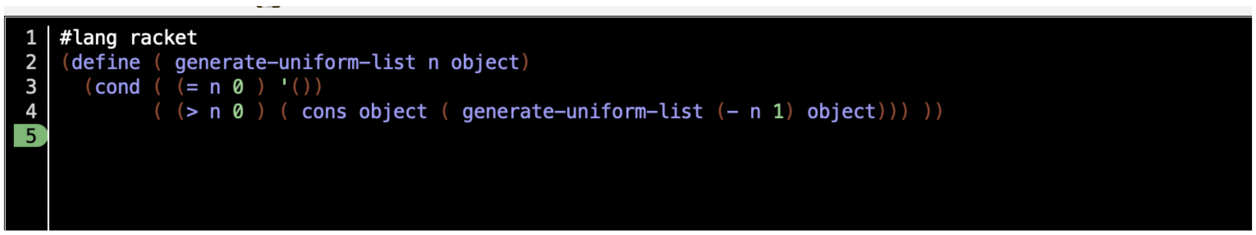
Racket Programming Assignment #4: RLP and HoFs

Abstract:

Working within the DrRacket PDE, please do each of the following tasks. The first five tasks pertain to straight- forward recursive list processing. The remaining tasks pertain to list processing with higher order functions. The final task, which you might like to work on from the start, is the document compilation/posting task, which as usual amounts to crafting a single structured document that reflects your work on each programming task, and saving it as a pdf file.

Generate Uniform List:

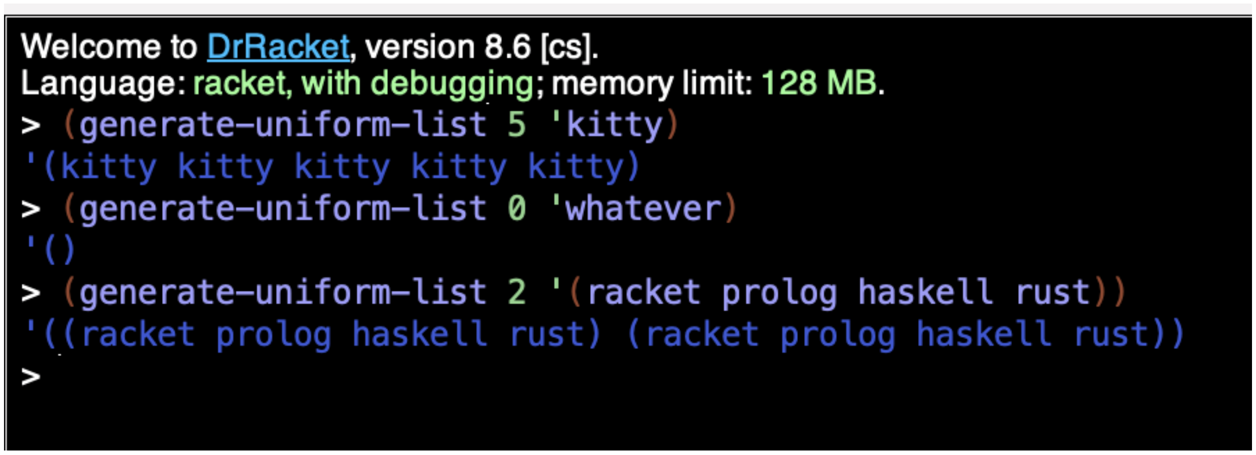
Uniform List Demo



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

Generate Uniform List:

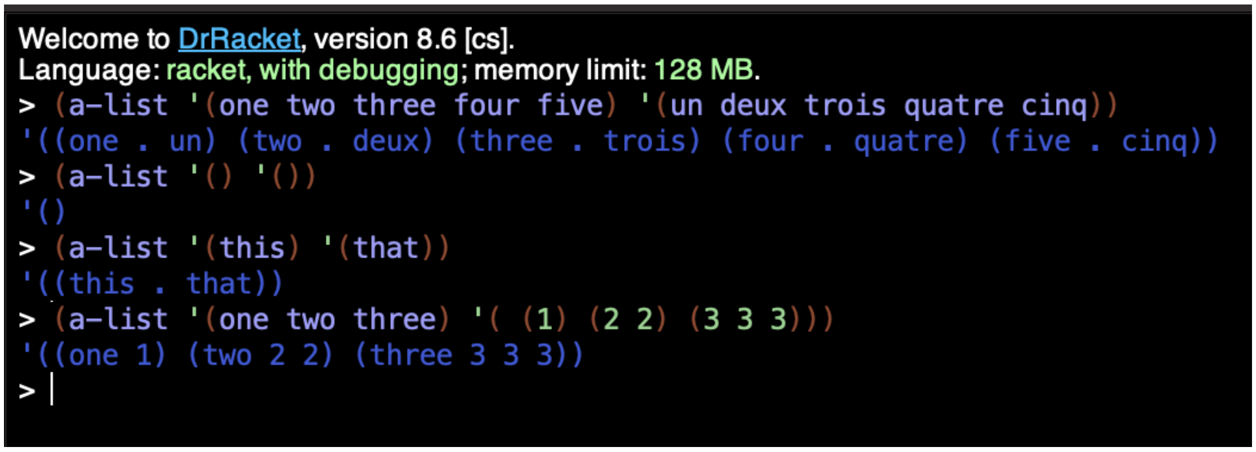
Uniform List Code



```
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 0 'whatever)
'()
> (generate-uniform-list 2 '(racket prolog haskell rust))
'((racket prolog haskell rust) (racket prolog haskell rust))
>
```

Association List Generator:

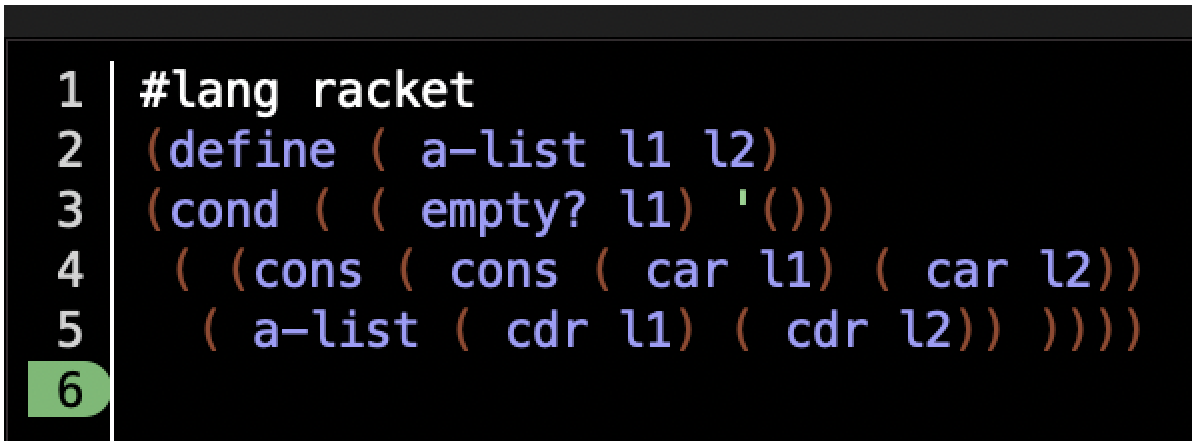
Association List Generator Demo



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

Association List Generator:

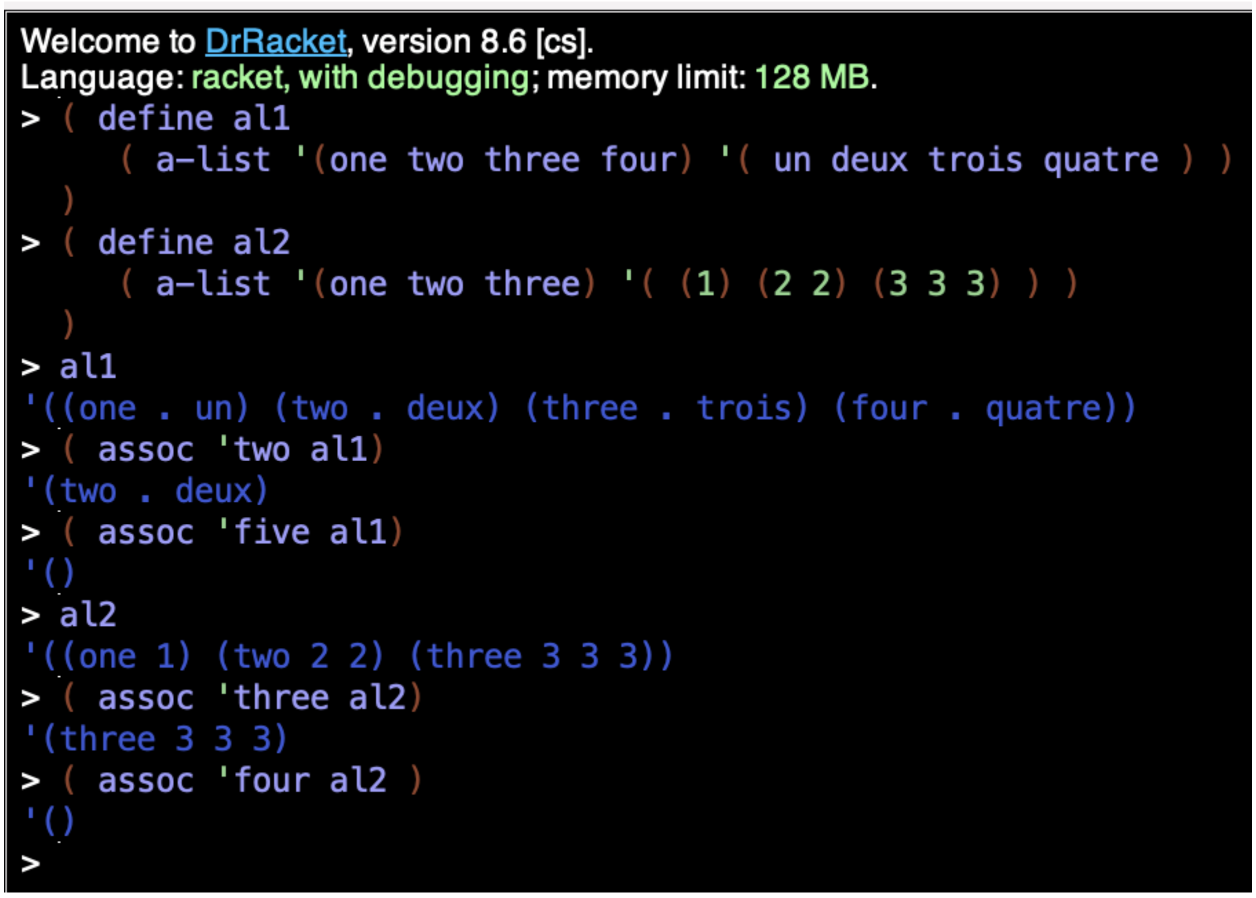
Association List Generator Code



```
#lang racket
(define ( a-list l1 l2)
(cond ( ( empty? l1) '())
 ( (cons ( cons ( car l1) ( car l2))
  ( a-list ( cdr l1) ( cdr l2)) ))))
```

Assoc:

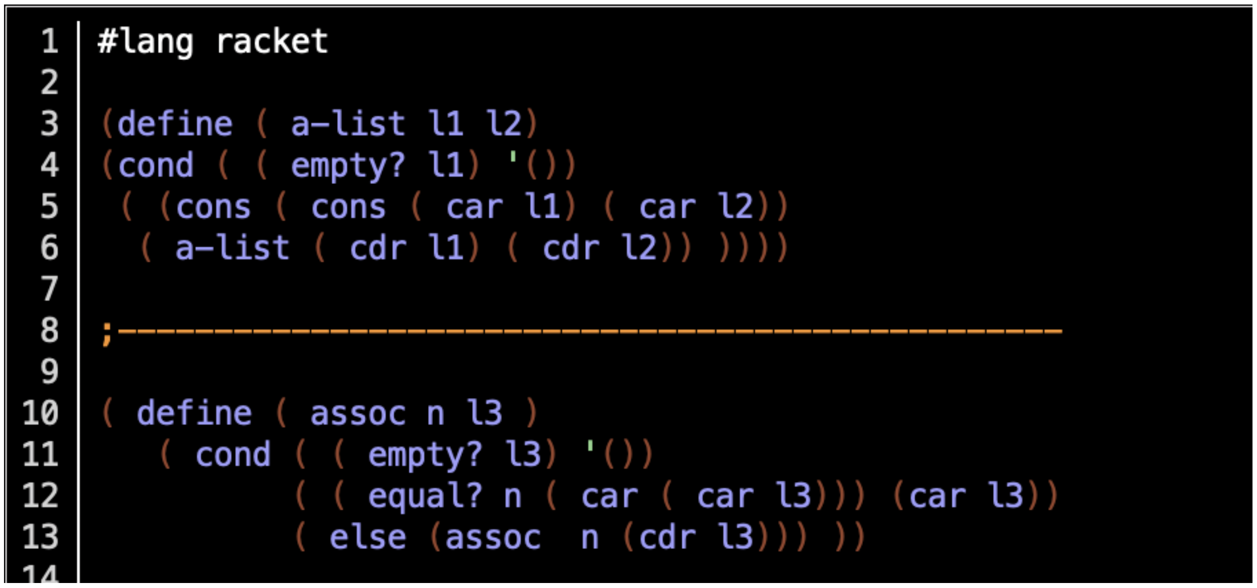
Assoc Demo



```
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 )
'()
>
```

Assoc:

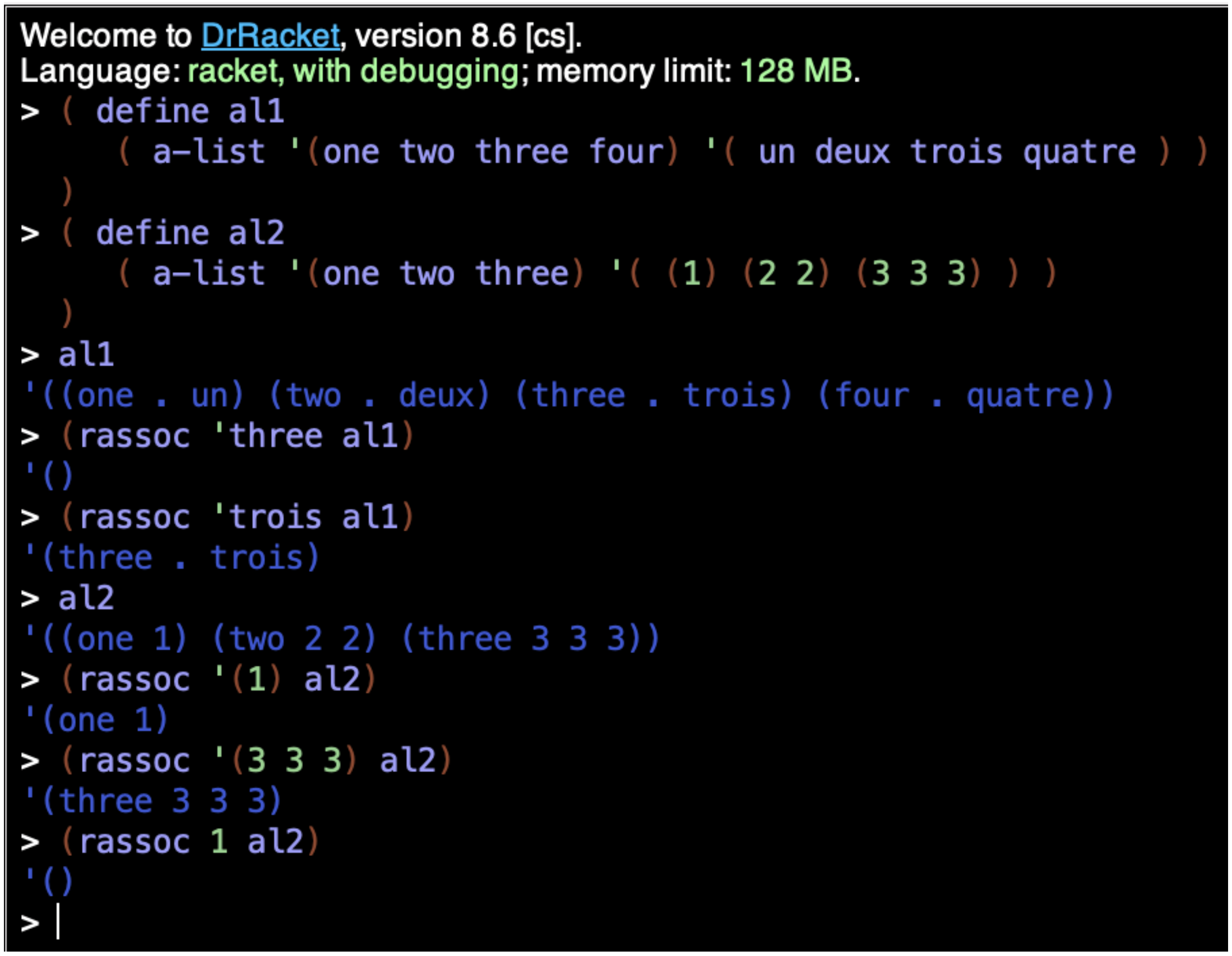
Assoc Code



```
#lang racket

(define ( a-list l1 l2)
(cond ( ( empty? l1) '())
 ( (cons ( cons ( car l1) ( car l2))
  ( a-list ( cdr l1) ( cdr l2)) ))))

;-------------------------------------------------

( define ( assoc n l3 )
   ( cond ( ( empty? l3) '())
          ( ( equal? n ( car ( car l3))) (car l3))
          ( else (assoc  n (cdr l3))) ))
```

Rassoc:

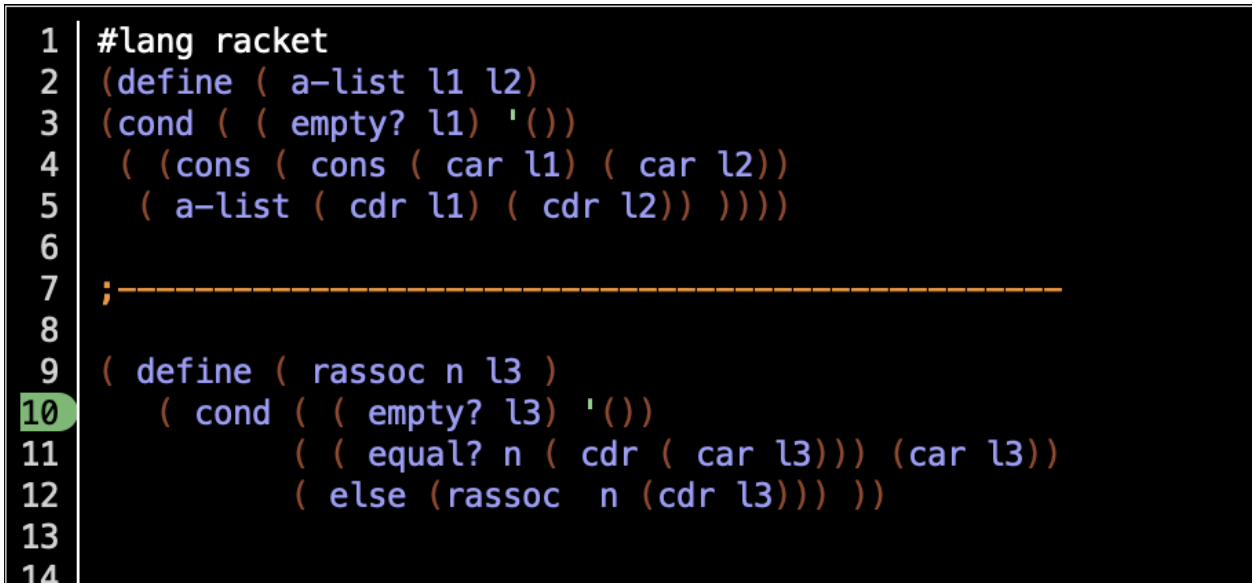
Rassoc Demo



```
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)
'()
> |
```

Rassoc:

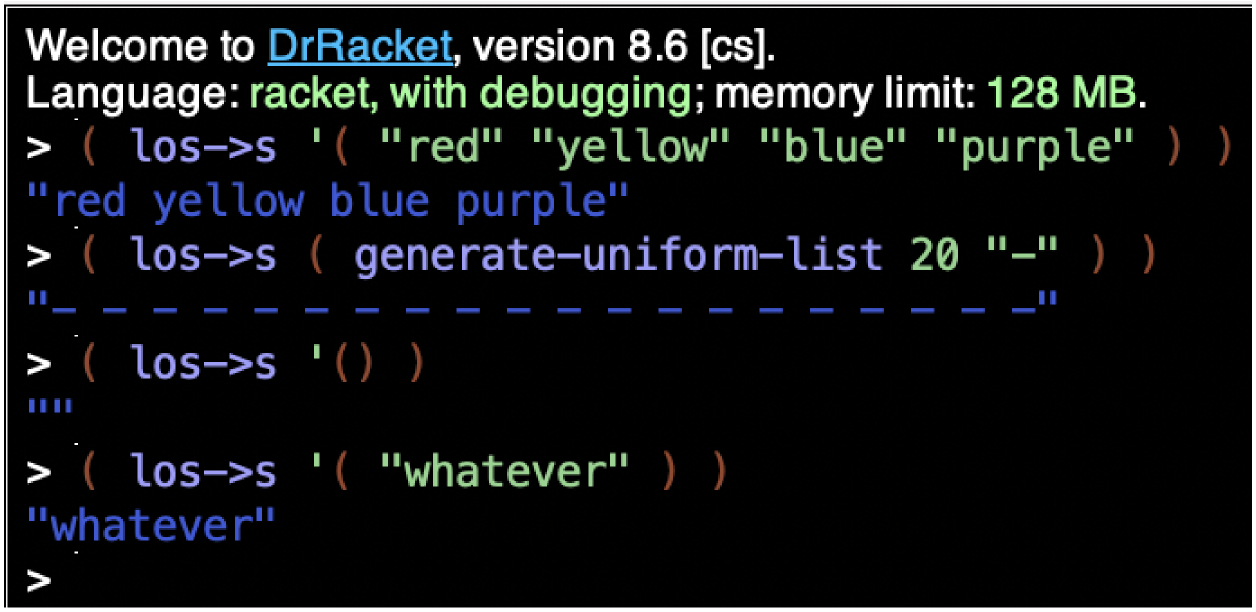
Rassoc Code



```
#lang racket
(define ( a-list l1 l2)
(cond ( ( empty? l1) '())
 ( (cons ( cons ( car l1) ( car l2))
  ( a-list ( cdr l1) ( cdr l2)) ))))

;-------------------------------------------------

( define ( rassoc n l3 )
   ( cond ( ( empty? l3) '())
          ( ( equal? n ( cdr ( car l3))) (car l3))
          ( else (rassoc  n (cdr l3))) ))


```

Los->:

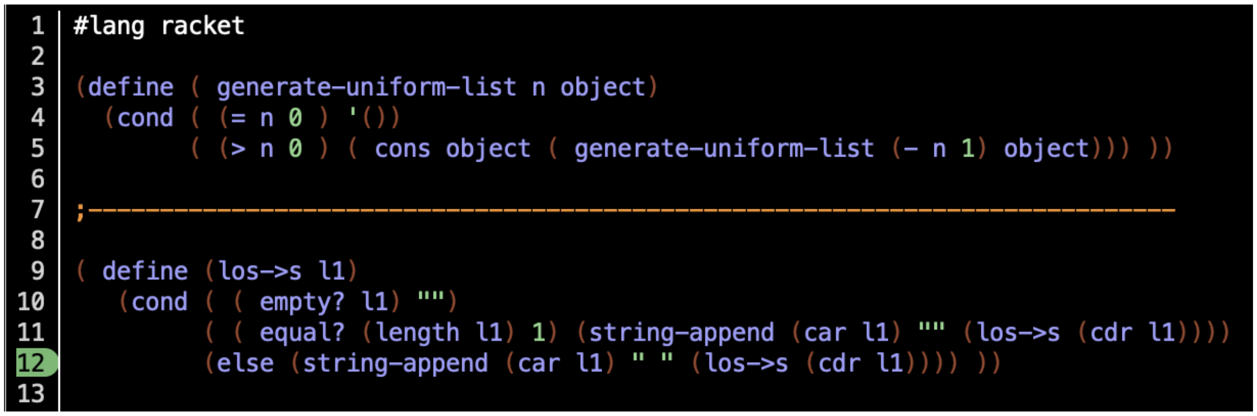
Los-> Demo



```
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"
>
```

Los->:

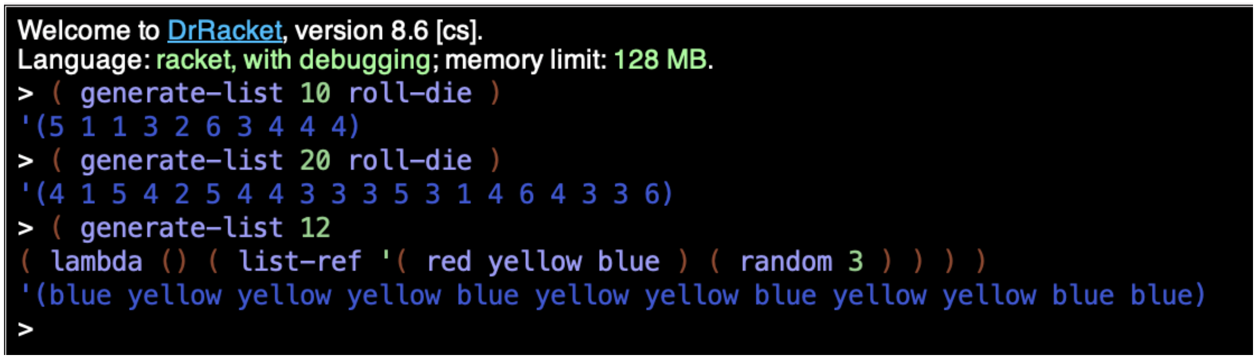
Los-> Code



```
#lang racket

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

;----------------------------------------------------------------------------

( define (los->s l1)
   (cond ( ( empty? l1) "")
         ( ( equal? (length l1) 1) (string-append (car l1) "" (los->s (cdr l1))))
         (else (string-append (car l1) " " (los->s (cdr l1)))) ))

```

Generate List:

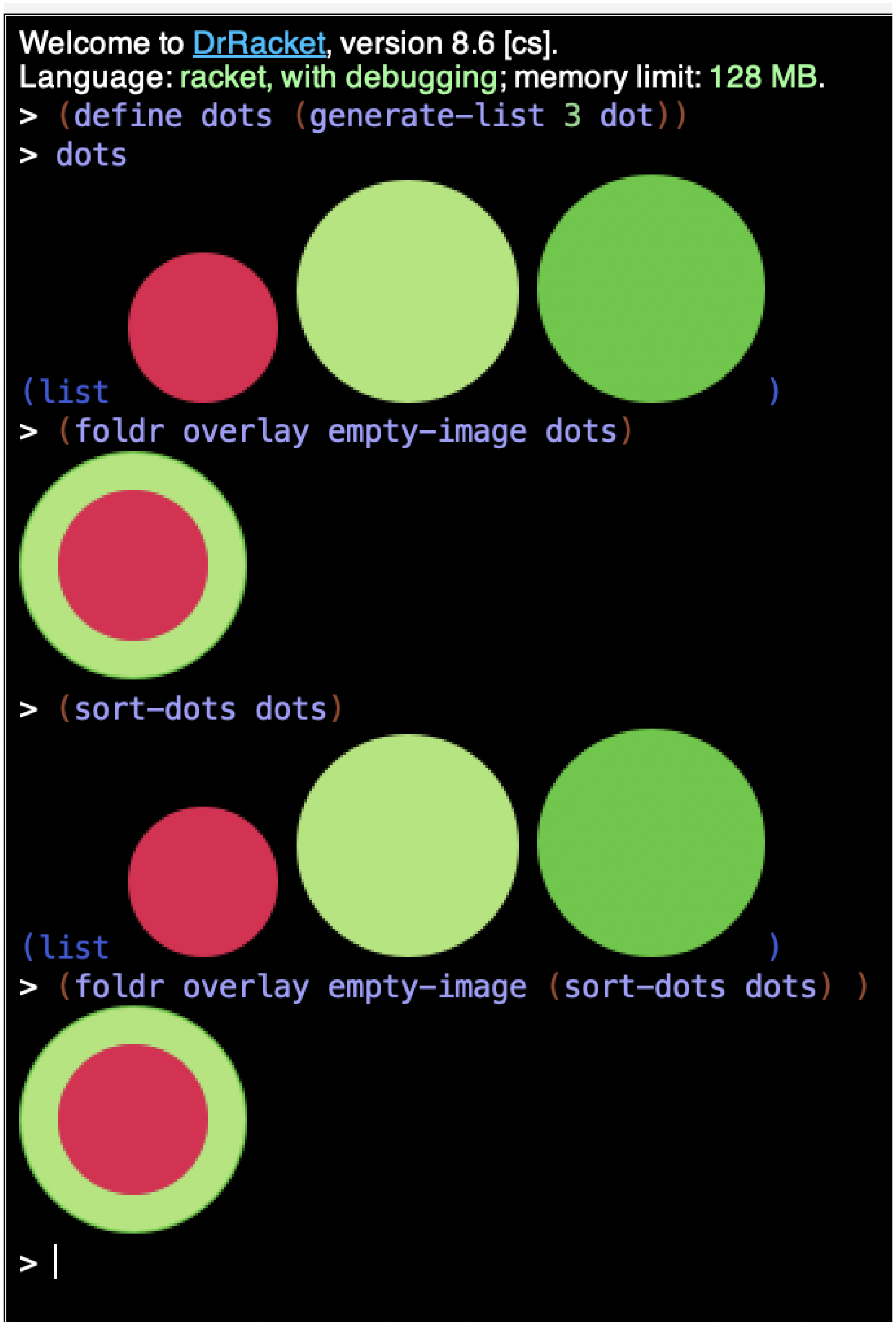
Generate List Demo 1



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

Generate List:

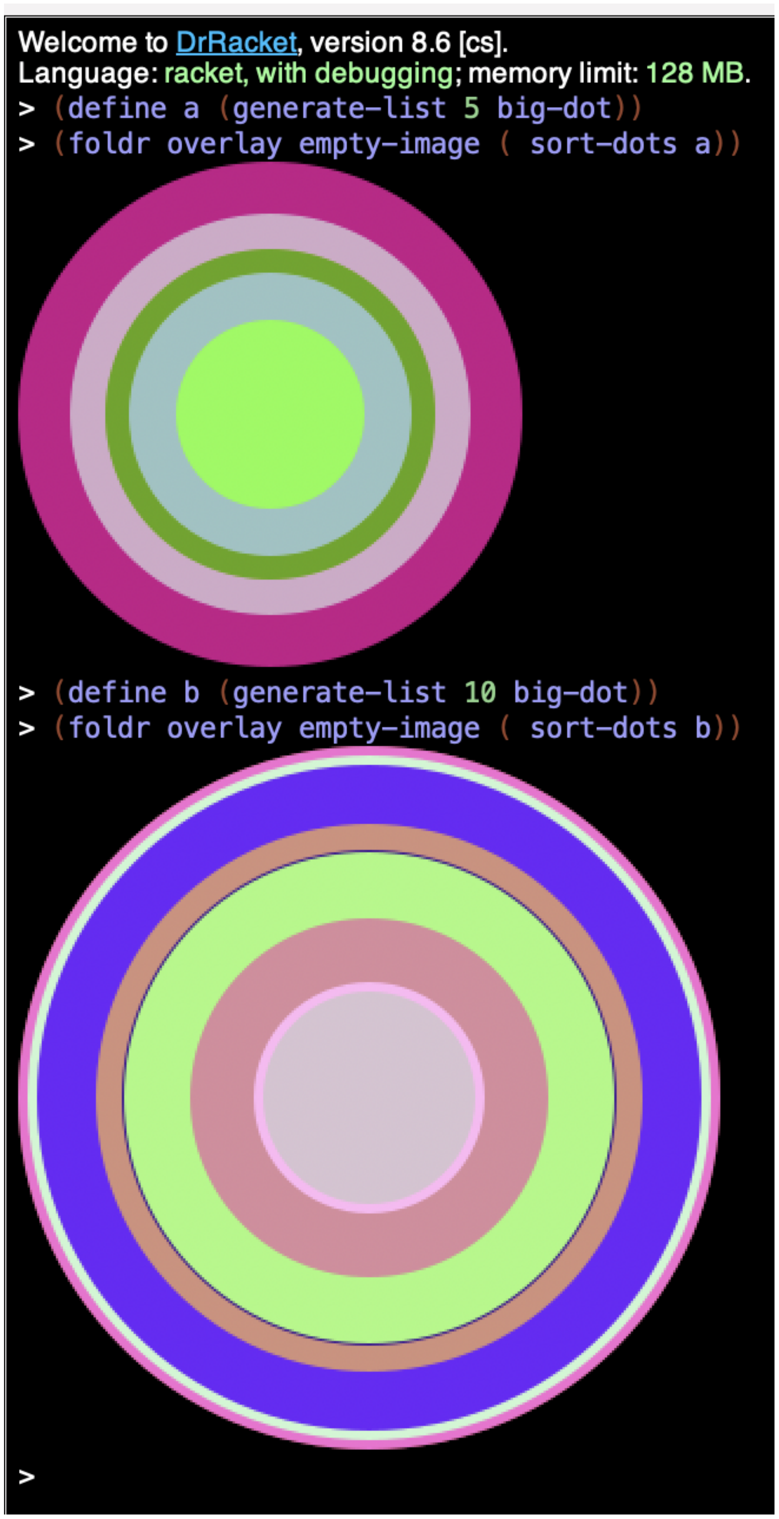
Generate List Demo 2



```
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) )

> |
```

Generate List:

Generate List Demo 3



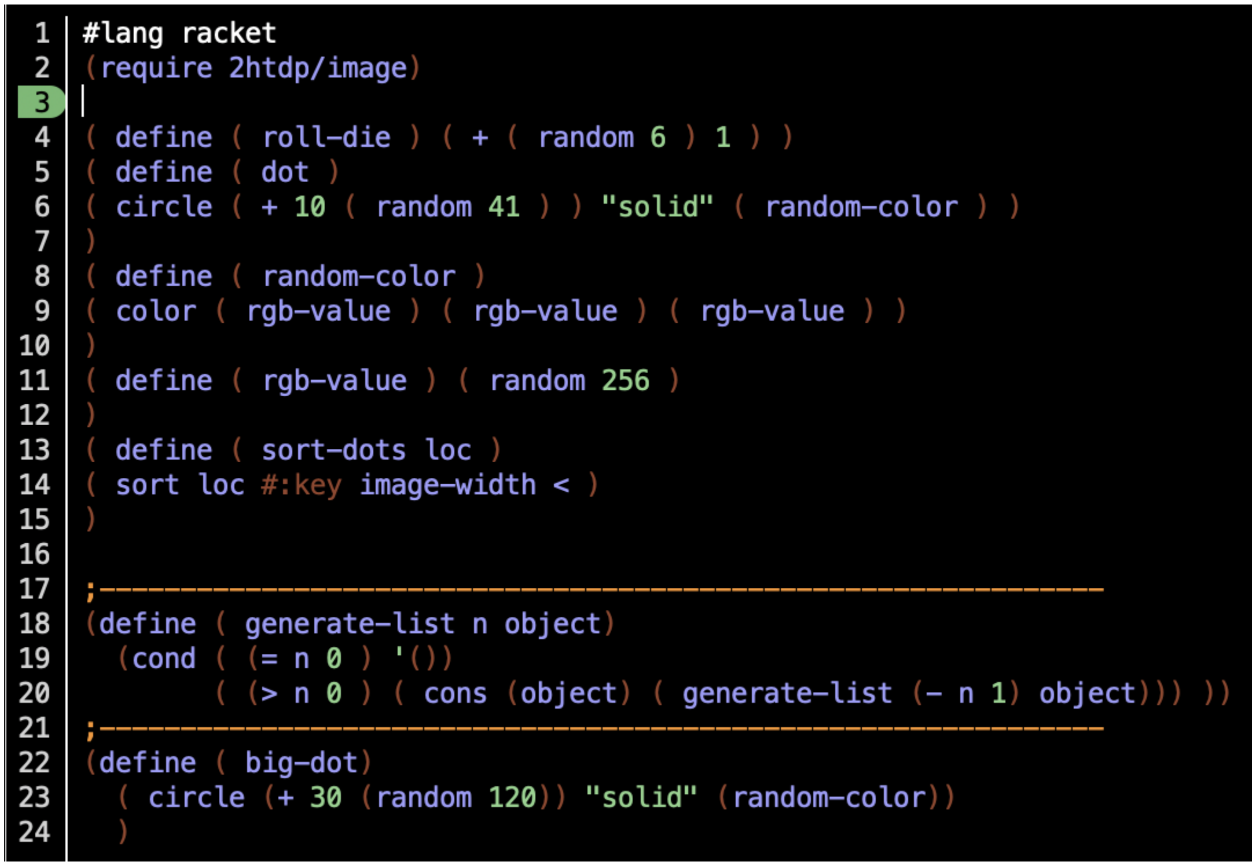
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define a (generate-list 5 big-dot))
> (foldr overlay empty-image ( sort-dots a))
```

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

```
>
```

Generate List:

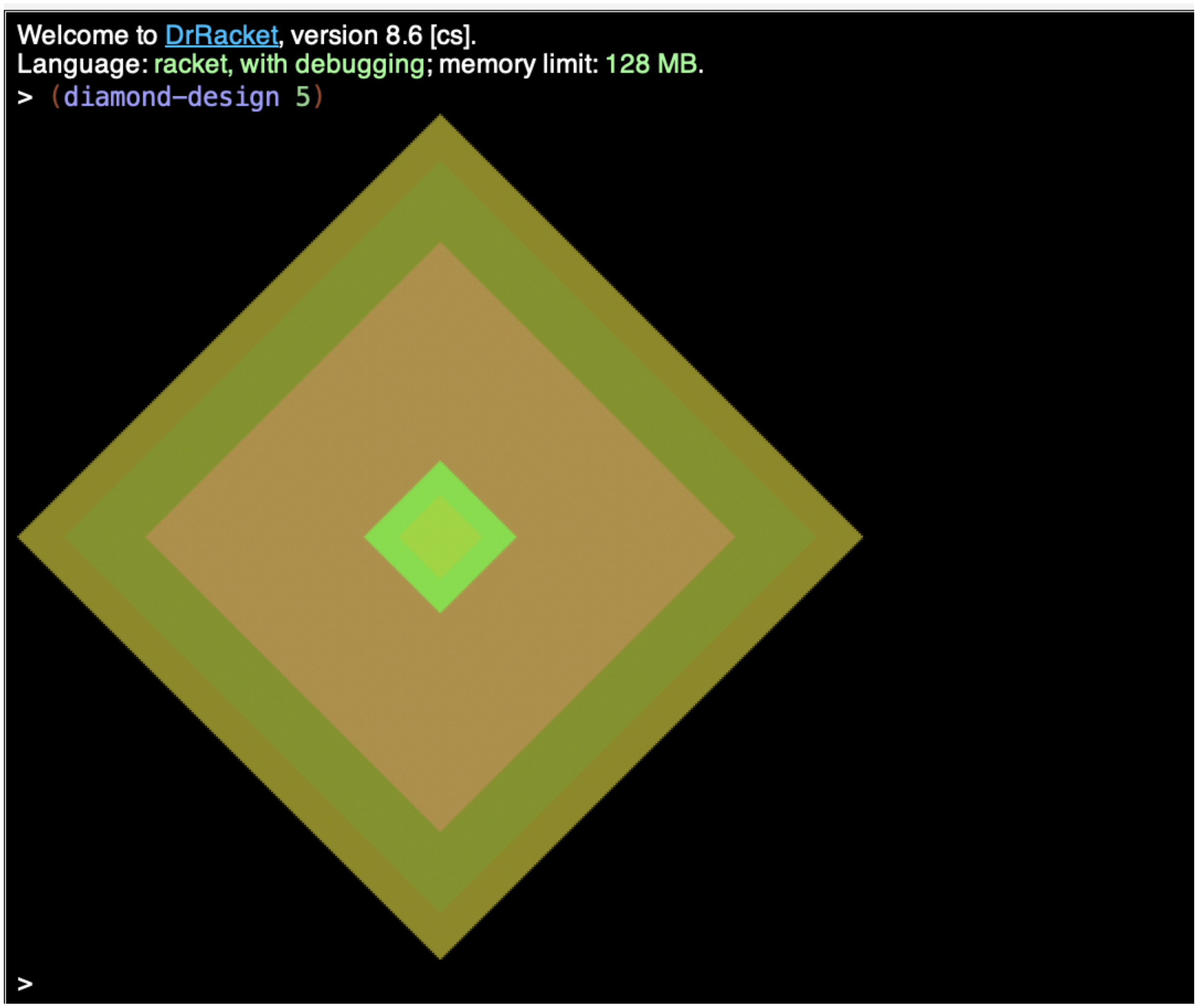
Generate List Code



```racket
#lang racket
(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 ( sort-dots loc )
( sort loc #:key image-width < )
)

;---------------------------------------------------------------
(define ( generate-list n object)
  (cond ( (= n 0 ) '())
        ( (> n 0 ) ( cons (object) ( generate-list (- n 1) object))) ))
;---------------------------------------------------------------
(define ( big-dot)
  ( circle (+ 30 (random 120)) "solid" (random-color))
  )
```

The Diamond:

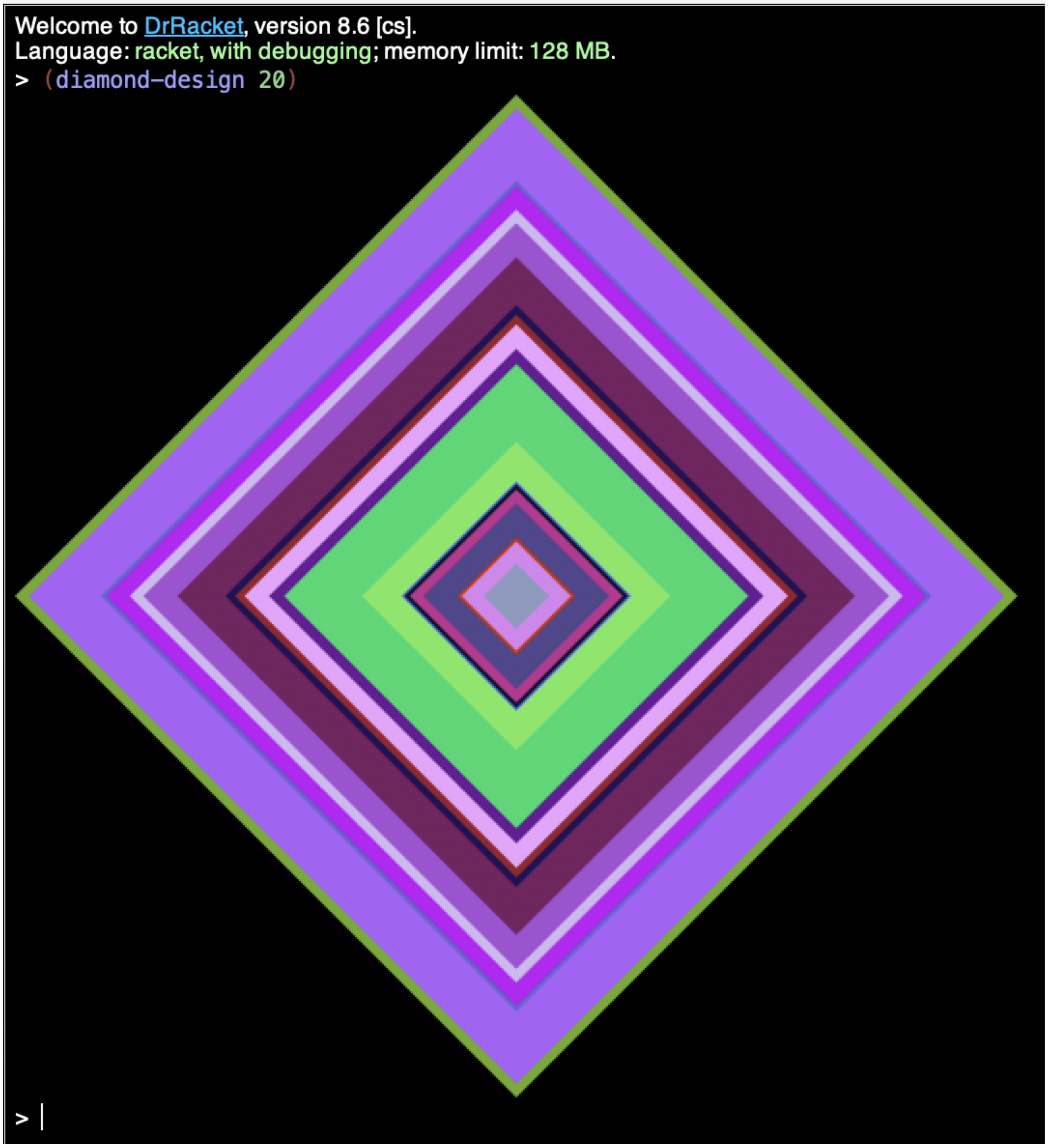
The Diamond Demo 1



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

The Diamond:

The Diamond Demo 2

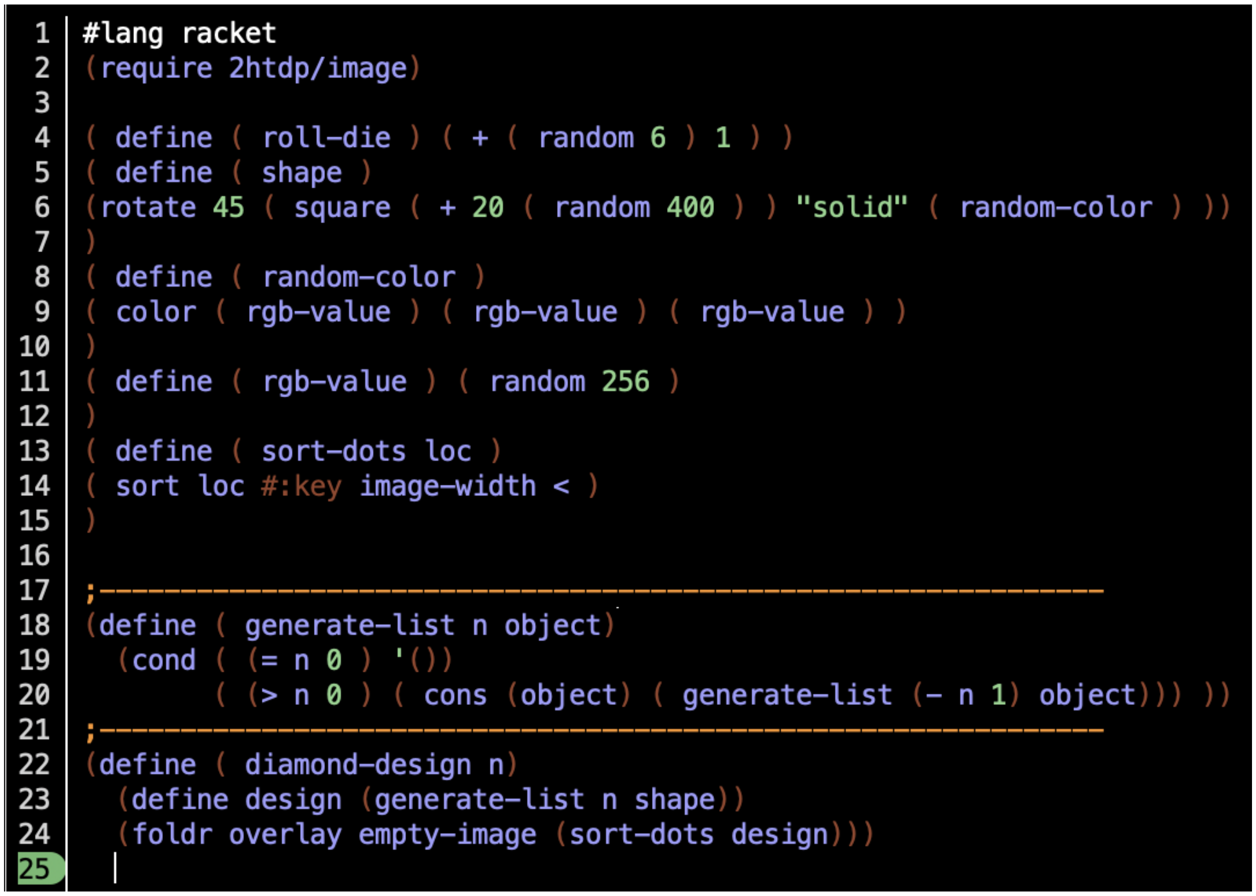


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (diamond-design 20)
```

```
>
```

The Diamond:

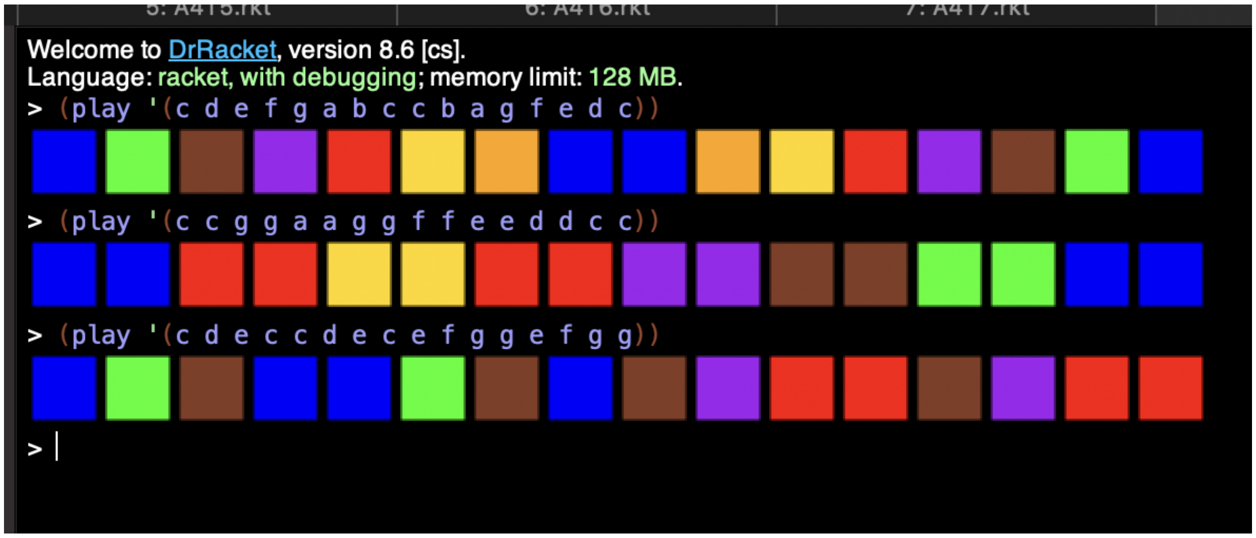
The Diamond Code



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

( define ( roll-die ) ( + ( random 6 ) 1 ) )
( define ( shape )
(rotate 45 ( square ( + 20 ( random 400 ) ) "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 n object)
  (cond ( (= n 0 ) '())
        ( (> n 0 ) ( cons (object) ( generate-list (- n 1) object))) ))
;-----------------------------------------------------------------
(define ( diamond-design n)
  (define design (generate-list n shape))
  (foldr overlay empty-image (sort-dots design)))
```

Chromesthetic Renderings:

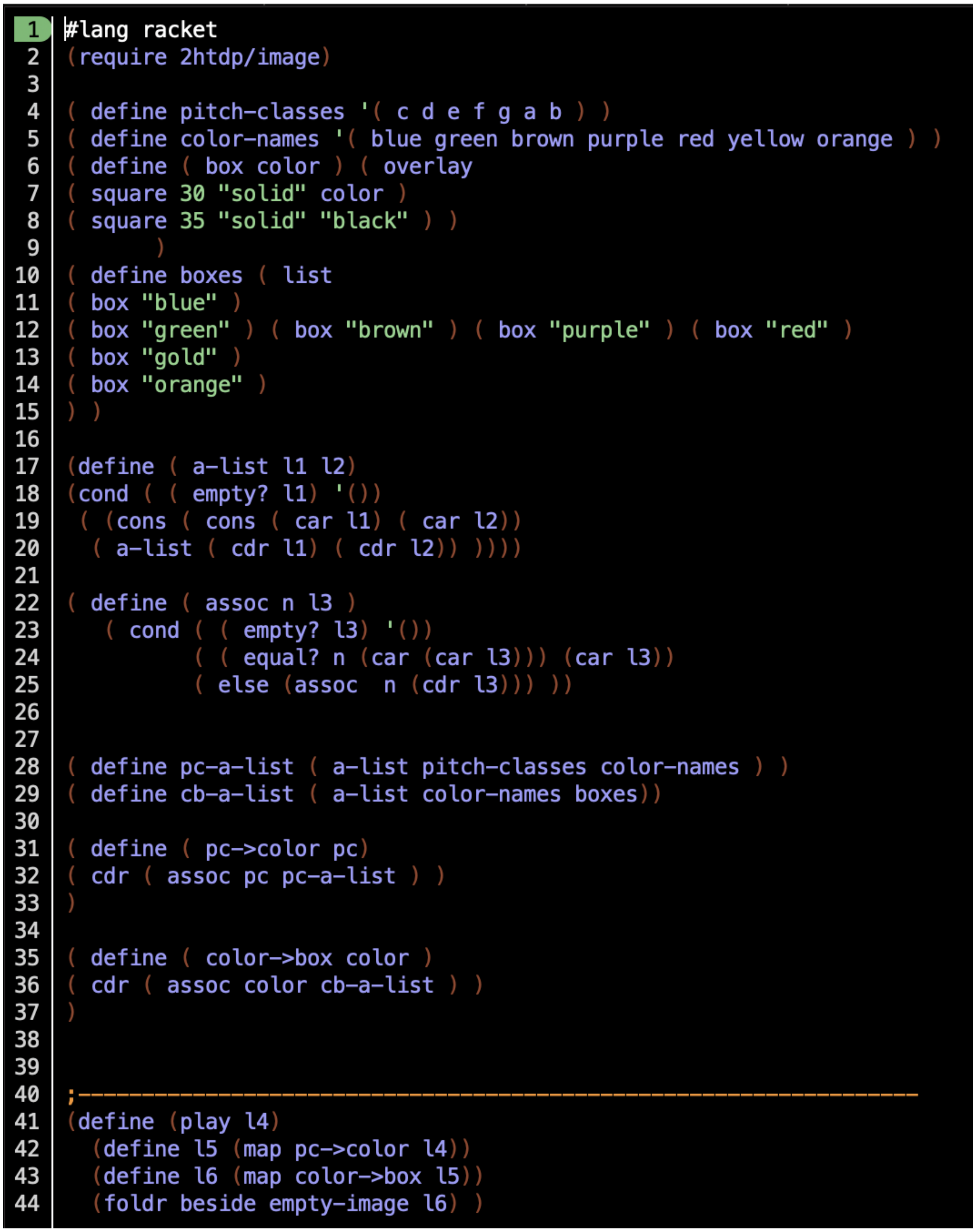
Chromesthetic Renderings Demo



```
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 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))
>
```

Chromesthetic Renderings:

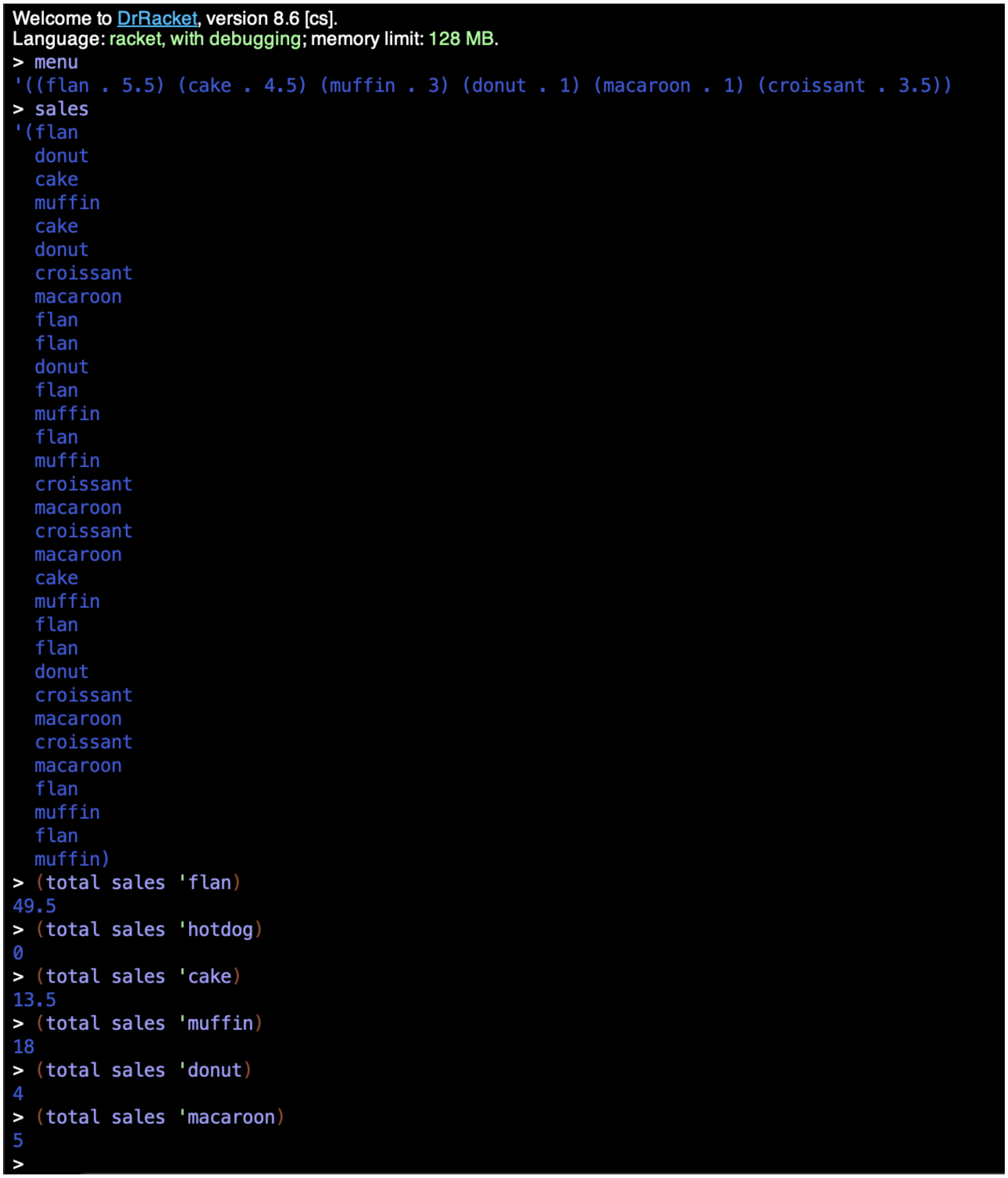
Chromesthetic Renderings Demo



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

( 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 ( a-list l1 l2)
(cond ( ( empty? l1) '())
 ( (cons ( cons ( car l1) ( car l2))
  ( a-list ( cdr l1) ( cdr l2)) ))))

( define ( assoc n l3 )
   ( cond ( ( empty? l3) '())
          ( ( equal? n (car (car l3))) (car l3))
          ( else (assoc  n (cdr l3))) ))


( 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 l4)
  (define l5 (map pc->color l4))
  (define l6 (map color->box l5))
  (foldr beside empty-image l6) )
```

Diner:

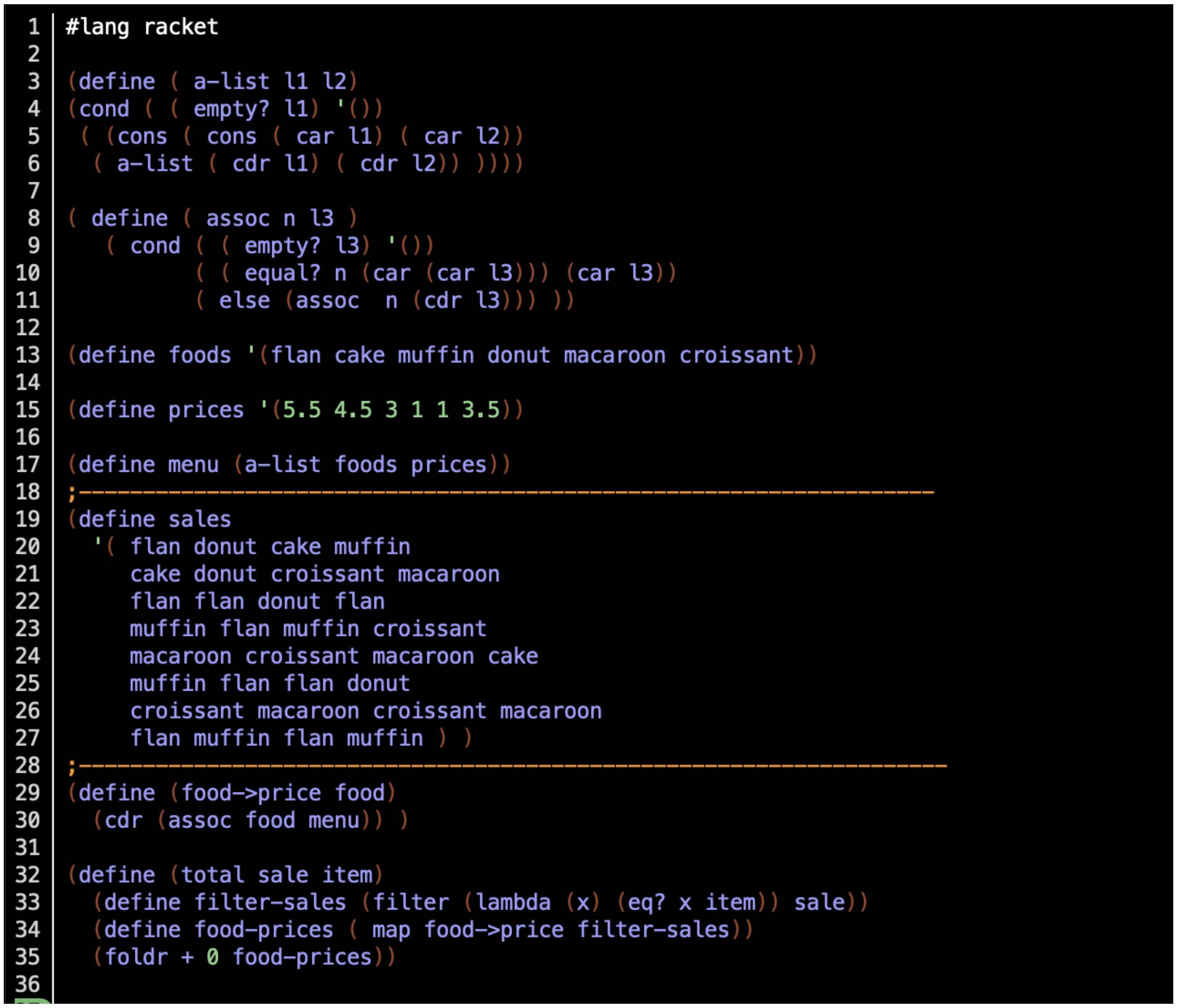
Diner Demo



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> menu
'((flan . 5.5) (cake . 4.5) (muffin . 3) (donut . 1) (macaroon . 1) (croissant . 3.5))
> sales
'(flan
  donut
  cake
  muffin
  cake
  donut
  croissant
  macaroon
  flan
  flan
  donut
  flan
  muffin
  flan
  muffin
  croissant
  macaroon
  croissant
  macaroon
  cake
  muffin
  flan
  flan
  donut
  croissant
  macaroon
  croissant
  macaroon
  flan
  muffin
  flan
  muffin)
> (total sales 'flan)
49.5
> (total sales 'hotdog)
0
> (total sales 'cake)
13.5
> (total sales 'muffin)
18
> (total sales 'donut)
4
> (total sales 'macaroon)
5
>
```

Diner:

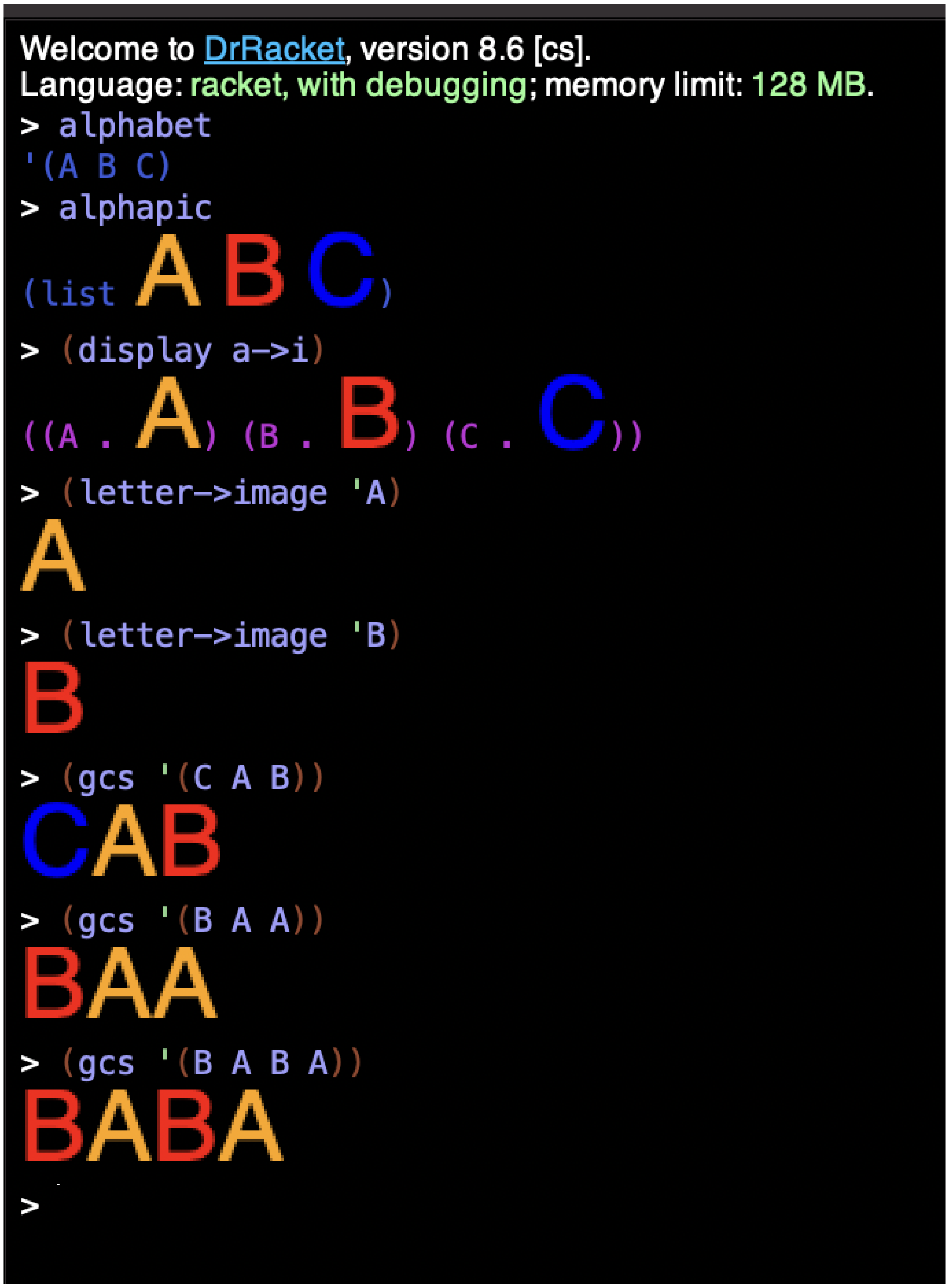
Diner Code



```
#lang racket

(define ( a-list l1 l2)
(cond ( ( empty? l1) '())
 ( (cons ( cons ( car l1) ( car l2))
  ( a-list ( cdr l1) ( cdr l2)) ))))

( define ( assoc n l3 )
   ( cond ( ( empty? l3) '())
          ( ( equal? n (car (car l3))) (car l3))
          ( else (assoc  n (cdr l3))) ))

(define foods '(flan cake muffin donut macaroon croissant))

(define prices '(5.5 4.5 3 1 1 3.5))

(define menu (a-list foods prices))
;-------------------------------------------------------------------
(define sales
  '( flan donut cake muffin
      cake donut croissant macaroon
      flan flan donut flan
      muffin flan muffin croissant
      macaroon croissant macaroon cake
      muffin flan flan donut
      croissant macaroon croissant macaroon
      flan muffin flan muffin ) )
;--------------------------------------------------------------------
(define (food->price food)
  (cdr (assoc food menu)) )

(define (total sale item)
  (define filter-sales (filter (lambda (x) (eq? x item)) sale))
  (define food-prices ( map food->price filter-sales))
  (foldr + 0 food-prices))

```

Grapheme Color Synesthesia:

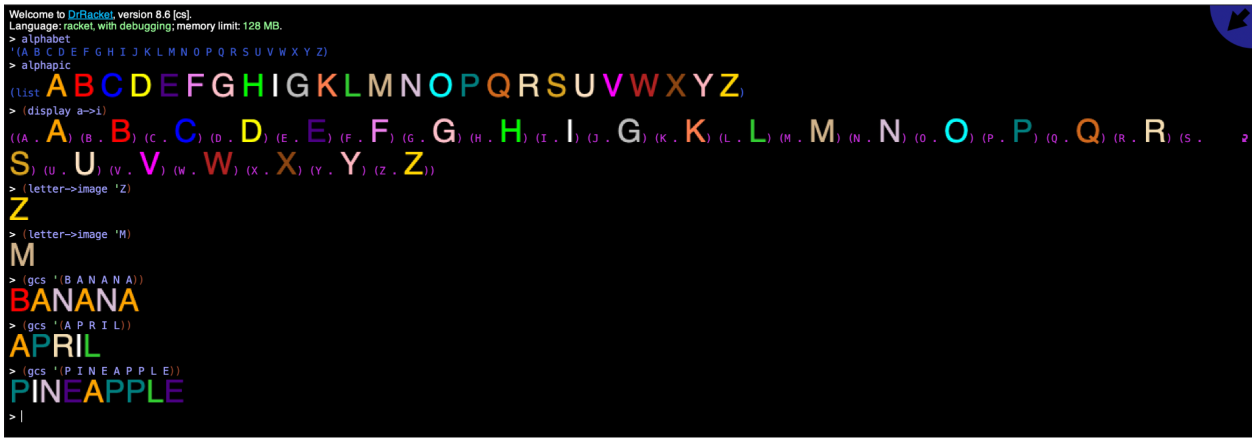
Grapheme Color Synesthesia Demo 1



```
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
>
```

Grapheme Color Synesthesia:

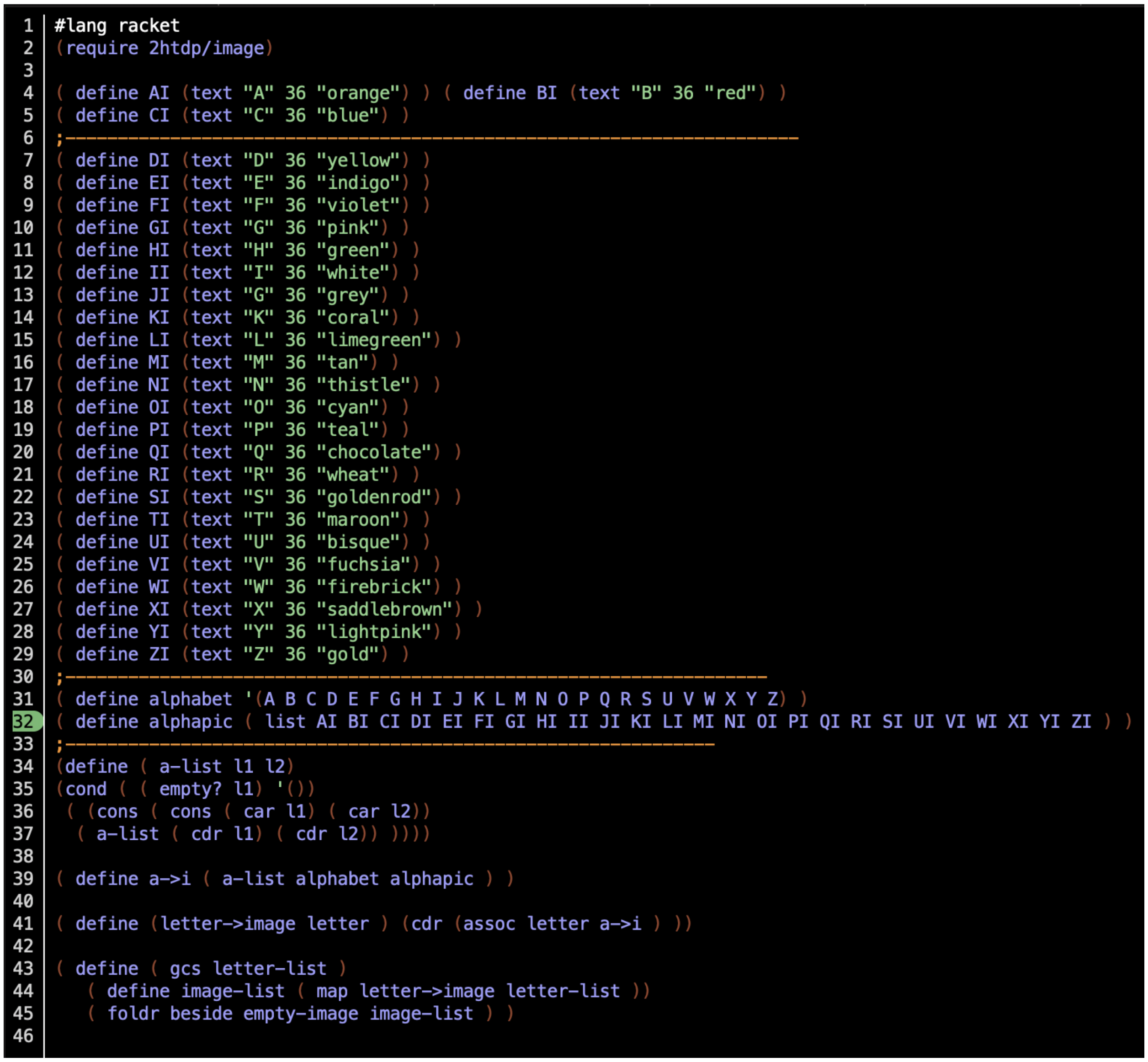
Grapheme Color Synesthesia Demo 2



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> alphabet
'(A B C D E F G H I J K L M N O P Q R S U V W X Y Z)
> alphapic
(list A B C D E F G H I G K L M N O P Q R S U V W X Y Z)
> (display a->i)
((A . A) (B . B) (C . C) (D . D) (E . E) (F . F) (G . G) (H . H) (I . I) (J . G) (K . K) (L . L) (M . M) (N . N) (O . O) (P . P) (Q . Q) (R . R) (S .
S) (U . U) (V . V) (W . W) (X . X) (Y . Y) (Z . Z))
> (letter->image 'Z)
Z
> (letter->image 'M)
M
> (gcs '(B A N A N A))
BANANA
> (gcs '(A P R I L))
APRIL
> (gcs '(P I N E A P P L E))
PINEAPPLE
> |
```

Grapheme Color Synesthesia:

Grapheme Color Synesthesia Code



```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 "yellow") )
( define EI (text "E" 36 "indigo") )
( define FI (text "F" 36 "violet") )
( define GI (text "G" 36 "pink") )
( define HI (text "H" 36 "green") )
( define II (text "I" 36 "white") )
( define JI (text "G" 36 "grey") )
( define KI (text "K" 36 "coral") )
( define LI (text "L" 36 "limegreen") )
( define MI (text "M" 36 "tan") )
( define NI (text "N" 36 "thistle") )
( define OI (text "O" 36 "cyan") )
( define PI (text "P" 36 "teal") )
( define QI (text "Q" 36 "chocolate") )
( define RI (text "R" 36 "wheat") )
( define SI (text "S" 36 "goldenrod") )
( define TI (text "T" 36 "maroon") )
( define UI (text "U" 36 "bisque") )
( define VI (text "V" 36 "fuchsia") )
( define WI (text "W" 36 "firebrick") )
( define XI (text "X" 36 "saddlebrown") )
( define YI (text "Y" 36 "lightpink") )
( define ZI (text "Z" 36 "gold") )
;-------------------------------------------------------------------
( define alphabet '(A B C D E F G H I J K L M N O P Q R S 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 UI VI WI XI YI ZI ) )
;--------------------------------------------------------------
(define ( a-list l1 l2)
(cond ( ( empty? l1) '())
 ( (cons ( cons ( car l1) ( car l2))
  ( a-list ( cdr l1) ( cdr l2)) ))))

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

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

( define ( gcs letter-list )
   ( define image-list ( map letter->image letter-list ))
   ( foldr beside empty-image image-list ) )

```