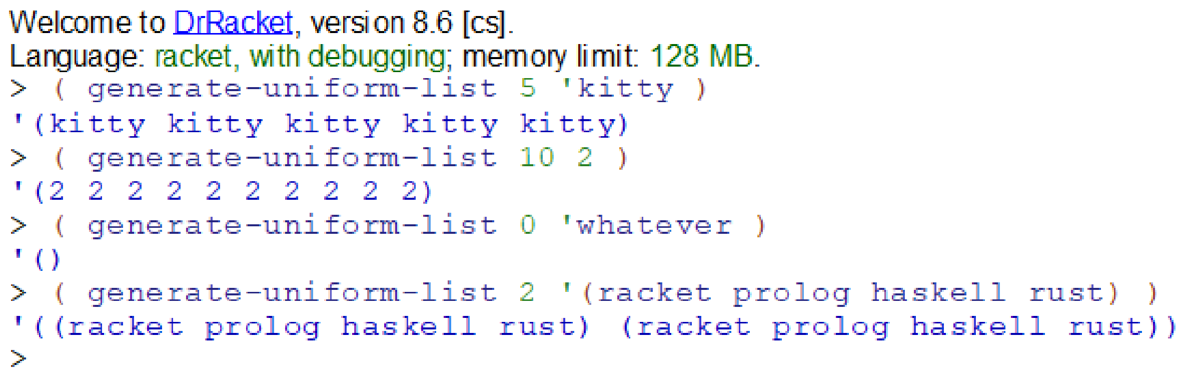
Racket Programming Assignment #4: RLP and HoFs

Learning Abstract:

This assignment consists of various recursive lisp and high order functions. It generates random objects, colors and numbers in a repeated pattern. The diamond and circles display different sizes and randomize colors.

Task 1- Generate Uniform List

Demo

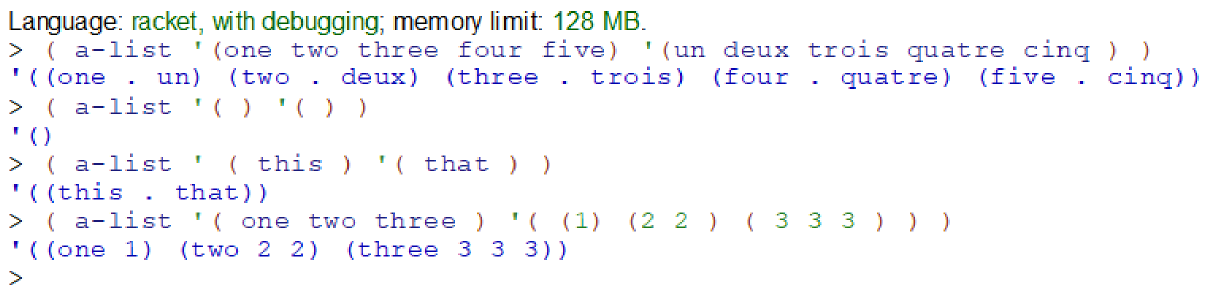


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-uniform-list 5 'kitty )
'(kitty kitty kitty kitty kitty)
> ( generate-uniform-list 10 2 )
'(2 2 2 2 2 2 2 2 2 2)
> ( generate-uniform-list 0 'whatever )
'()
> ( generate-uniform-list 2 '(racket prolog haskell rust) )
'((racket prolog haskell rust) (racket prolog haskell rust))
>
```

Task 2 - Association List Generator

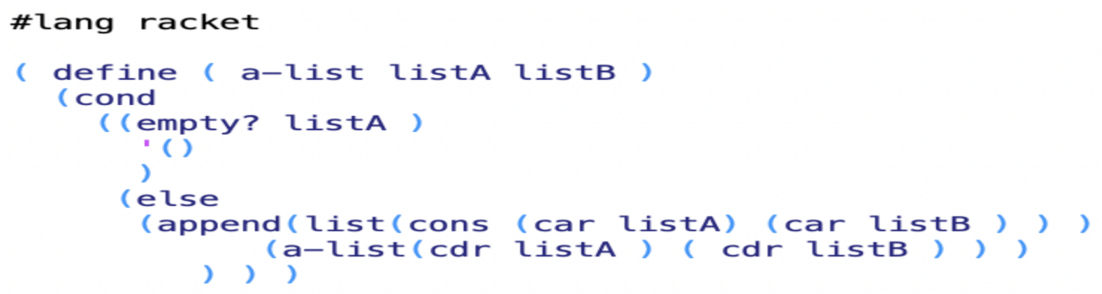
```
#lang racket
( define (a-list equal-listA equal-listB)
   (cond
     ((empty? equal-listA )
      (list)
      )

     (else
      (append(list(cons (car equal-listA) (car equal-listB ) ) )
             (a-list(cdr equal-listA ) (cdr equal-listB ) ) )
      ) ) )
```

Demo

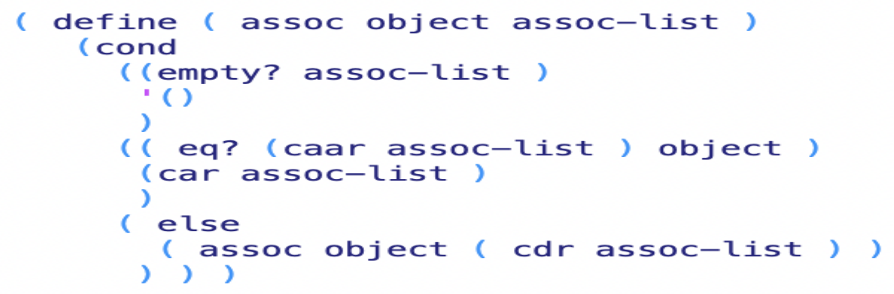


```
Language: racket, with debugging; memory limit: 128 MB.
> ( a-list '(one two three four five) '(un deux trois quatre cinq ) )
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
> ( a-list '( ) '( ) )
'()
> ( a-list ' ( this ) '( that ) )
'((this . that))
> ( a-list '( one two three ) '( (1) (2 2 ) ( 3 3 3 ) ) )
'((one 1) (two 2 2) (three 3 3 3))
>
```

Task 3 - Assoc

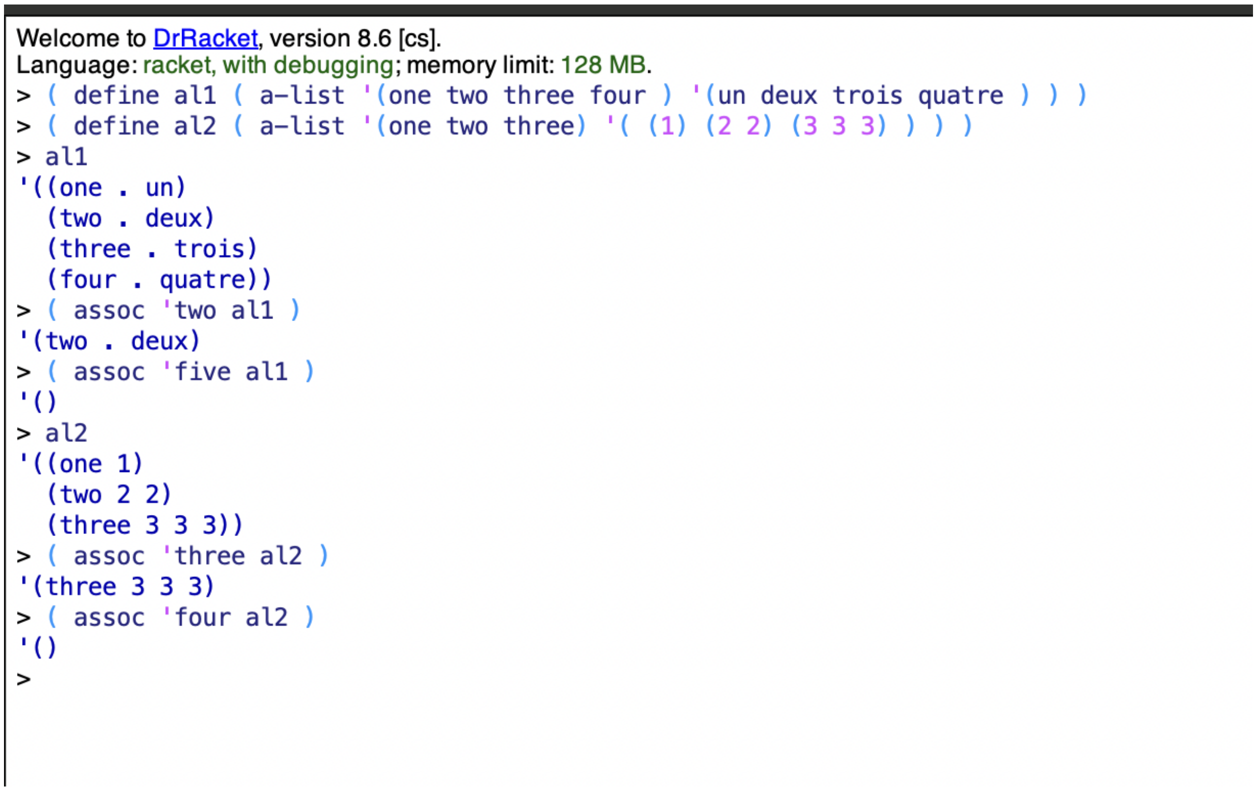


```
#lang racket

( define ( a-list listA listB )
   (cond
     ((empty? listA )
      '()
      )
     (else
      (append(list(cons (car listA) (car listB ) ) )
             (a-list(cdr listA ) ( cdr listB ) ) )
         ) ) )
```

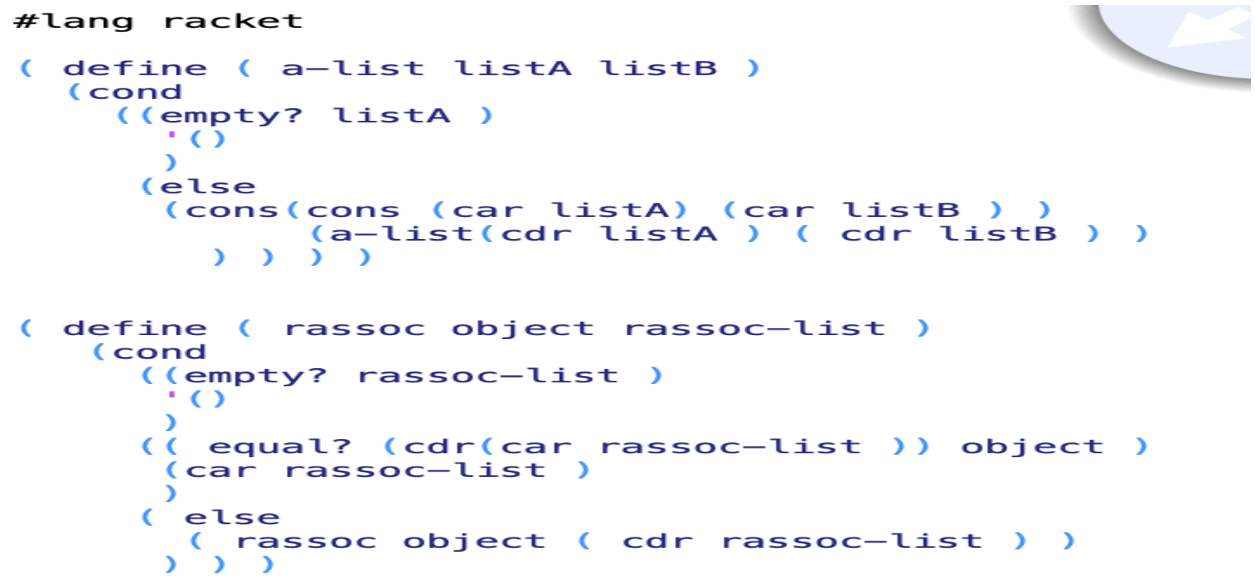


```
( define ( assoc object assoc-list )
    (cond
      ((empty? assoc-list )
       '()
       )
      (( eq? (caar assoc-list ) object )
       (car assoc-list )
       )
      ( else
        ( assoc object ( cdr assoc-list ) )
       ) ) )
```

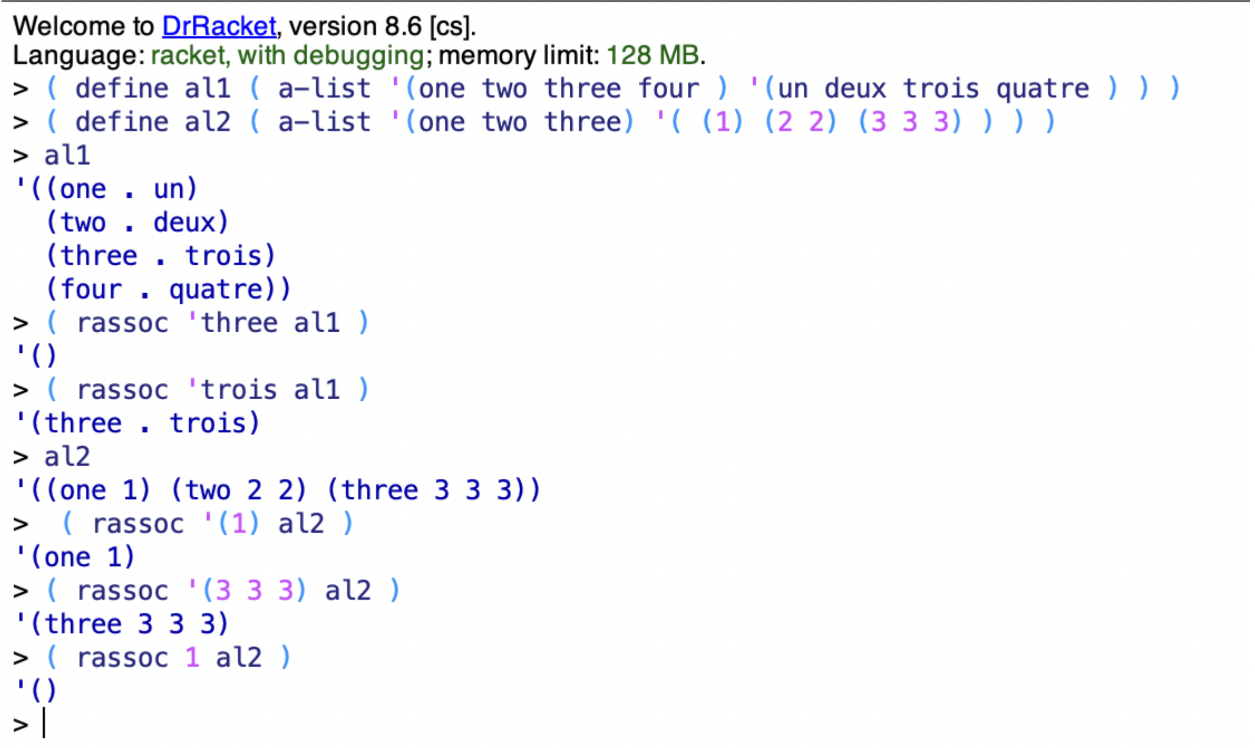
Demo


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( 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

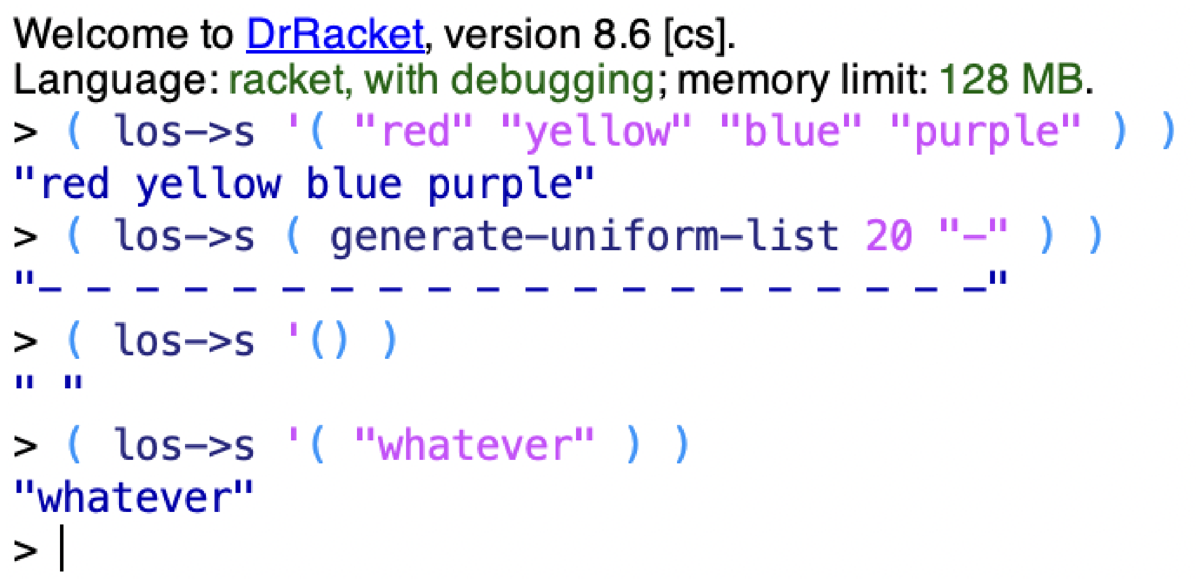

```
#lang racket

( define ( a-list listA listB )
   (cond
     ((empty? listA )
       '()
      )
      (else
       (cons(cons (car listA) (car listB ) )
             (a-list(cdr listA ) ( cdr listB ) )
        ) ) ) )

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

Demo



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( 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 )
'()
> 
```

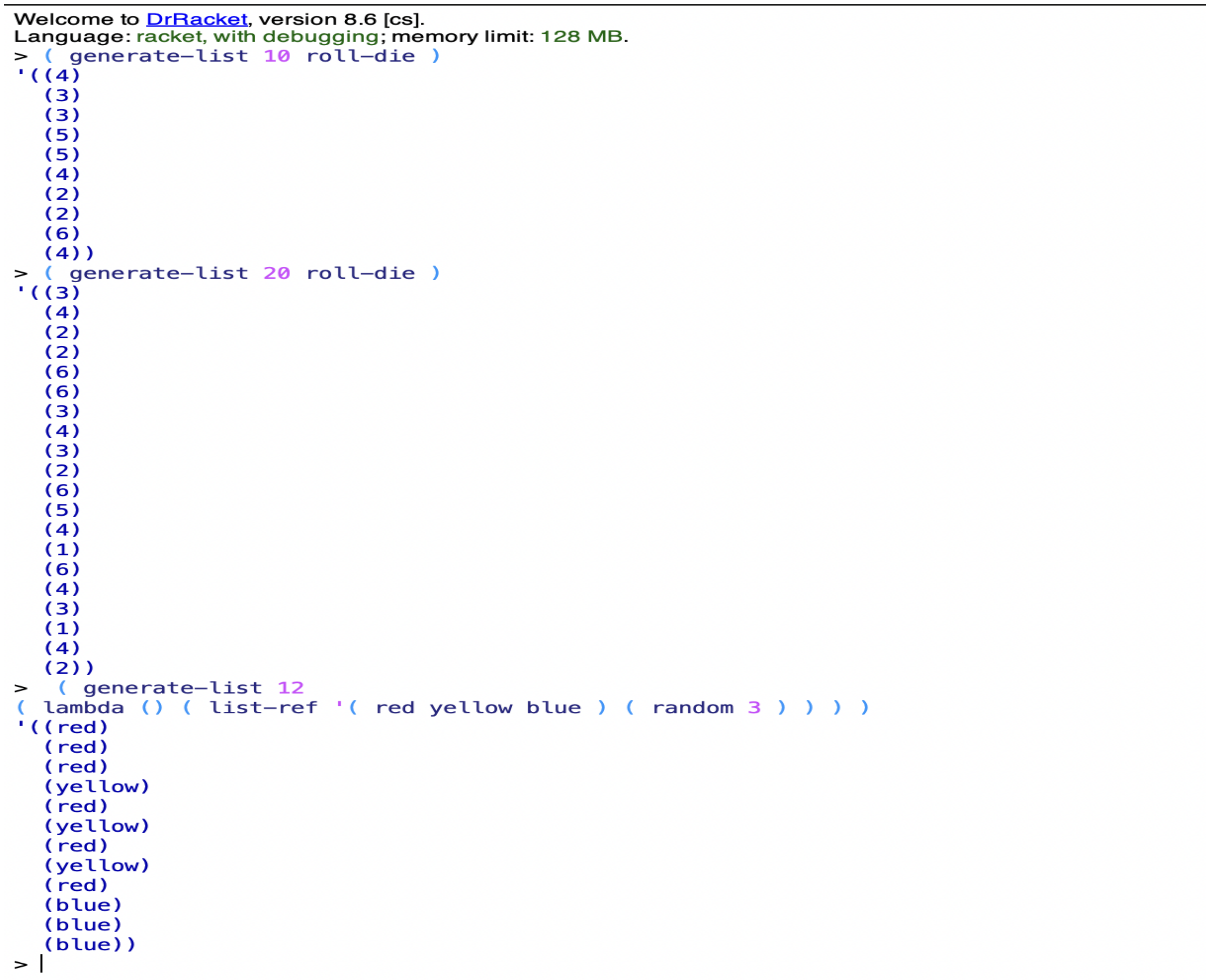
Task 5 - Los->s



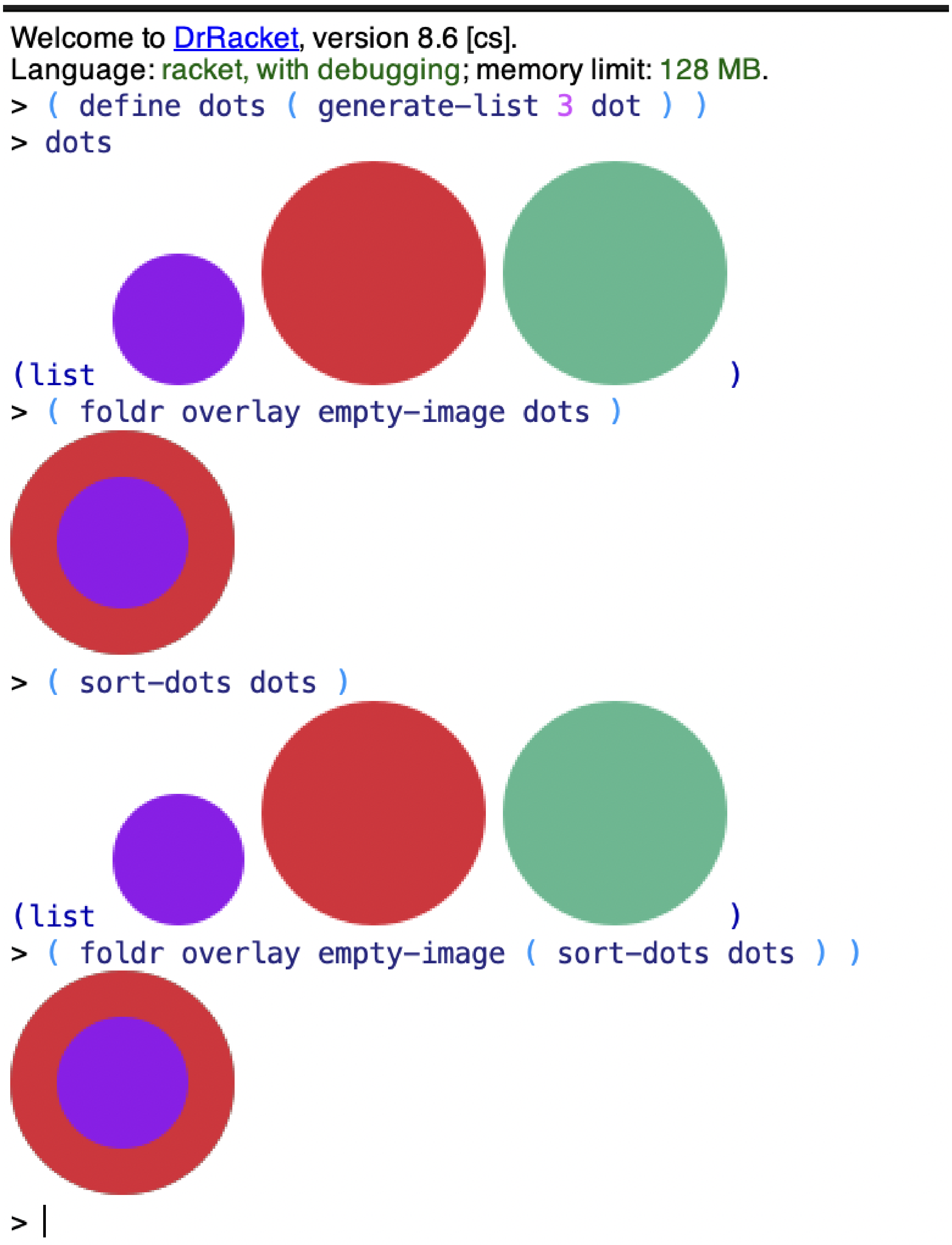
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( los->s '( "red" "yellow" "blue" "purple" ) )
"red yellow blue purple"
> ( los->s ( generate-uniform-list 20 "-" ) )
"- - - - - - - - - - - - - - - - - - - -"
> ( los->s '() )
" "
> ( los->s '( "whatever" ) )
"whatever"
> 
```

```
#lang racket
( define (generate-uniform-list unit object)
   (cond
     (( = unit 0 )
      (list )
      )
     ((> unit 0)
      (cons object (generate-uniform-list (- unit 1) object) ) ) ) )

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

Task 6 - Generate list

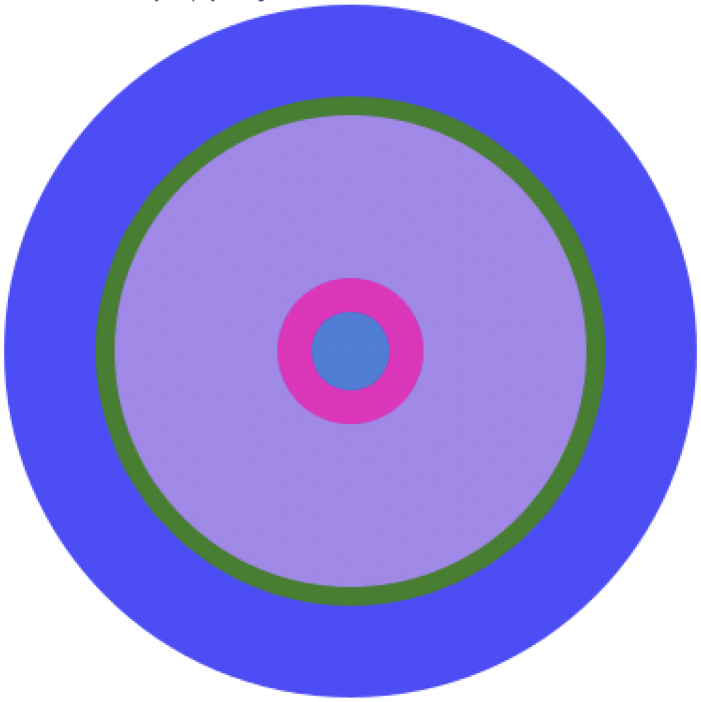


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-list 10 roll-die )
'((4)
  (3)
  (3)
  (5)
  (5)
  (4)
  (2)
  (2)
  (6)
  (4))
> ( generate-list 20 roll-die )
'((3)
  (4)
  (2)
  (2)
  (6)
  (6)
  (3)
  (4)
  (3)
  (2)
  (6)
  (5)
  (4)
  (1)
  (6)
  (4)
  (3)
  (1)
  (4)
  (2))
>  ( generate-list 12
( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) ) )
'((red)
  (red)
  (red)
  (yellow)
  (red)
  (yellow)
  (red)
  (yellow)
  (red)
  (blue)
  (blue)
  (blue))
> |
```

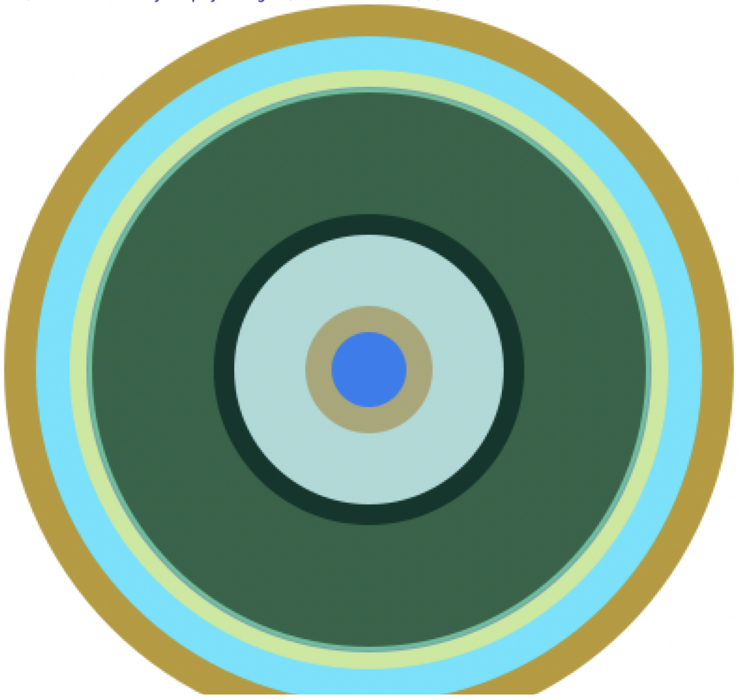


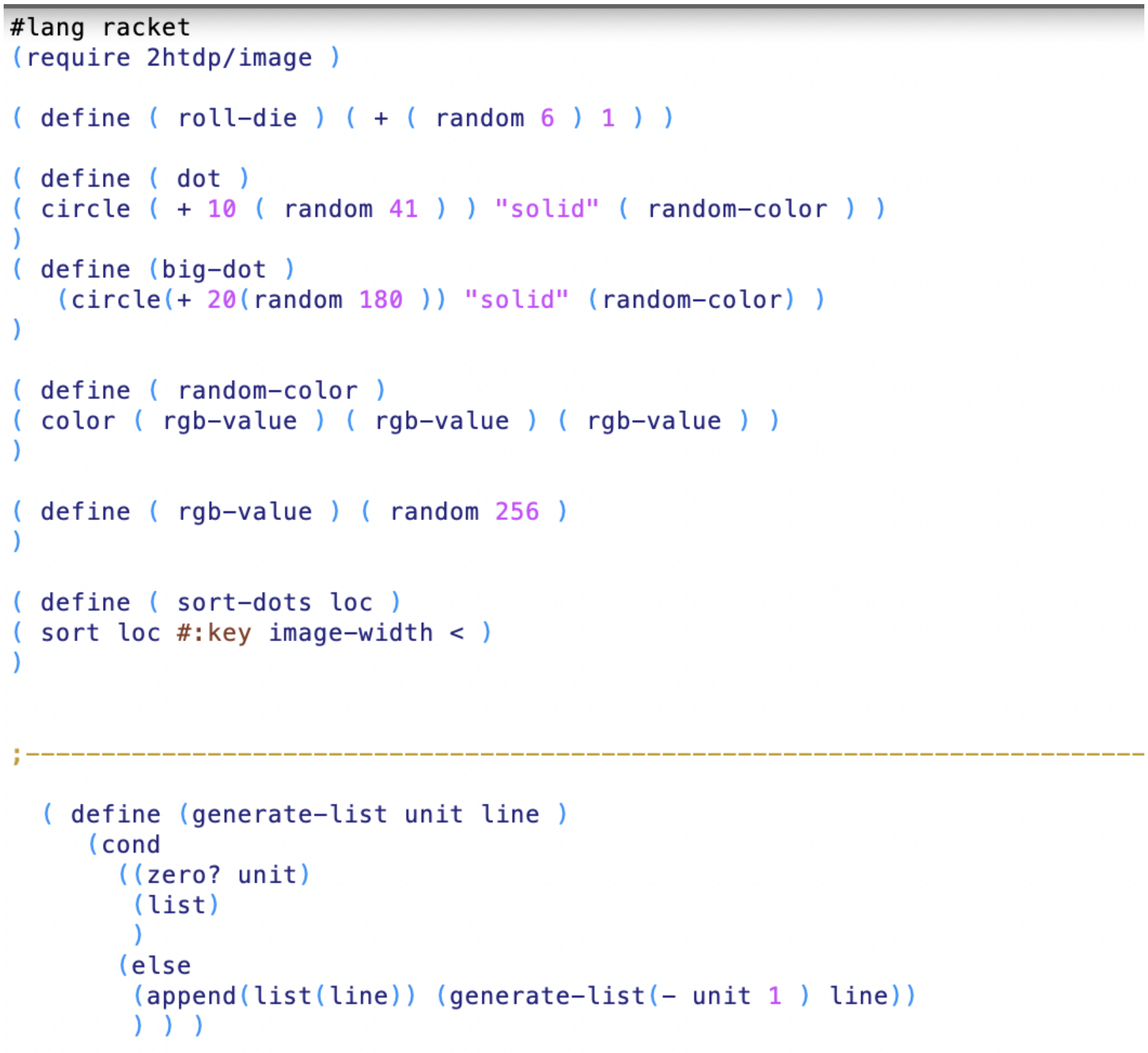
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define dots ( generate-list 3 dot ) )
> dots
(list                                         )
> ( foldr overlay empty-image dots )

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

> 
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define a ( generate-list 5 big-dot ) )

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



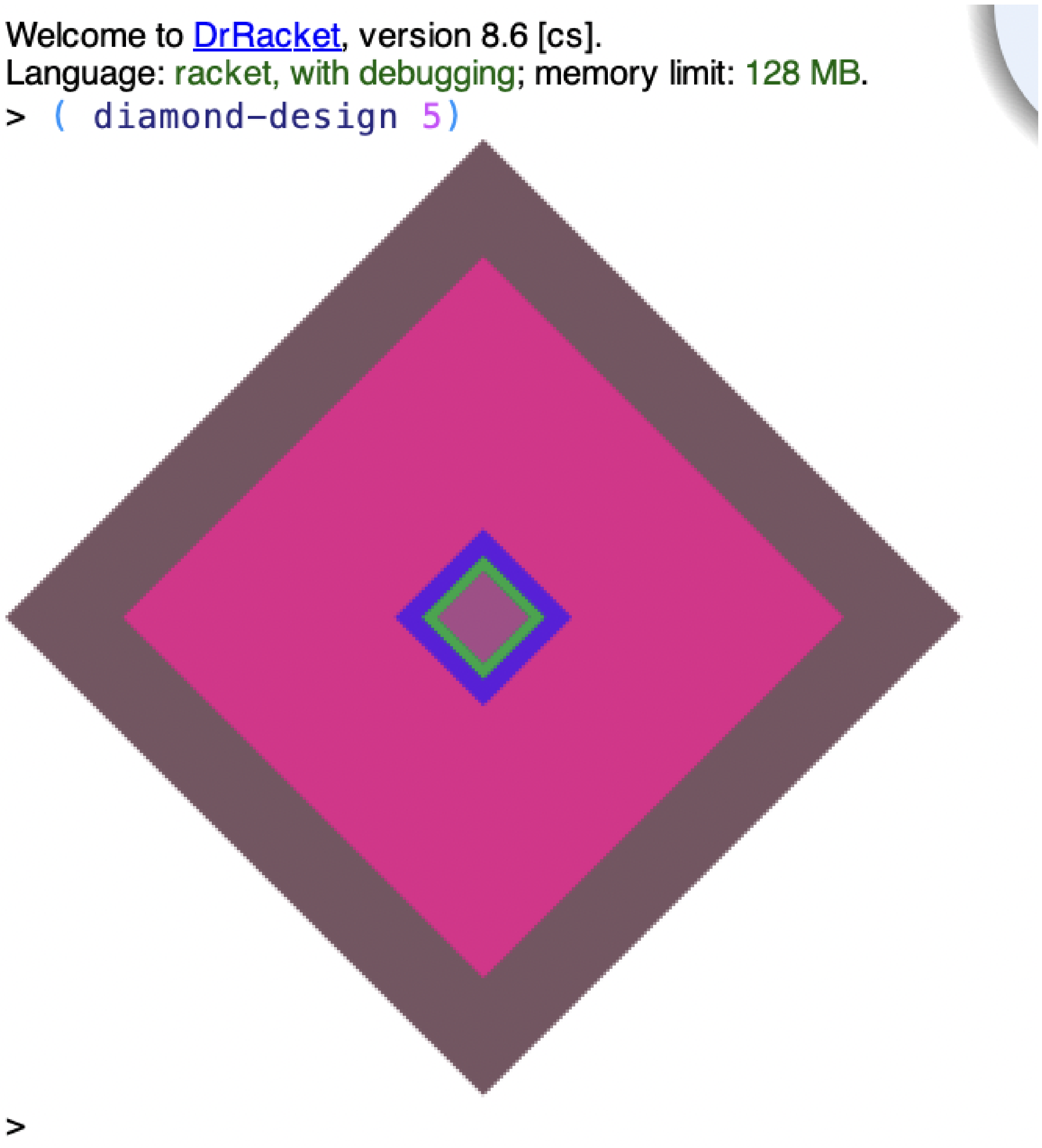
```
> ( define b ( generate-list 10 big-dot ) )
> ( foldr overlay empty-image ( sort-dots b ) )
```





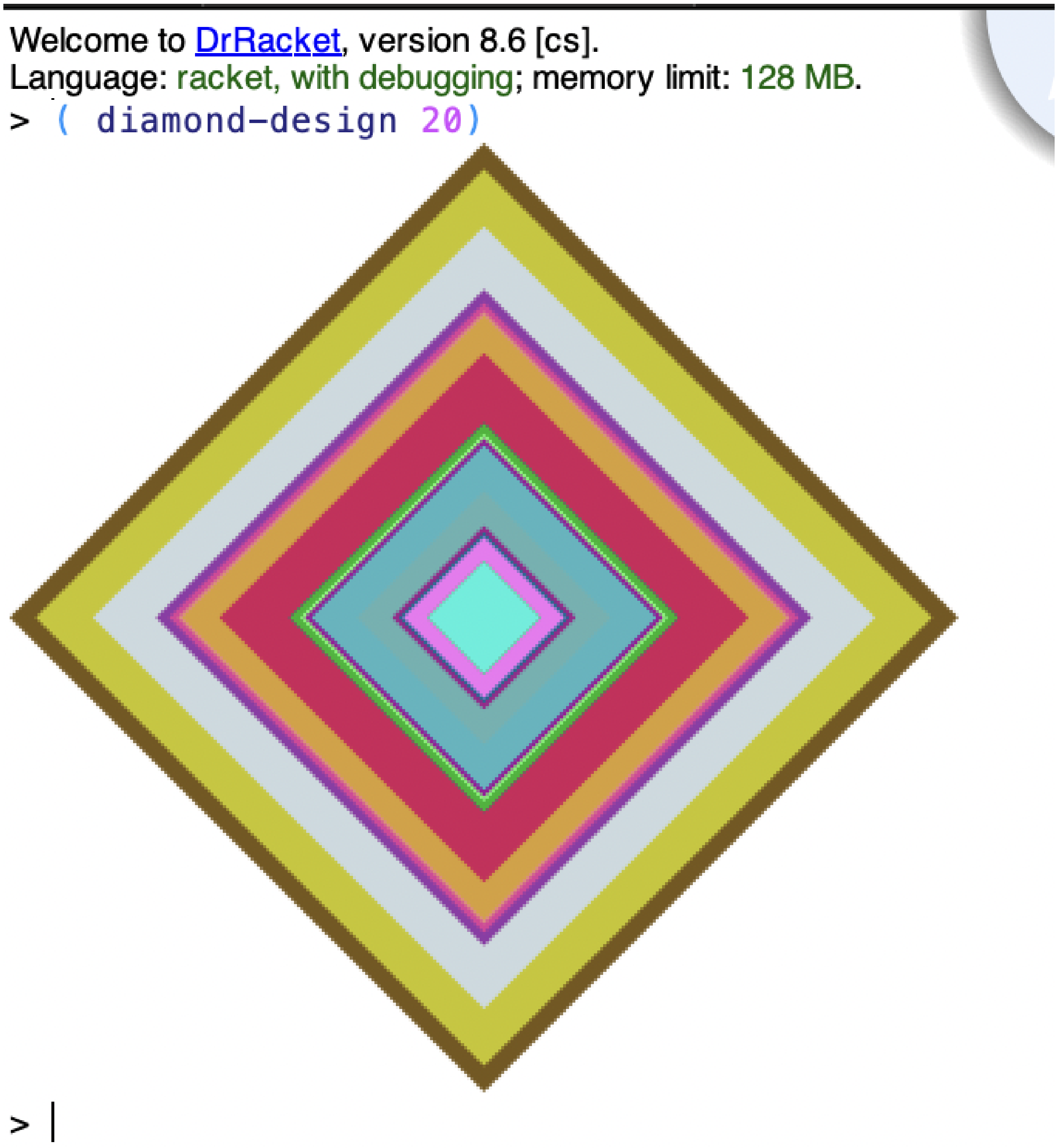
```
#lang racket
(require 2htdp/image )

( define ( roll-die ) ( + ( random 6 ) 1 ) )

( define ( dot )
( circle ( + 10 ( random 41 ) ) "solid" ( random-color ) )
)
( define (big-dot )
   (circle(+ 20(random 180 )) "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 unit line )
     (cond
       ((zero? unit)
        (list)
        )
       (else
        (append(list(line)) (generate-list(- unit 1 ) line))
        ) ) )
```

Task 7 - The Diamond

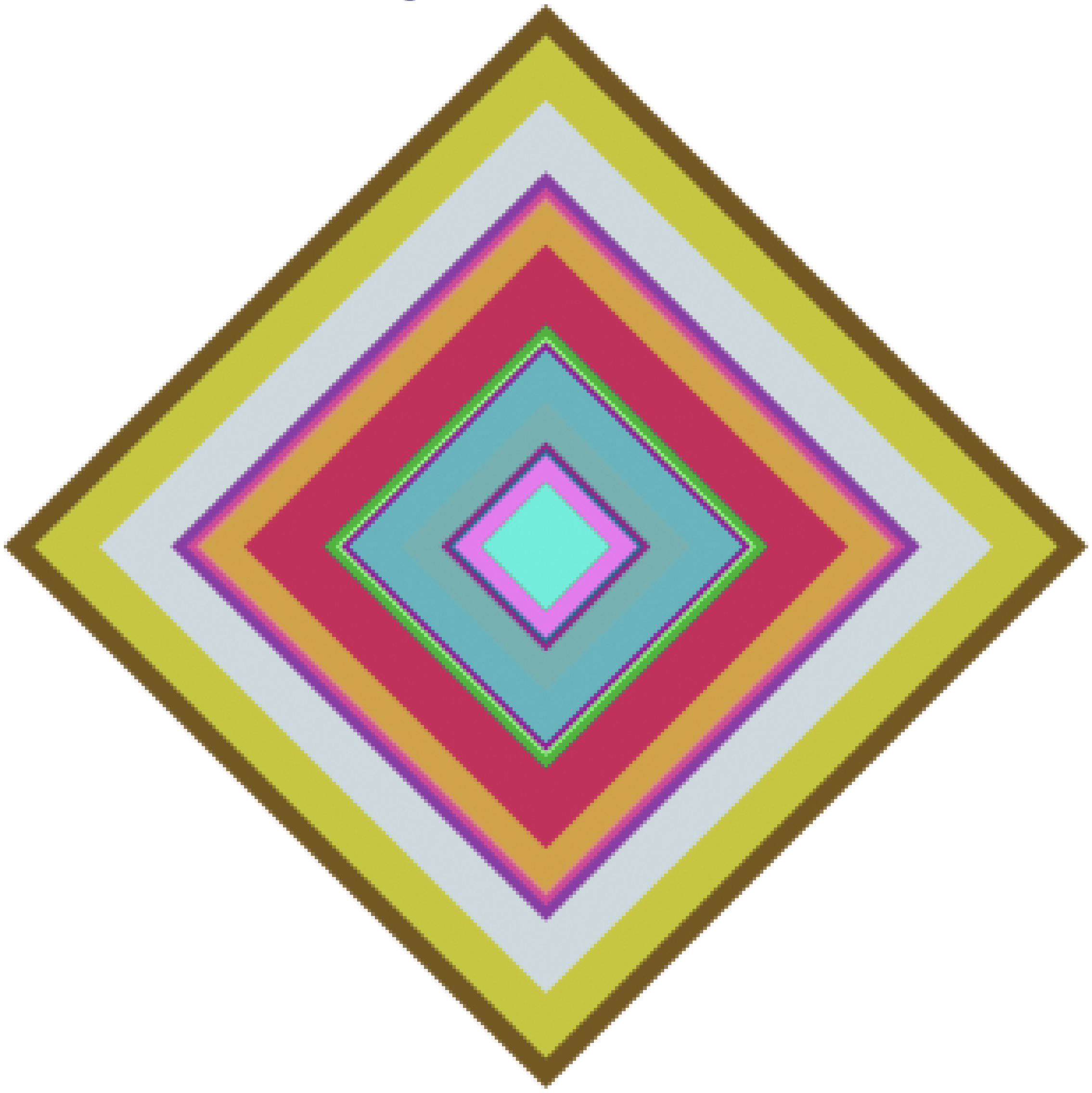


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 5)
```

```
>
```



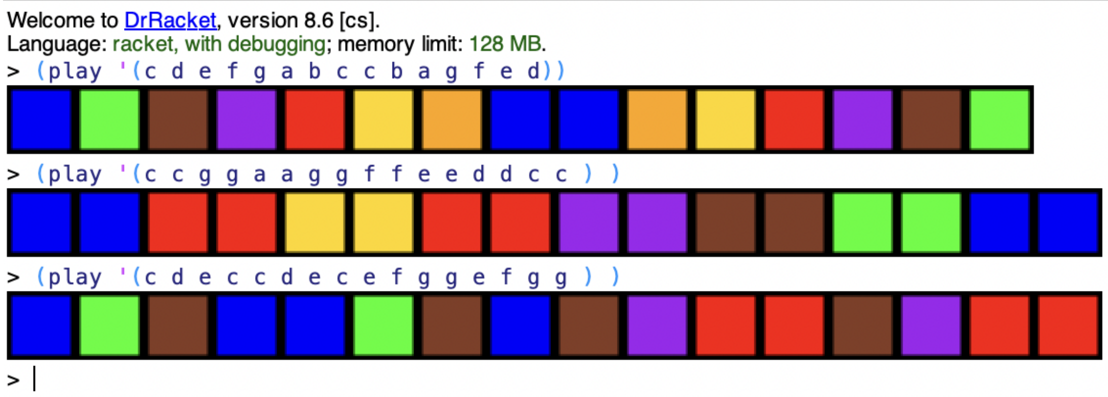
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 20)
```



```
> 
```

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

 ( define (generate-list unit line )
     (cond
       ((zero? unit)
        (list)
        )
      (else
       (append(list(line)) (generate-list(- unit 1 ) line))
      ) ) )

( define (diamond )
   (rotate 45(square(+ 20(random 200 )) "solid" (random-color) ))
)
( define ( sort-diamonds loc )
( sort loc #:key image-width < )
)
( define( diamond-design number)
    (define set (generate-list number diamond ) )
   (foldr overlay empty-image(sort-diamonds set) )
)
( define ( random-color )
( color ( rgb-value ) ( rgb-value ) ( rgb-value ) )
)

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

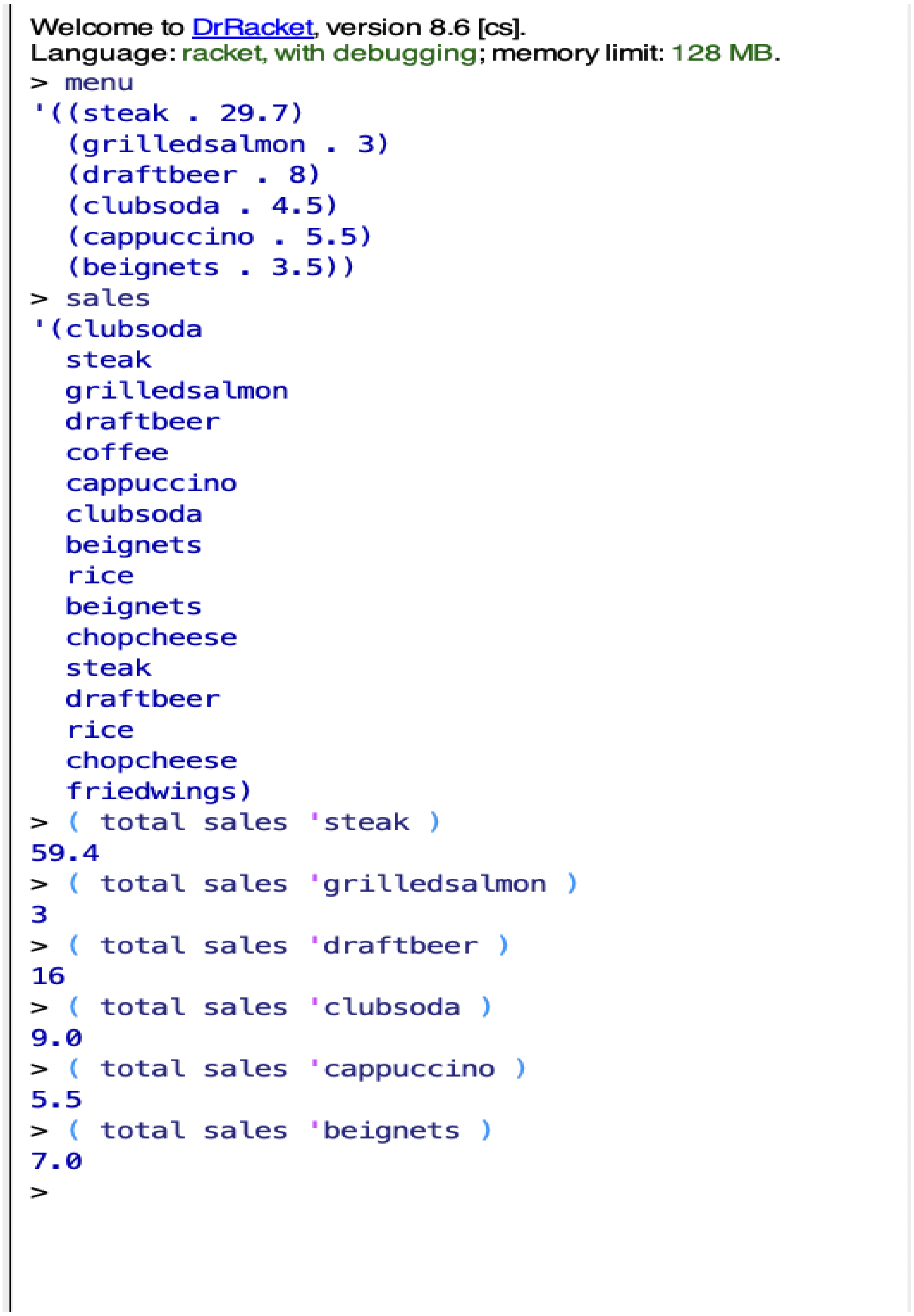
Task 8 - Chromesthetic renderings



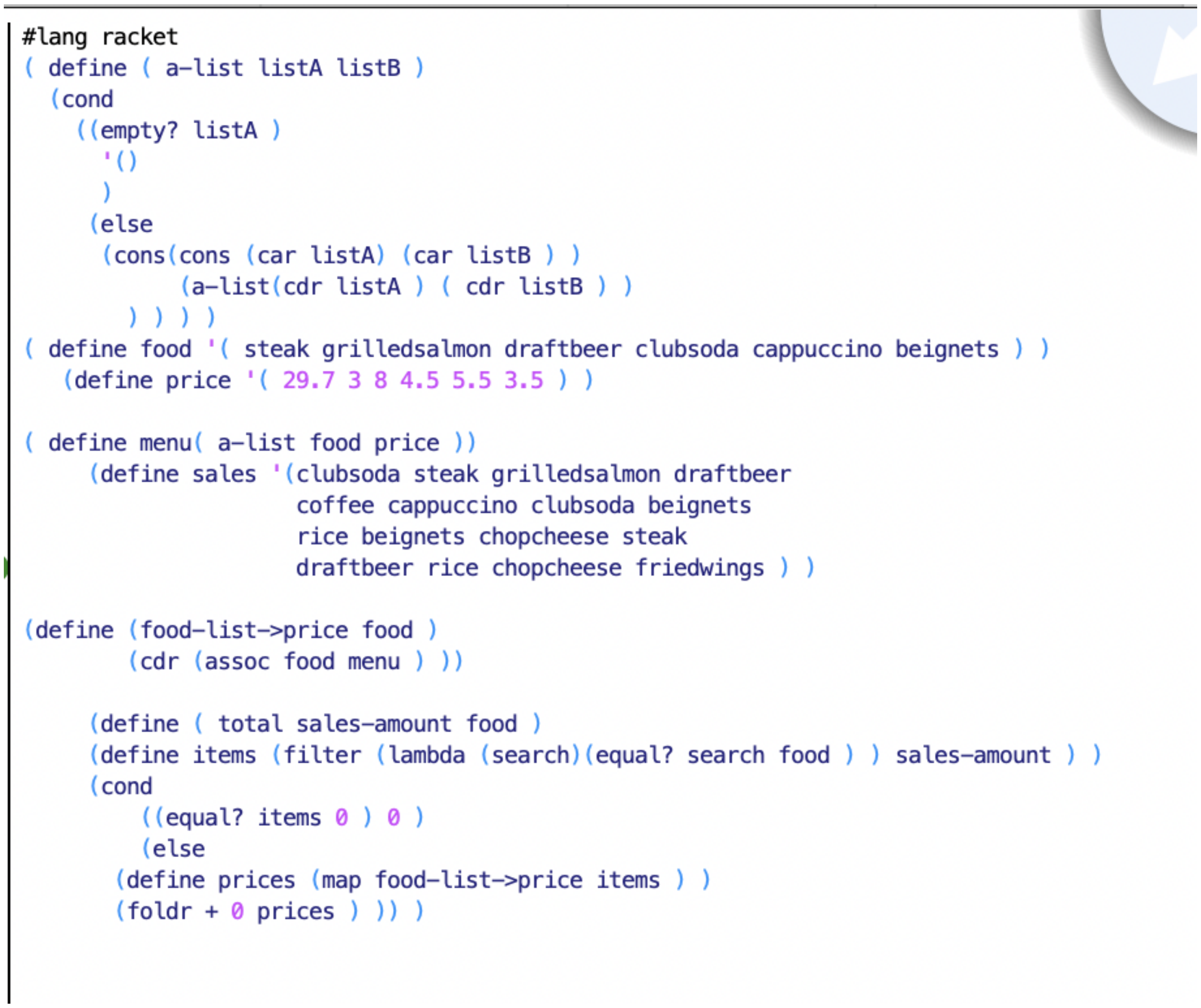
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (play '(c d e f g a b c c b a g f e d))
> (play '(c c g g a a g g f f e e d d c c ) )
> (play '(c d e c c d e c e f g g e f g g ) )
> 
```

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

( define ( a-list listA listB )
  (cond
    ((empty? listA )
      '()
      )
     (else
      (cons(cons (car listA) (car listB ) )
            (a-list(cdr listA ) ( cdr listB ) )
        ) ) ) )

( 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 c)
  (cond
    ((= (length c ) 0) empty-image )
    (else
     (beside ( color->box (pc->color (car c)))(play (cdr c))))))
```

Task 9 - Diner

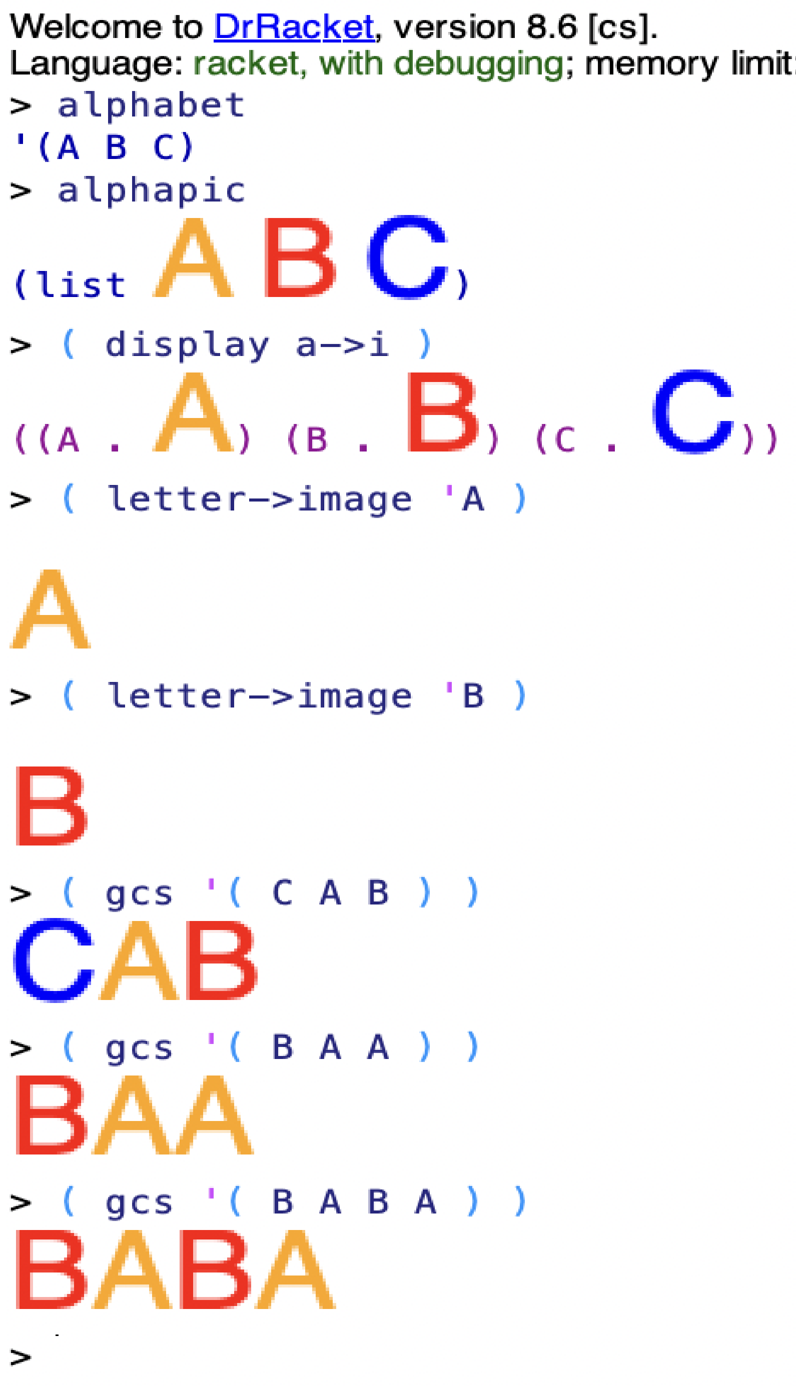


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> menu
'((steak . 29.7)
  (grilledsalmon . 3)
  (draftbeer . 8)
  (clubsoda . 4.5)
  (cappuccino . 5.5)
  (beignets . 3.5))
> sales
'(clubsoda
  steak
  grilledsalmon
  draftbeer
  coffee
  cappuccino
  clubsoda
  beignets
  rice
  beignets
  chopcheese
  steak
  draftbeer
  rice
  chopcheese
  friedwings)
> ( total sales 'steak )
59.4
> ( total sales 'grilledsalmon )
3
> ( total sales 'draftbeer )
16
> ( total sales 'clubsoda )
9.0
> ( total sales 'cappuccino )
5.5
> ( total sales 'beignets )
7.0
>
```

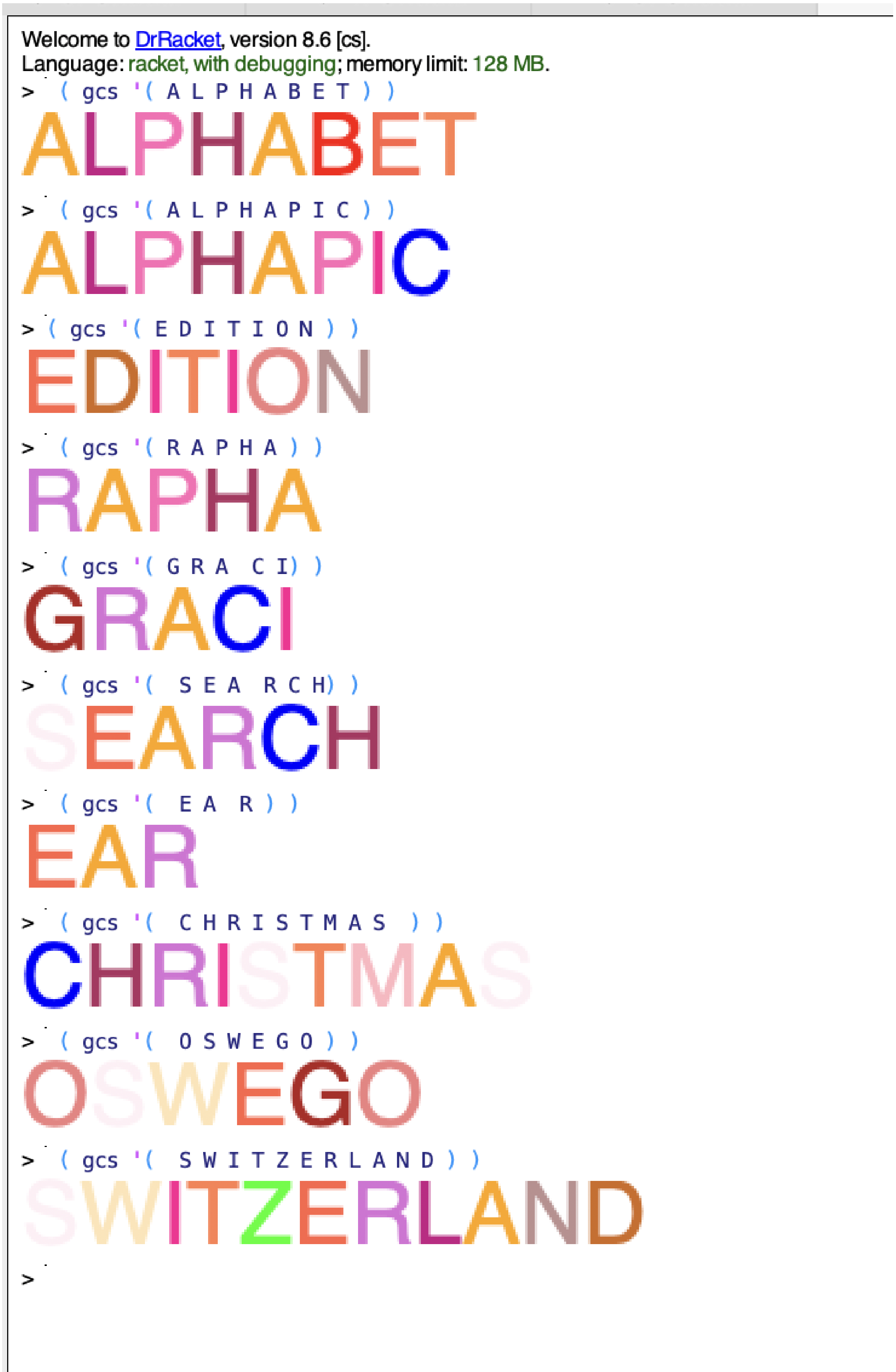


```
#lang racket
( define ( a-list listA listB )
  (cond
    ((empty? listA )
      '()
      )
     (else
      (cons(cons (car listA) (car listB ) )
            (a-list(cdr listA ) ( cdr listB ) )
        ) ) ) )
( define food '( steak grilledsalmon draftbeer clubsoda cappuccino beignets ) )
   (define price '( 29.7 3 8 4.5 5.5 3.5 ) )

( define menu( a-list food price ))
     (define sales '(clubsoda steak grilledsalmon draftbeer
                     coffee cappuccino clubsoda beignets
                     rice beignets chopcheese steak
                     draftbeer rice chopcheese friedwings ) )

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

     (define ( total sales-amount food )
     (define items (filter (lambda (search)(equal? search food ) ) sales-amount ) )
     (cond
         ((equal? items 0 ) 0 )
         (else
       (define prices (map food-list->price items ) )
       (foldr + 0 prices ) )) )
```

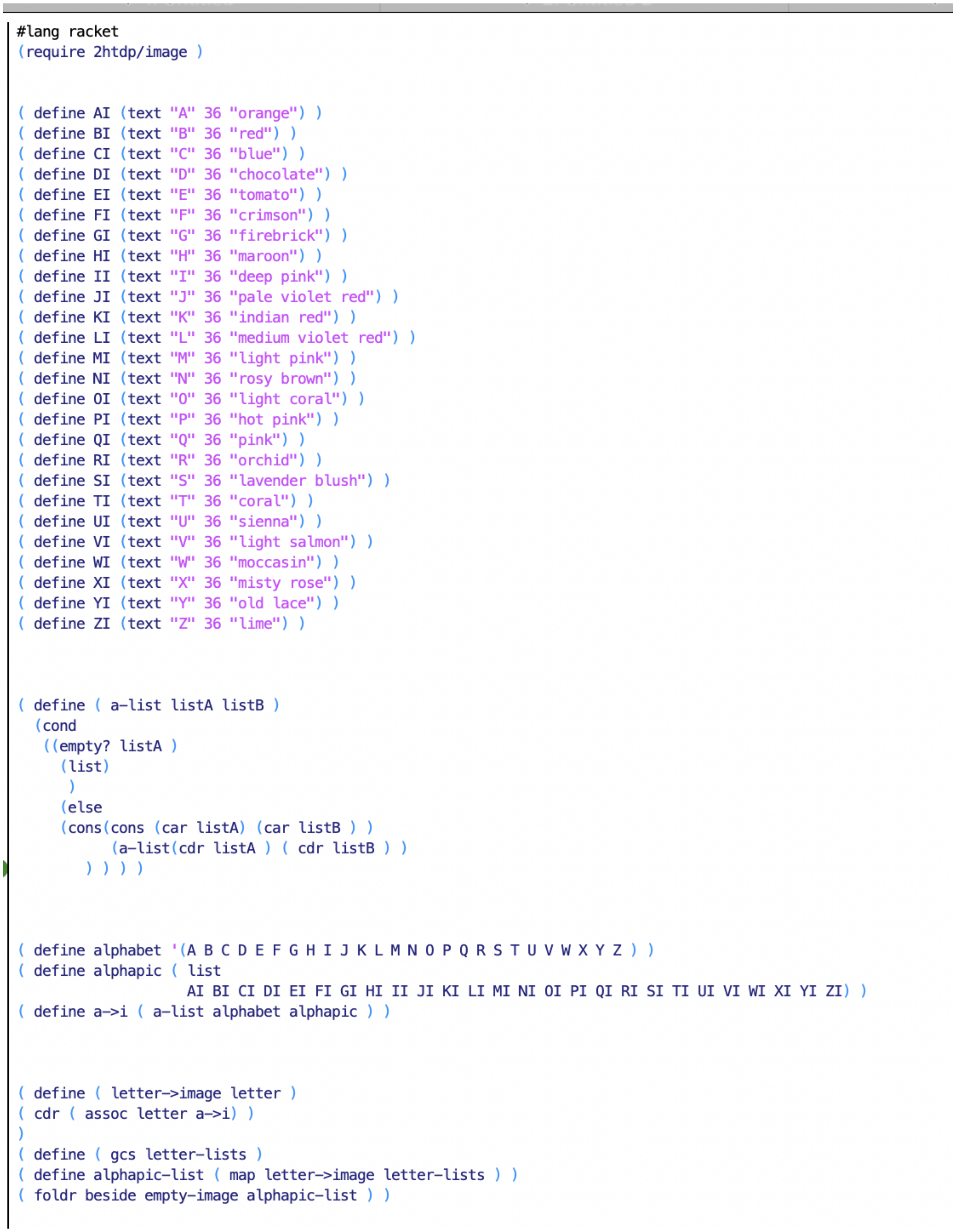
Task 10 - Grapheme Color Synesthesia



```
Welcome to DrRacket, version 8.6 [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
>
```



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( gcs '( A L P H A B E T ) )
ALPHABET
> ( gcs '( A L P H A P I C ) )
ALPHAPIC
> ( gcs '( E D I T I O N ) )
EDITION
> ( gcs '( R A P H A ) )
RAPHA
> ( gcs '( G R A  C I) )
GRACI
> ( gcs '(  S E A  R C H) )
SEARCH
> ( gcs '(  E A  R ) )
EAR
> ( gcs '(  C H R I S T M A S  ) )
CHRISTMAS
> ( gcs '(  O S W E G O ) )
OSWEGO
> ( gcs '(  S W I T Z E R L A N D ) )
SWITZERLAND
>
```



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


( define AI (text "A" 36 "orange") )
( define BI (text "B" 36 "red") )
( define CI (text "C" 36 "blue") )
( define DI (text "D" 36 "chocolate") )
( define EI (text "E" 36 "tomato") )
( define FI (text "F" 36 "crimson") )
( define GI (text "G" 36 "firebrick") )
( define HI (text "H" 36 "maroon") )
( define II (text "I" 36 "deep pink") )
( define JI (text "J" 36 "pale violet red") )
( define KI (text "K" 36 "indian red") )
( define LI (text "L" 36 "medium violet red") )
( define MI (text "M" 36 "light pink") )
( define NI (text "N" 36 "rosy brown") )
( define OI (text "O" 36 "light coral") )
( define PI (text "P" 36 "hot pink") )
( define QI (text "Q" 36 "pink") )
( define RI (text "R" 36 "orchid") )
( define SI (text "S" 36 "lavender blush") )
( define TI (text "T" 36 "coral") )
( define UI (text "U" 36 "sienna") )
( define VI (text "V" 36 "light salmon") )
( define WI (text "W" 36 "moccasin") )
( define XI (text "X" 36 "misty rose") )
( define YI (text "Y" 36 "old lace") )
( define ZI (text "Z" 36 "lime") )



( define ( a-list listA listB )
  (cond
   ((empty? listA )
     (list)
      )
     (else
     (cons(cons (car listA) (car listB ) )
           (a-list(cdr listA ) ( cdr listB ) )
        ) ) ) )



( 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 letter-lists )
( define alphapic-list ( map letter->image letter-lists ) )
( foldr beside empty-image alphapic-list ) )
```