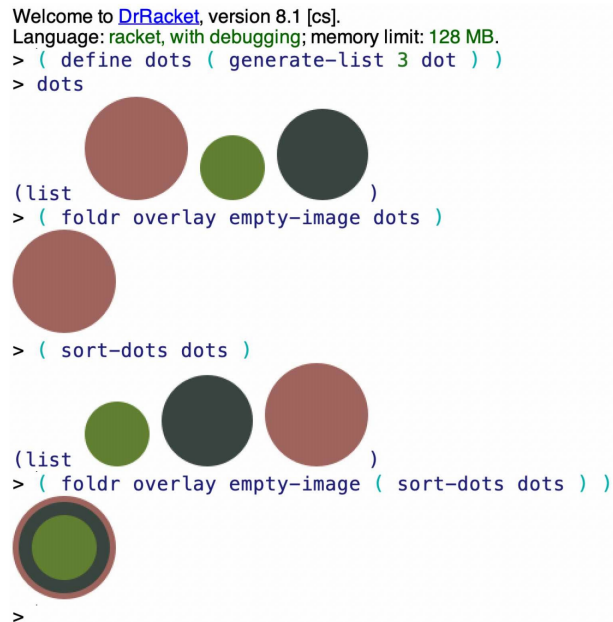

Fourth Racket Programming Assignment Specification

```
Welcome to DrRacket, version 8.1 [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)
        (foldr overlay empty-image (sort-dots dots)))
      )
>
```



Working within the DrRacket PDE, please do each of the following tasks. The first five tasks pertain to straightforward 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.

Task 1 - Generate Uniform List

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

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

```
( define ( a-list listA listB )
  ( cond
    ( ( = 0 ( length listA ) ) '() )
    (
      ( cons
        ( cons ( car listA) ( car listB ) )
        ( a-list ( cdr listA) ( cdr listB ) )
      )
    )
  )
)
```

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

```
( define ( assoc word aList )
  ( cond
    ( ( = 0 ( length aList ) ) '() )
    ( ( eq? word ( car ( car aList ) ) )
      ( car aList )
    )
    ( ( assoc word ( cdr aList ) ) )
  )
)
```

Demo

```
> (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) ) ) )
> ( assoc 'two al1 )
'(two . deux)
> ( assoc 'five al1 )
'()
> ( assoc 'three al2 )
'(three 3 3 3)
> ( assoc 'four al2 )
'()
```

Task 4 – Rassoc

```
( define ( rassoc word aList )  
  ( cond  
    ( ( = 0 (length aList ) ) '() )  
    ( ( eq? word ( cdr ( car aList ) ) )  
      ( car aList )  
    )  
    ( ( rassoc word ( cdr aList ) ) )  
  )  
)
```

Demo

```
> (define al1 ( a-list '(one two three four ) '(un deux trois quatre ) ) )  
> ( rassoc 'three al1 )  
'()  
> ( rassoc 'trois al1 )  
'(three . trois)
```

Task 5 - Los->s

```
(define ( los->s list )
  ( cond
    ( ( = 0 ( length list ) ) "" )
    ( ( = 1 ( length list ) ) ( car list ) )
    ( ( string-append ( car list ) " " ( los->s ( cdr 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

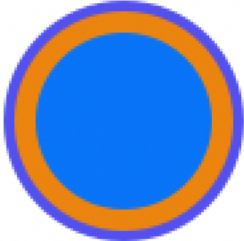
```
(define ( generate-list size func )  
  ( cond  
    ( ( = 1 size ) (cons (func) '() ) )  
    ( ( cons ( func ) ( generate-list ( - size 1 ) func ) ) )  
  )  
)
```

Demo 1

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

Demo 2

```
> ( foldr overlay empty-image (sort-images (generate-list 3 dot) ) )
```

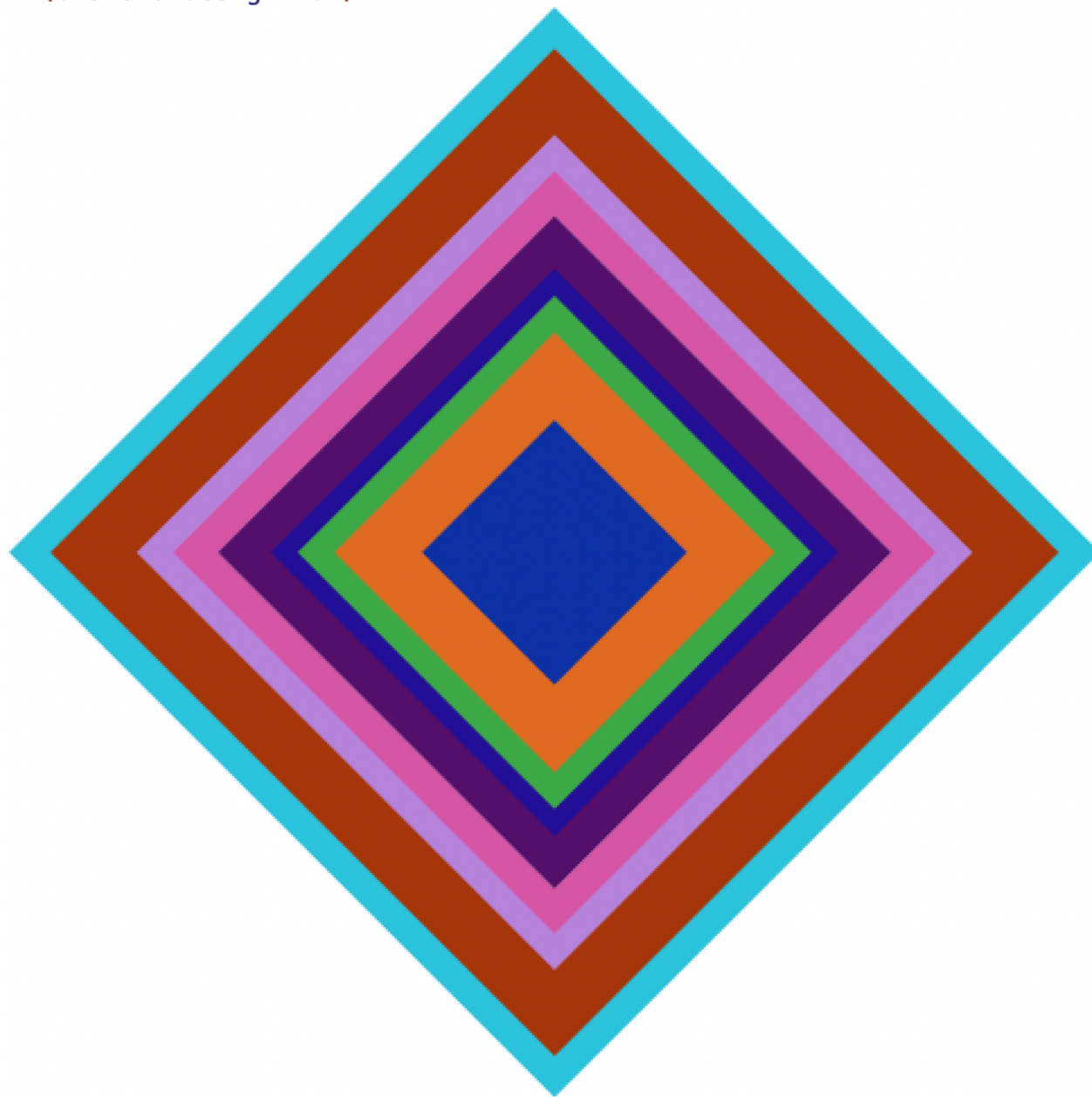


Task 7 - The Diamond

```
( define ( sort-images loc )  
  ( sort loc #:key image-width < )  
)  
  
( define ( diamond )  
  ( rotate 45 ( square ( + 20 ( random 380 ) ) "solid" ( random-color ) ) ) )  
)  
  
( define (diamond-design num)  
  (define diamonds ( sort-images ( generate-list num diamond ) ) )  
  (foldr overlay empty-image diamonds)  
)
```

Demo 1

```
> (diamond-design 10 )
```



Task 8 - Chromesthetic renderings

```
(define ( play song )  
  (foldr beside empty-image ( map box (map pc->color song) ) )  
)
```

Demo

Language: racket, with debugging, memory, minimal GC mem.
> (play '(c d e f g a b c c b a g f e d c))



Task 9 - Diner

```
( define ( total sales item )  
  ( define listOfItems  
    ( filter  
      ( lambda ( listItem ) ( eq? item listItem ) )  
      sales  
    )  
  )  
  ( foldr + 0 ( map ( lambda ( i ) ( cdr ( assoc i menu ) ) ) listOfItems ) )  
)
```

Demo

```
> ( total sales 'malt )  
18
```

Task 10 - Wild Card

```
( define (my-creation)
  (define imageList (filter ( lambda (image) ( < ( image-width image ) 50 ) ) ( generate-list 10 dot ) ) )
  (foldr beside empty-image ( map ( lambda ( image ) ( overlay image (circle 60 "solid" ( random-color ) ) ) ) imageList ) )
)
> (my-creation)
```

