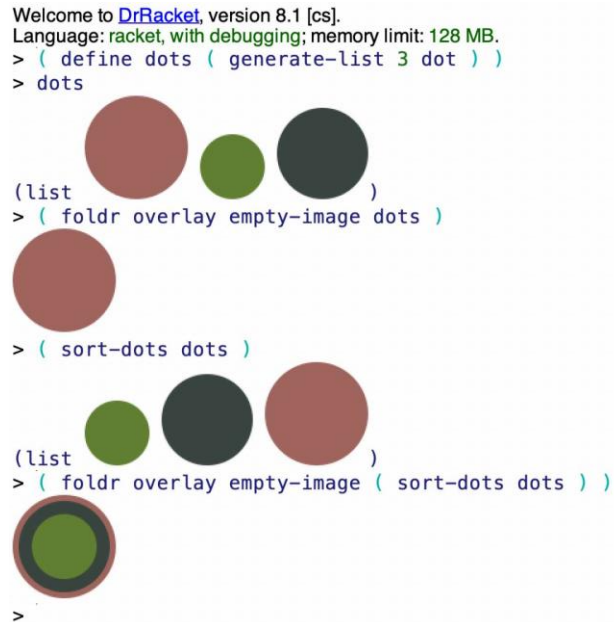

Fourth Racket Programming Assignment Solution By JIUN KIM

```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define dots ( generate-list 3 dot ) )
> dots
(list (image (circle 100 "solid" "red") 100)
      (image (circle 100 "solid" "green") 100)
      (image (circle 100 "solid" "black") 100) )
> ( foldr overlay empty-image dots )
(image (circle 100 "solid" "red") 100)
> ( sort-dots dots )
(list (image (circle 100 "solid" "green") 100)
      (image (circle 100 "solid" "black") 100)
      (image (circle 100 "solid" "red") 100) )
> ( foldr overlay empty-image ( sort-dots dots ) )
(image (circle 100 "solid" "green") 100)
>
```



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

Definition

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

Demo

```
Welcome to DrRacket, version 8.2 [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

Write the recursive function definition, mimic the demo, and build the source and the demo into your presentation document.

Task 2 - Association List Generator

Definition

```
#lang racket
(define ( a-list aList1 aList2 )
  ( cond
    ( ( = ( length aList1 ) 0 ) '() )
    ( ( > ( length aList1 ) 0 )
      ( cons ( cons ( car aList1 ) ( car aList2 ) ) ( a-list ( cdr aList1 ) ( cdr aList2 ) ) )
    )
  )
)
```

Demo

Welcome to [DrRacket](#), version 8.2 [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 ) ' ( this ) )
'((this . this))
> ( a-list '( one two three ) ' ( (1) ( 2 2 ) ( 3 3 3 ) ) )
'((one 1) (two 2 2) (three 3 3 3))
>
```

Task 3 - Assoc





Definition

```
( define ( assoc a b )
  ( cond
    ( ( = ( length b ) 0 ) '() )
    ( ( equal? a (car(car b)) ) ( car b ) )
    ( else
      ( assoc a ( cdr b ) )
    )
  )
)
```

Demo

Welcome to [DrRacket](#), version 8.2 [cs].

Language: racket, with debugging; memory limit: 256 MB.

```
> ( define all
      ( a-list '(one two three four ) '(un deux trois quatre ) )
    )
> ( define al2
      ( a-list '(one two three ) '( (1) (2 2) (3 3 3) ) )
    )
> all
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( assoc 'two all )
'(two . deux)
> ( assoc 'five all )
'()
> al2
  a12: undefined;
cannot reference an identifier before its definition
> al2
'((one 1) (two 2 2) (three 3 3 3))
> ( assoc 'three al2 )
'(three 3 3 3)
> ( assoc 'four al2 )
  a12: undefined;
cannot reference an identifier before its definition
> ( assoc 'four al2 )
'()
>
```

Task 4 - Rassoc

Definition

```
( define ( rassoc a b )
  ( cond
    ( ( = ( length b ) 0 ) ' ( ) )
    ( ( equal? a ( cdr ( car b ) ) ) ( car b ) )
    ( else
      ( rassoc a ( cdr b ) )
    )
  )
)
```

Demo

Welcome to [DrRacket](#), version 8.2 [cs].

Language: racket, with debugging; memory limit: 256 MB.

```
> ( define all
  ( a-list '( one two three four ) '( un deux trois quatre ) )
)
> ( define al2
  ( a-list '( one two three ) '( (1) (2 2) (3 3 3) ) )
)
> all
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( rassoc 'three all )
'()
> ( rassoc 'trois all )
'(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

Definition

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

Demo

```
Welcome to DrRacket, version 8.2 [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

Definition

```
( define ( generate-list a b )
  ( cond
    ( ( = a 0 ) '() )
    ( ( > a 0 )
      ( cons ( b ) ( generate-list ( - a 1 ) b ) )
    )
  )
)
```

Some auxiliary code to support the demo

```
( 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 < )
)
```

Demo 1

```
> ( generate-list 10 roll-die )
'(5 1 4 4 4 4 2 2 5 3)
> ( generate-list 20 roll-die )
'(6 2 1 2 2 3 4 4 4 6 4 1 1 5 4 5 4 5 3 2)
> ( generate-list 12
      ( lambda () (list-ref '( red yellow blue ) ( random 3 ) ) )
    )
'(red red red red yellow blue yellow blue yellow blue yellow red)
```

Demo 2

Welcome to [DrRacket](#), version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.

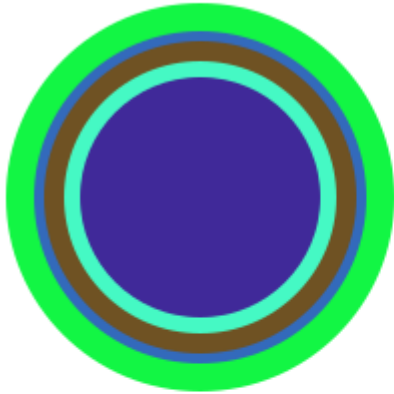
```
> ( define dots ( generate-list 3 dot ) )
> dots

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

> ( sort-dots dots )

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

>
```

Demo 3

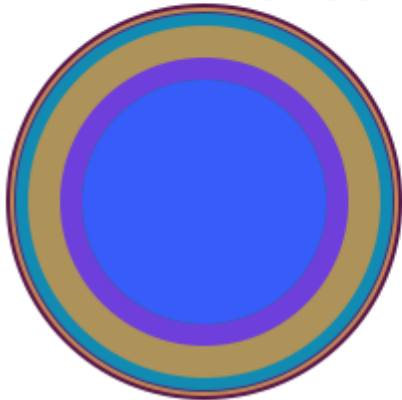
Welcome to [DrRacket](#), version 8.2 [cs].

Language: racket, with debugging; memory limit: 256 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 ) )
```



```
>
```

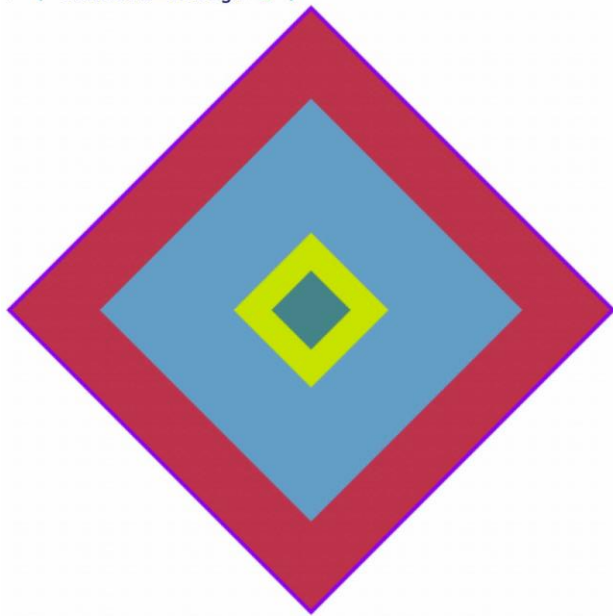
Task 7 - The Diamond

Definition

```
( define ( random-side ) ( random 20 400 ) )  
  
( define ( diamond ) ( rotate 45 ( square ( random-side ) "solid" ( random-color ) ) ) )  
  
( define ( diamond-design n )  
  ( define diamonds ( generate-list n diamond ) )  
  ( foldr overlay empty-image ( sort-dots diamonds) )  
  )
```

Demo 1

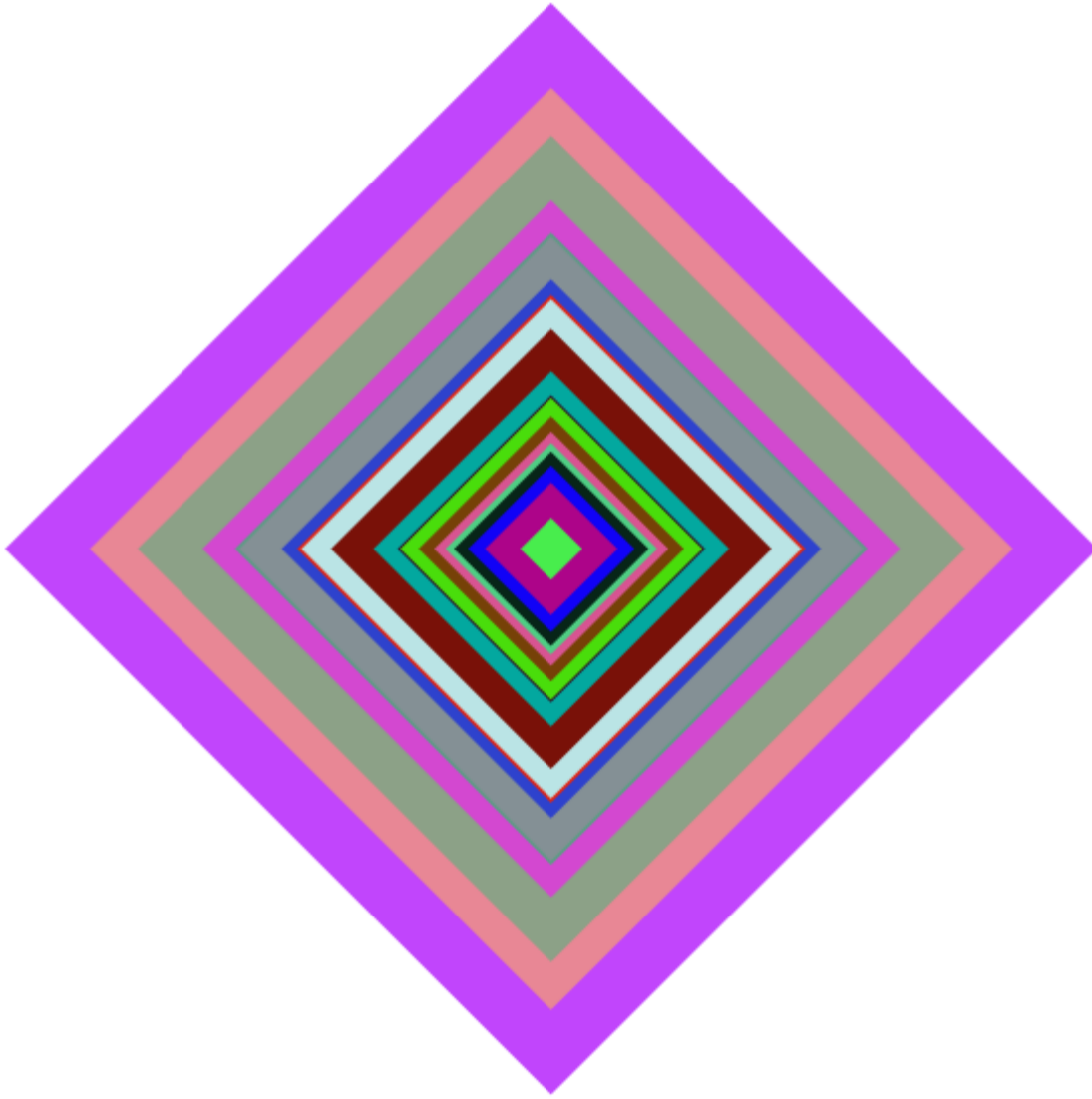
Welcome to [DrRacket](#), version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (diamond-design 5)



> |

Demo 2

Welcome to [DrRacket](#), version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (diamond-design 20)



>

Task 8 - Chromesthetic renderings

Definition

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

Some auxilliary for you to use

```
( 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 ) )
)
```

Demo

Welcome to [DrRacket](#), version 8.2 [cs].

Language: racket, with debugging; memory limit: 256 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))



>

Task 9 - Diner

Definition

```
( define food '( hamburger chickenwings pizza soda water milk ) )

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

( define menu ( a-list food price ) )

( define ( random-sales n )
  ( cond
    ( ( = n 0 )
      '() )
    ( ( > n 0 )
      ( cons ( list-ref food ( random 6 ) ) ( random-sales ( - n 1 ) ) )
    )
  )
)

( define sales ( random-sales 30 ) )

( define ( prices nl )
  ( cdr ( assoc nl menu ) )
)

( define ( total nl foods )
  ( foldr + 0 ( map prices ( filter ( lambda ( n2 ) ( equal? n2 foods ) ) sales ) ) )
)
```

Demo

```
Welcome to DrRacket, version 8.2 [cs]
Language: racket, with debugging; memory limit: 256 MB.
> menu
'[(hamburger . 5.5) (chickenwings . 4.5) (pizza . 3) (soda . 1) (water . 1) (milk . 3.5)]
> sales
'(hamburger milk soda chickenwings soda milk hamburger water soda milk water soda hamburger water pizza soda hamburger milk chickenwings milk soda hamburger hamburger milk milk hamburger soda soda hamburger water)
> (total sales 'hamburger)
44.0
> (total sales 'chickenwings )
5.0
> (total sales 'pizza )
3
> (total sales 'soda )
8
> (total sales 'water )
4
> (total sales 'milk )
24.5
>
```

Task 10 - Wild Card

Definition

```
( define color-classes '( b g r y o d w ) )

( define color-name '( blue green red yellow orange black white ) )

( define ( line color )
  ( rectangle 200 35 "solid" color )
)

( define lines
  (list
    ( line "blue" )
    ( line "green" )
    ( line "red" )
    ( line "yellow" )
    ( line "orange" )
    ( line "black" )
    ( line "white" )
  )
)

( define cc-a-list ( a-list color-classes color-name ) )
( define cl-a-list ( a-list color-name lines ) )
( define ( cc->color cc ) ( cdr ( assoc cc cc-a-list ) ) )
( define ( color->line color ) ( cdr ( assoc color cl-a-list ) ) )

( define ( draw a )
  ( foldr above empty-image ( map color->line ( map cc->color a ) ) ) )
```

Demo

Welcome to [DrRacket](#), version 8.2 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (draw '(d r o))



> (draw ' (r w r))



> (draw ' (w b r))



> |

I used foldr and map to express some of European national flag.

1. Try to express German national flag.
2. Try to express Austrian national flag.
3. Try to express Russian national flag.

With foldr and map function, we can draw most of european national flag.