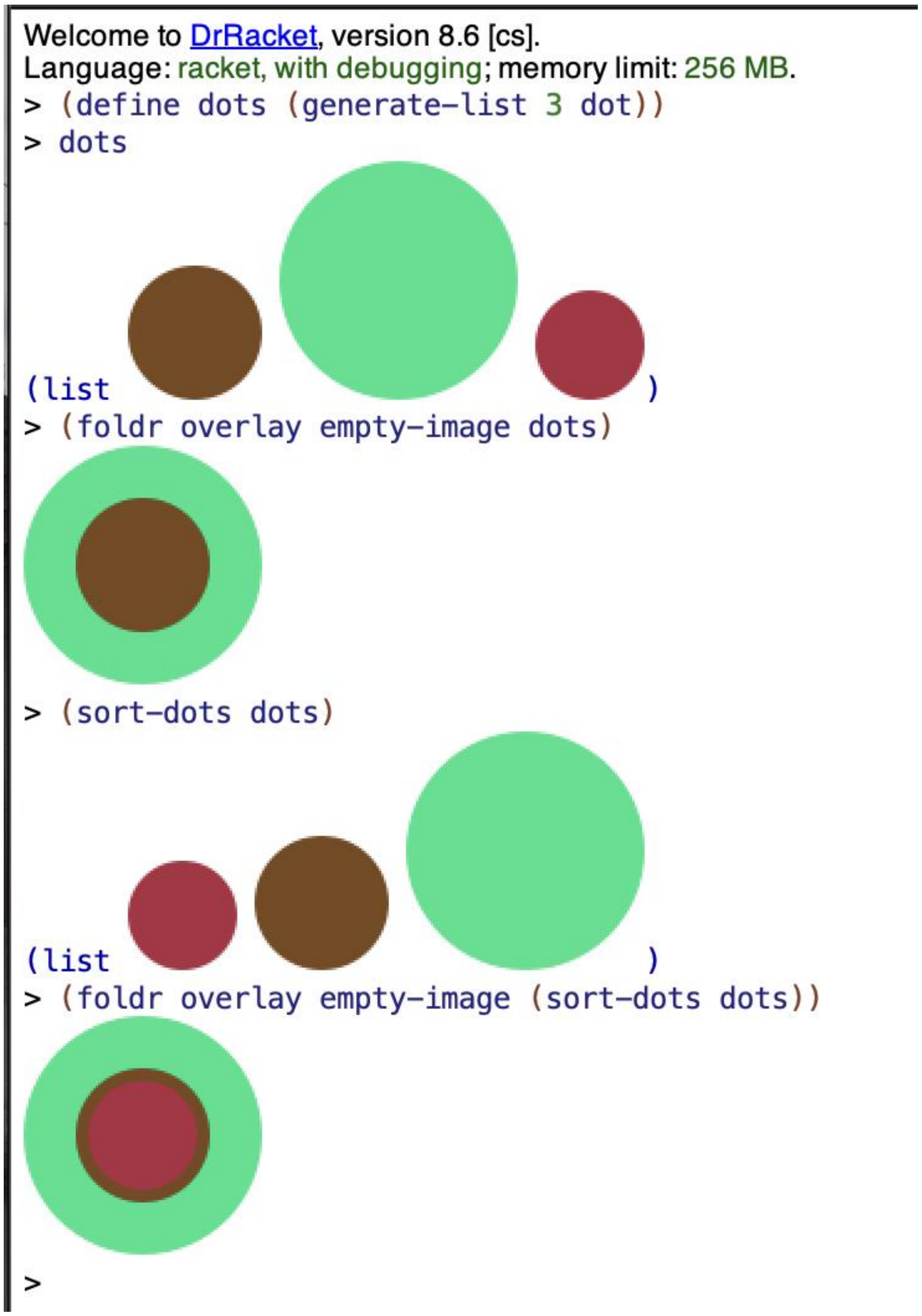
Task 6: Generate List



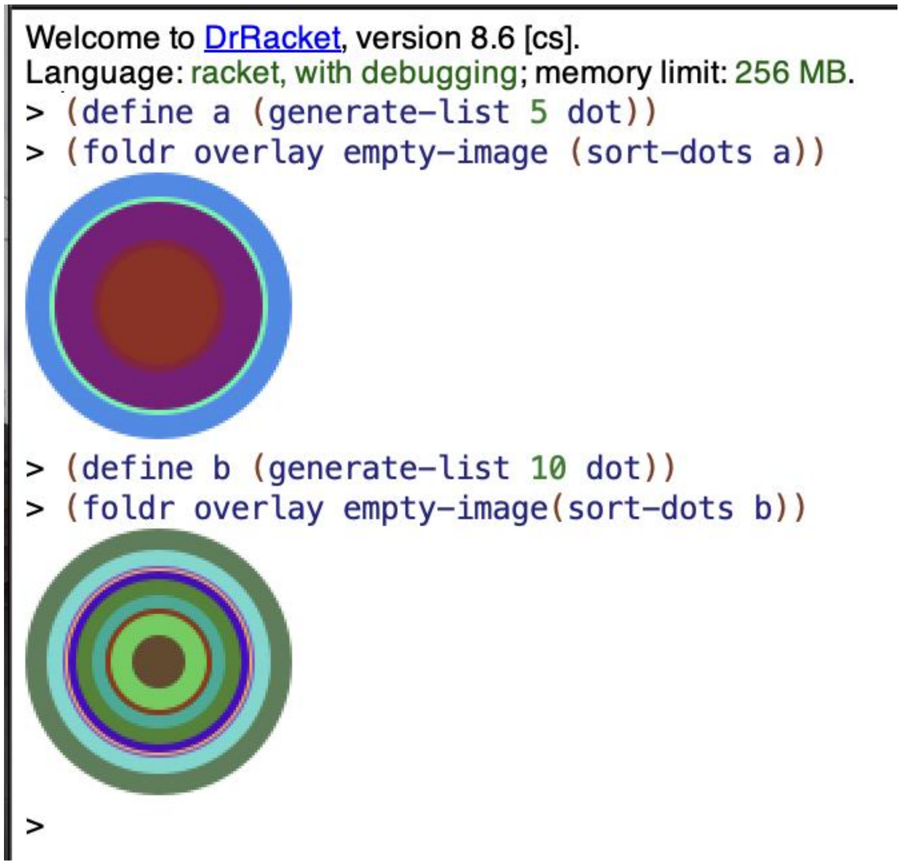
```
#lang racket
(require 2htdp/image)

(define (generate-list n func)
    (cond
        ( (= n 0) '() )
        (else
            (cons (func) (generate-list (- n 1) func) )
        )
    )
)

( 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 < )
)
```



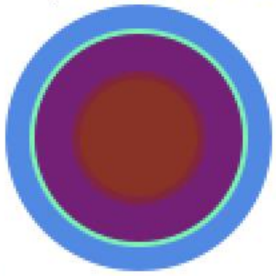
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
>  ( generate-list 10 roll-die )
'(1 1 1 4 1 1 3 6 4 4)
>  ( generate-list 20 roll-die )
'(6 1 4 1 3 1 3 1 6 1 2 6 2 6 6 5 6 5 1 6)
>  ( generate-list 12
( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) )
)
'(blue red red red blue red blue yellow blue blue yellow blue)
>
```



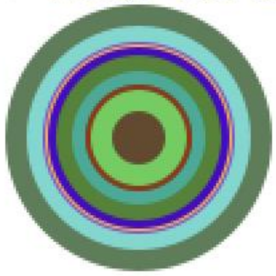
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (define dots (generate-list 3 dot))
> dots
(list                                    )
> (foldr overlay empty-image dots)

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

>
```



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (define a (generate-list 5 dot))
> (foldr overlay empty-image (sort-dots a))
```

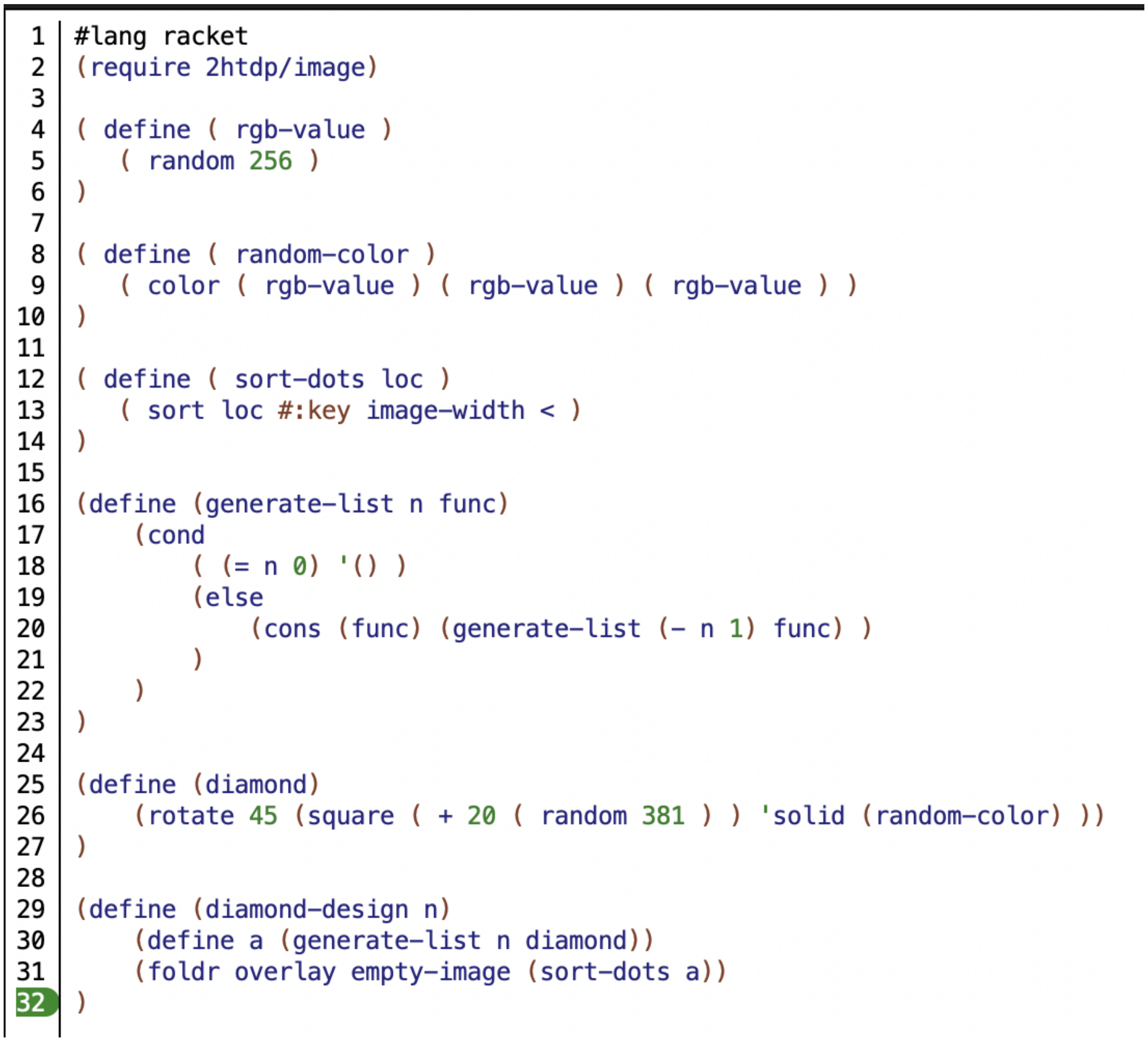


```
> (define b (generate-list 10 dot))
> (foldr overlay empty-image(sort-dots b))
```



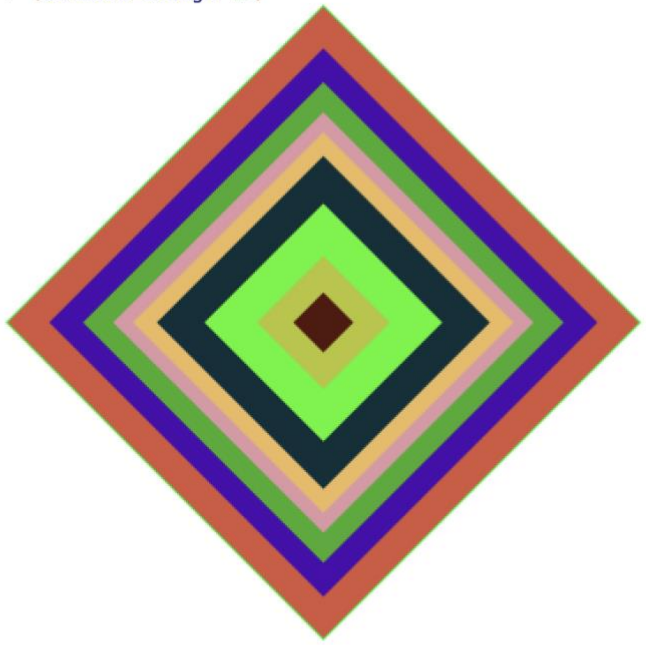
```
>
```

Task 7: The Diamond

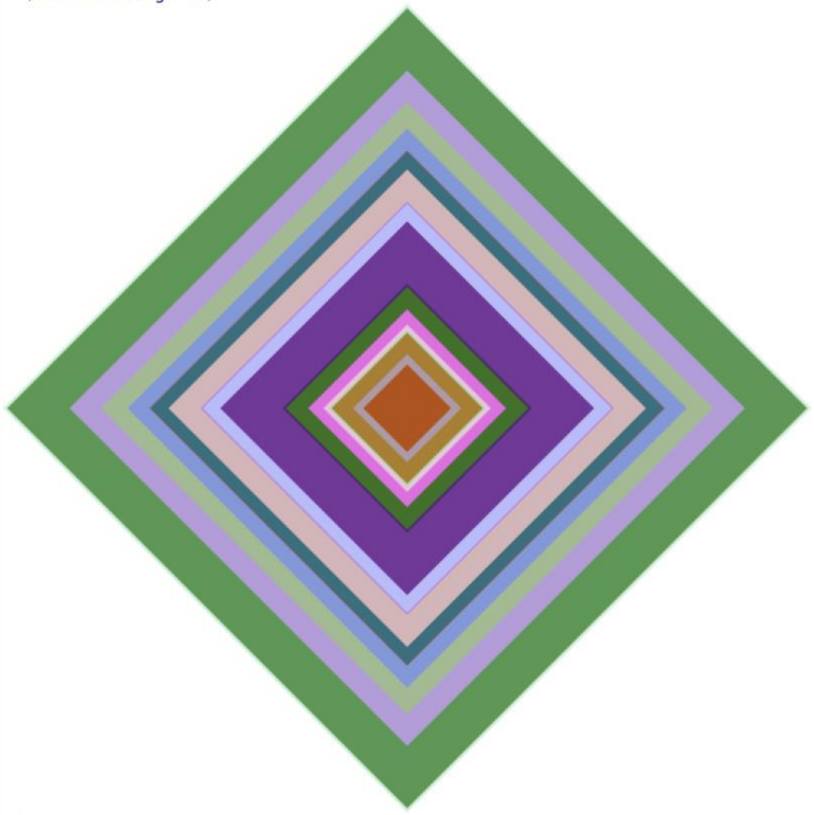


```
#lang racket
(require 2htdp/image)

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

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

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

(define (generate-list n func)
    (cond
        ( (= n 0) '() )
        (else
            (cons (func) (generate-list (- n 1) func) )
        )
    )
)

(define (diamond)
    (rotate 45 (square ( + 20 ( random 381 ) ) 'solid (random-color) ))
)

(define (diamond-design n)
    (define a (generate-list n diamond))
    (foldr overlay empty-image (sort-dots a))
)
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (diamond-design 10)
```

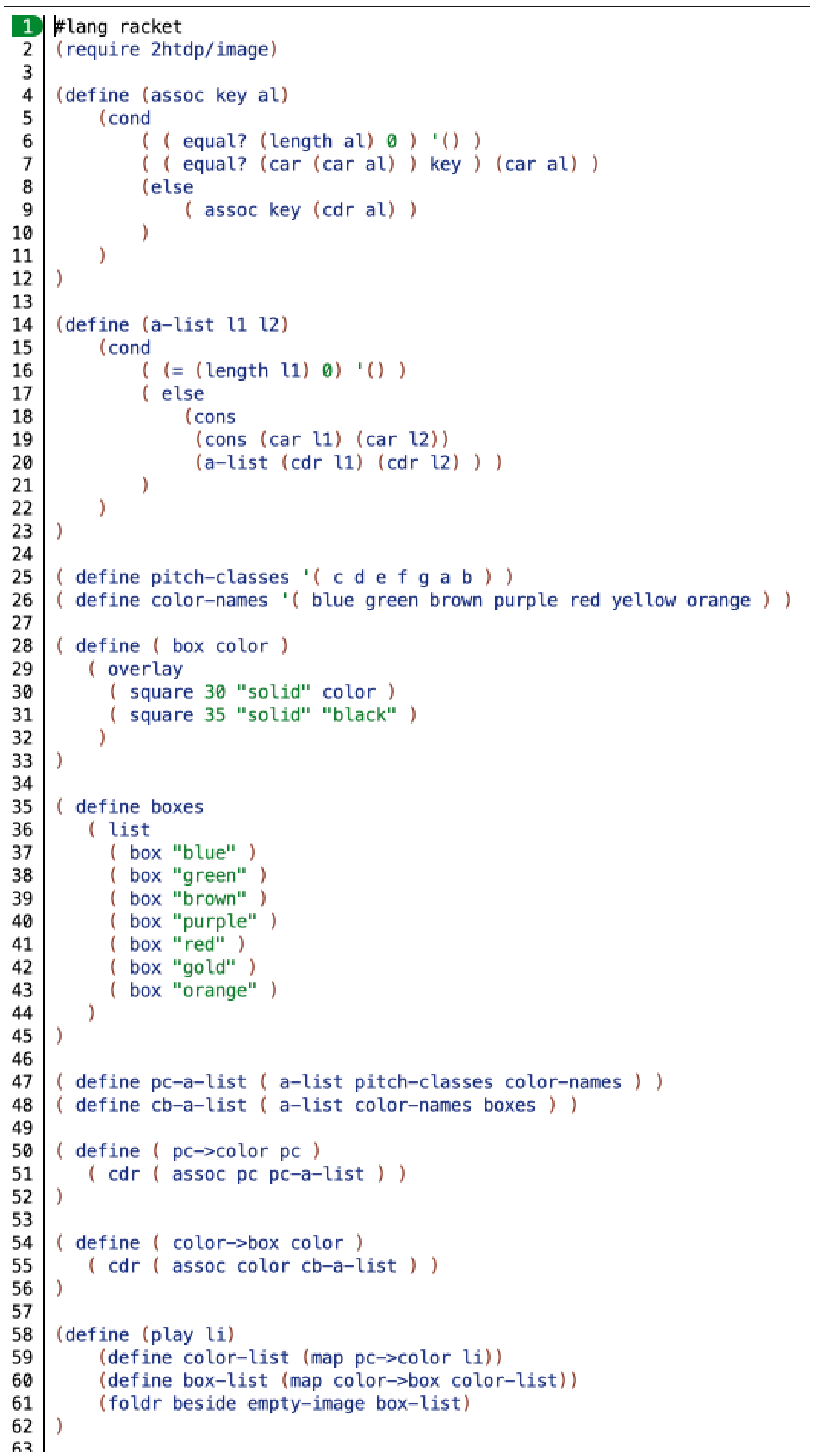


```
> (diamond-design 20)
```

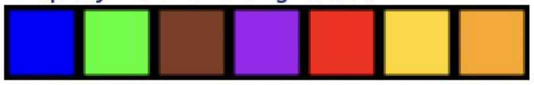


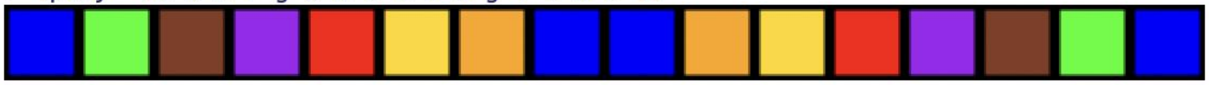
```
> |
```

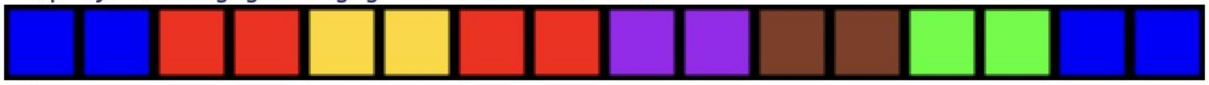
Task 8: Chromesthetic Renderings

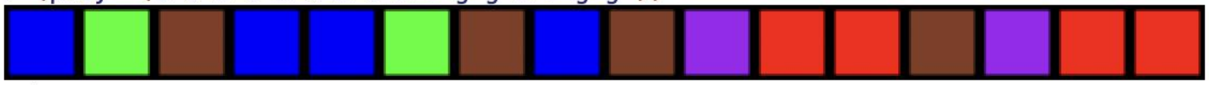


```racket
#lang racket
(require 2htdp/image)

(define (assoc key al)
    (cond
        ( ( equal? (length al) 0 ) '() )
        ( ( equal? (car (car al) ) key ) (car al) )
        (else
            ( assoc key (cdr al) )
        )
    )
)

(define (a-list l1 l2)
    (cond
        ( (= (length l1) 0) '() )
        ( else
            (cons
             (cons (car l1) (car l2))
             (a-list (cdr l1) (cdr l2) ) )
        )
    )
)

( 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 li)
    (define color-list (map pc->color li))
    (define box-list (map color->box color-list))
    (foldr beside empty-image box-list)
)
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (play '(c d e f g a b))
```


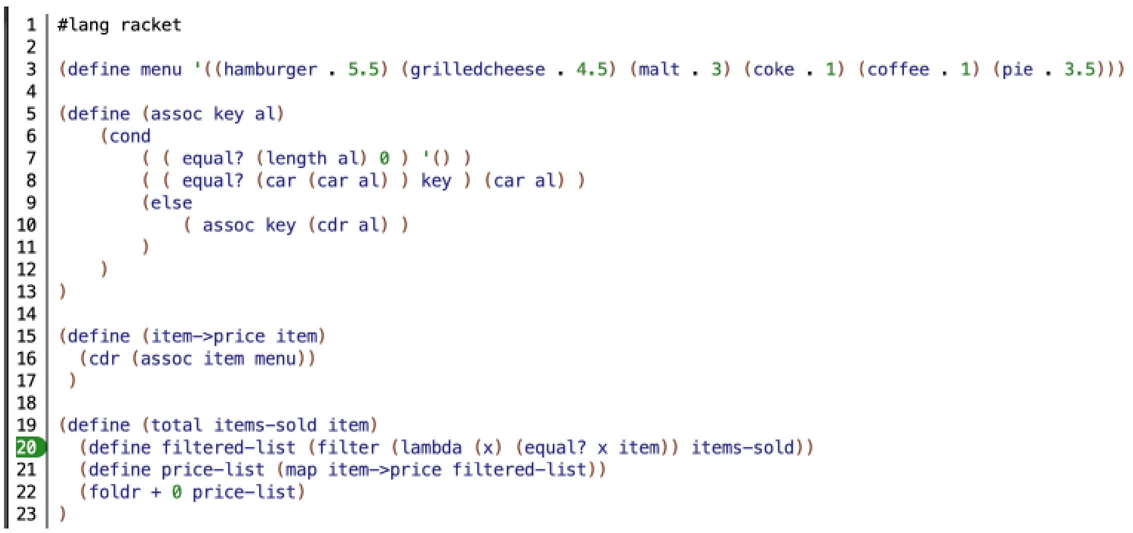
```
> (play '(c d e f g a b c c b a g f e d c ))
```


```
> (play '(c c g g a a g g f f e e d d c c ))
```


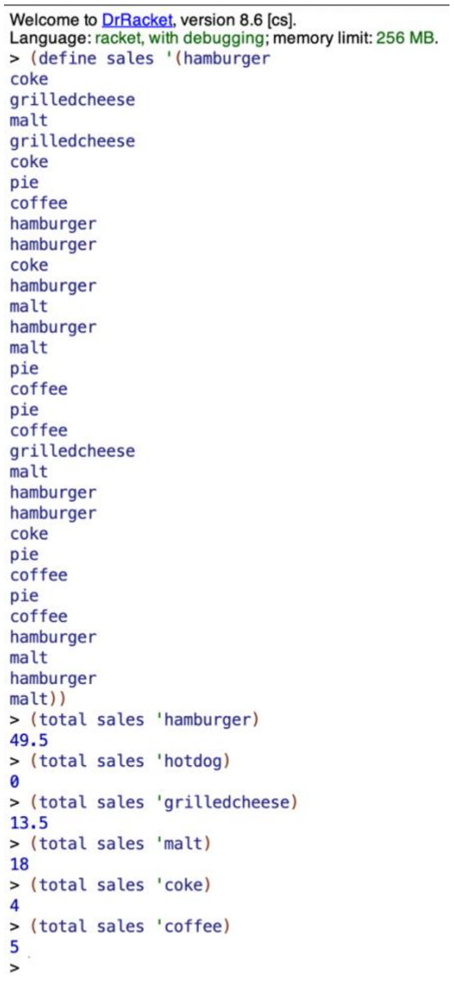
```
> (play '(c d e c c d e c e f g g e f g g ))
```


```
> |
```

Task 9: Diner



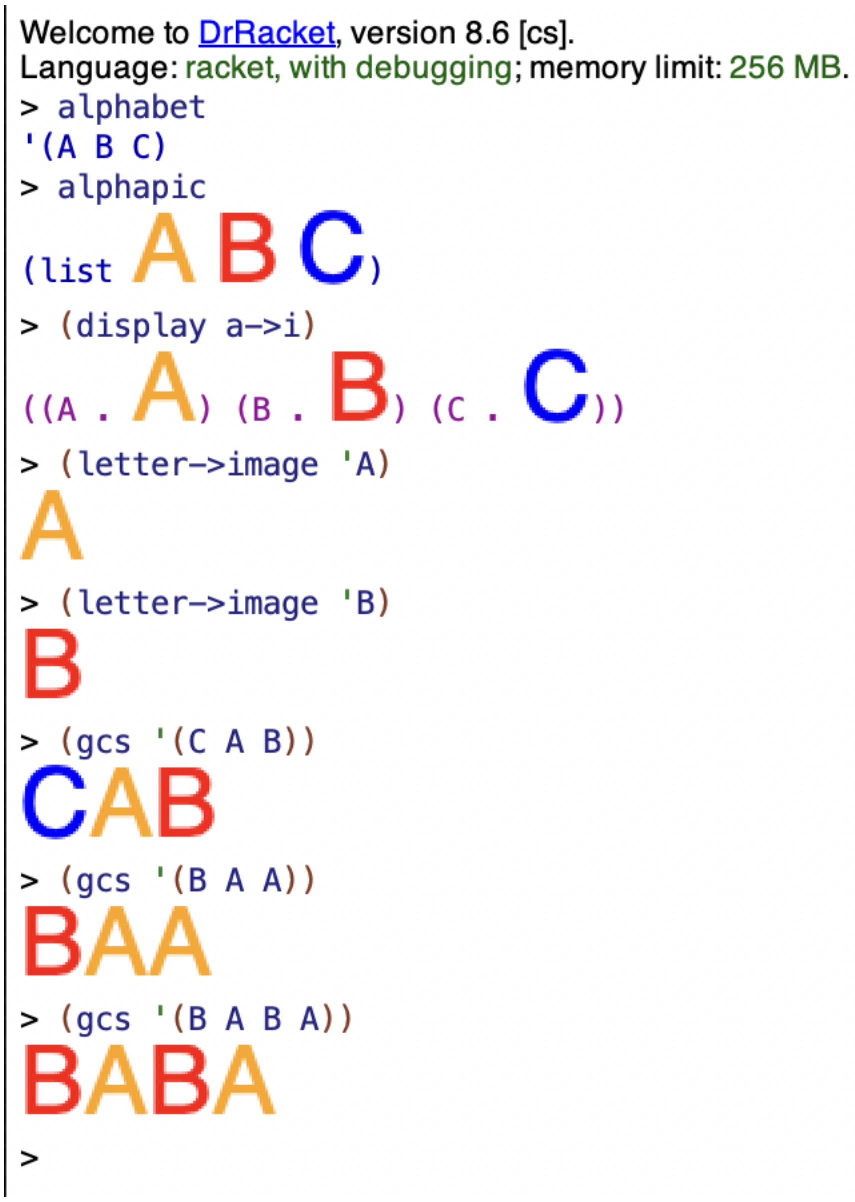
```
#lang racket

(define menu '((hamburger . 5.5) (grilledcheese . 4.5) (malt . 3) (coke . 1) (coffee . 1) (pie . 3.5)))

(define (assoc key al)
    (cond
        ( ( equal? (length al) 0 ) '() )
        ( ( equal? (car (car al) ) key ) (car al) )
        (else
            ( assoc key (cdr al) )
        )
    )
)

(define (item->price item)
  (cdr (assoc item menu))
 )

(define (total items-sold item)
  (define filtered-list (filter (lambda (x) (equal? x item)) items-sold))
  (define price-list (map item->price filtered-list))
  (foldr + 0 price-list)
)
```



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (define sales '(hamburger
coke
grilledcheese
malt
grilledcheese
coke
pie
coffee
hamburger
hamburger
coke
hamburger
malt
hamburger
malt
pie
coffee
pie
coffee
grilledcheese
malt
hamburger
hamburger
coke
pie
coffee
pie
coffee
hamburger
malt
hamburger
malt))
> (total sales 'hamburger)
49.5
> (total sales 'hotdog)
0
> (total sales 'grilledcheese)
13.5
> (total sales 'malt)
18
> (total sales 'coke)
4
> (total sales 'coffee)
5
>
```

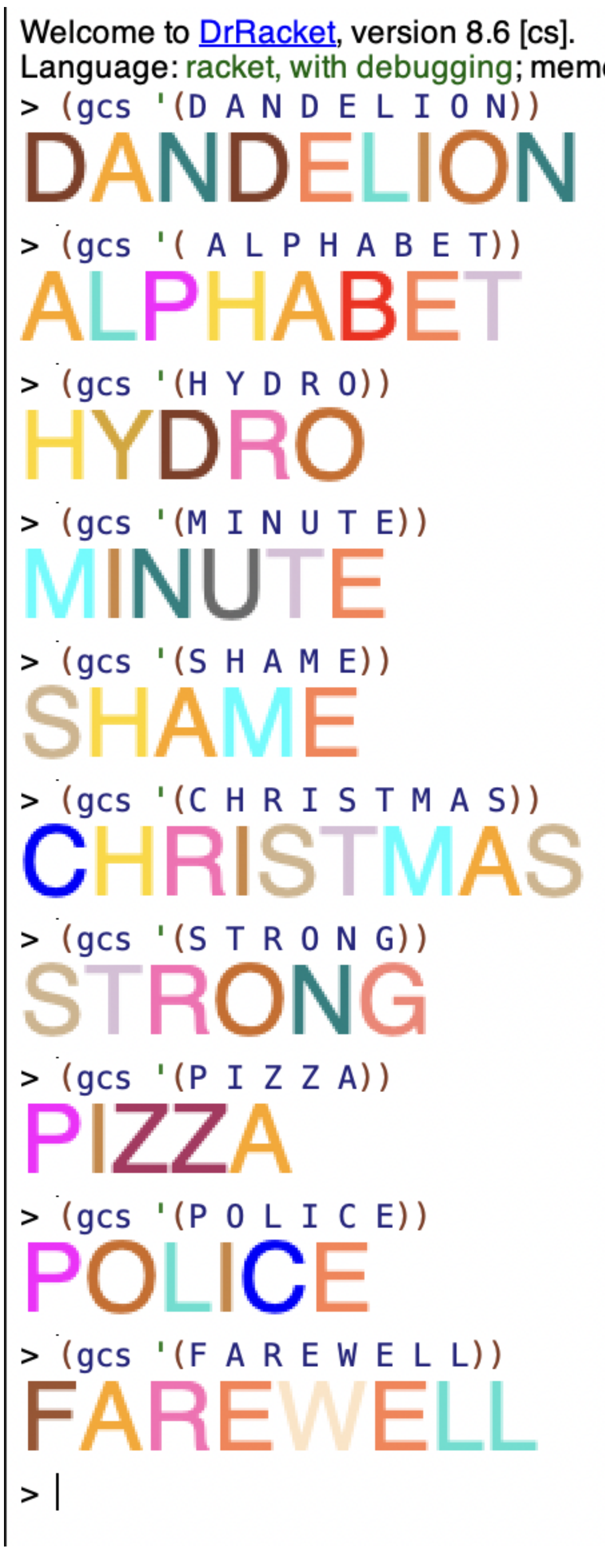
Task 10: Grapheme Color Synthesia

```
#lang racket
(require 2htdp/image)

(define (a-list l1 l2)
    (cond
        ( (= (length l1) 0) '() )
        ( else
            (cons
             (cons (car l1) (car l2))
             (a-list (cdr l1) (cdr l2) ) )
        )
    )
)

( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )


( define alphabet '(A B C) )

( define alphapic ( list AI BI CI ) )

( define a->i ( a-list alphabet alphapic ) )

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

(define (gcs letters)
    (define images (map letter->image letters))
    (foldr beside empty-image images)
)
```



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 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
>
```

```
( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
( define DI (text "D" 36 "brown") )
( define EI (text "E" 36 "coral") )

( define FI (text "F" 36 "sienna") )
( define GI (text "G" 36 "salmon") )
( define HI (text "H" 36 "gold") )
( define II (text "I" 36 "peru") )
( define JI (text "J" 36 "chartreuse") )

( define KI (text "K" 36 "aquamarine") )
( define LI (text "L" 36 "turquoise") )
( define MI (text "M" 36 "cyan") )
( define NI (text "N" 36 "teal") )
( define OI (text "O" 36 "chocolate") )

( define PI (text "P" 36 "fuchsia") )
( define QI (text "Q" 36 "limegreen") )
( define RI (text "R" 36 "hotpink") )
( define SI (text "S" 36 "tan") )
( define TI (text "T" 36 "thistle") )

( define UI (text "U" 36 "dimgray") )
( define VI (text "V" 36 "indigo") )
( define WI (text "W" 36 "bisque") )
( define XI (text "X" 36 "wheat") )
( define YI (text "Y" 36 "goldenrod") )
( define ZI (text "Z" 36 "maroon") )


( 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)
    (define images (map letter->image letters))
    (foldr beside empty-image images)
)
```



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (gcs '(D A N D E L I O N))
DANDELION
> (gcs '( A L P H A B E T))
ALPHABET
> (gcs '(H Y D R O))
HYDRO
> (gcs '(M I N U T E))
MINUTE
> (gcs '(S H A M E))
SHAME
> (gcs '(C H R I S T M A S))
CHRISTMAS
> (gcs '(S T R O N G))
STRONG
> (gcs '(P I Z Z A))
PIZZA
> (gcs '(P O L I C E))
POLICE
> (gcs '(F A R E W E L L))
FAREWELL
> |
```