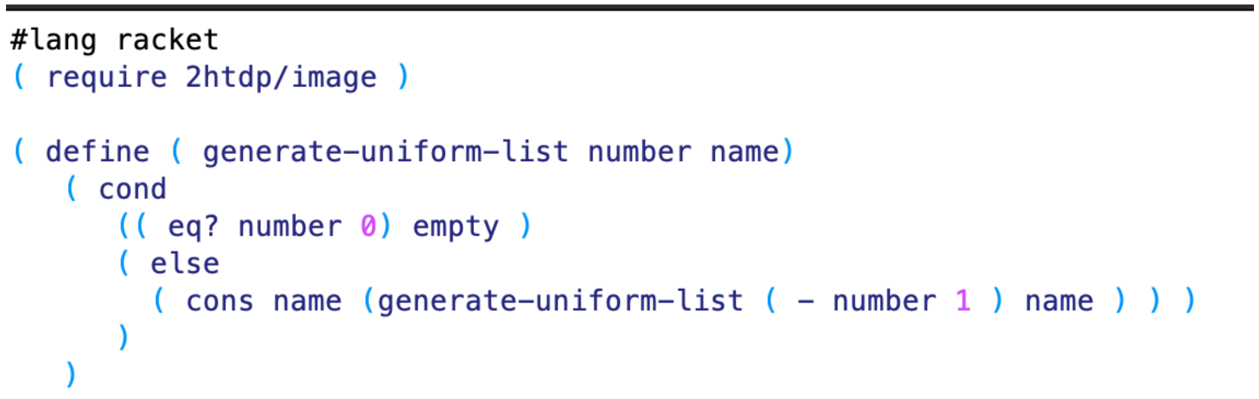
Fourth Racket Programming Assignment

Learning Abstract :

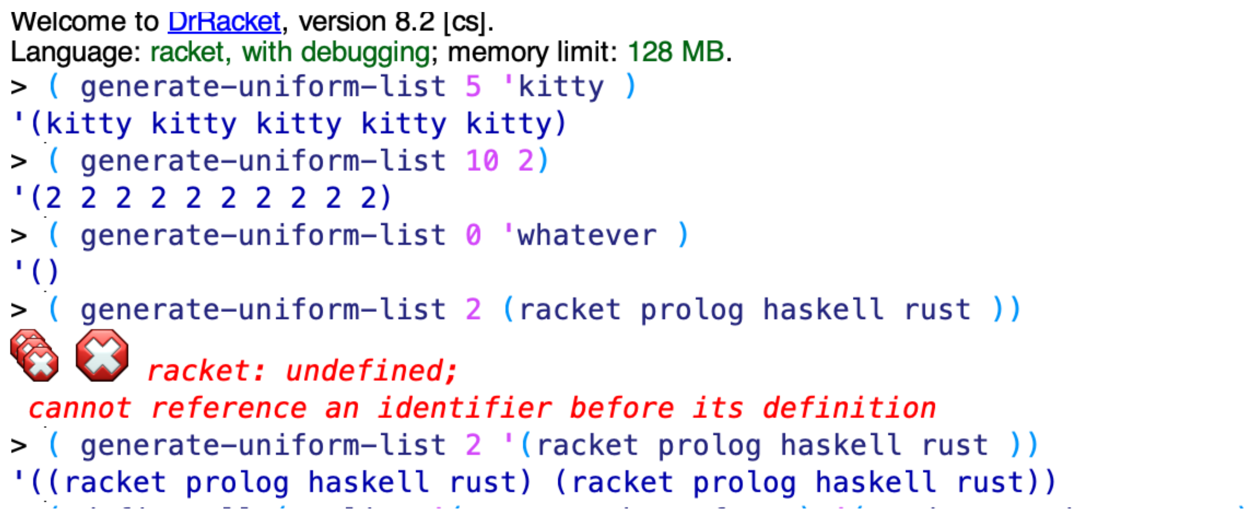
What I learned in this assignment is how to use things such as list foldr and as a whole, how to program in a completely different manner using lisp. For example, we have printed the same character multiple times. In a different assignment it was called copies and in this one it is called generate-uniform-list and we use else statements and ‘eq?’ Instead of ‘display’.

Task 1: Generate Uniform List



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

( define ( generate-uniform-list number name)
   ( cond
      (( eq? number 0) empty )
      ( else
        ( cons name (generate-uniform-list ( - number 1 ) name ) ) )
      )
   )
```

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: undefined;
 cannot reference an identifier before its definition
> ( generate-uniform-list 2 '(racket prolog haskell rust ))
'((racket prolog haskell rust) (racket prolog haskell rust))
```

Task 2 : Associate List Generator

```
( define ( a-list lst1 lst2 )
   ( cond
      (( eq? (length lst1 ) 0 ) empty )
      ( else
        ( map cons lst1 lst2 ) )
      )
   )
```

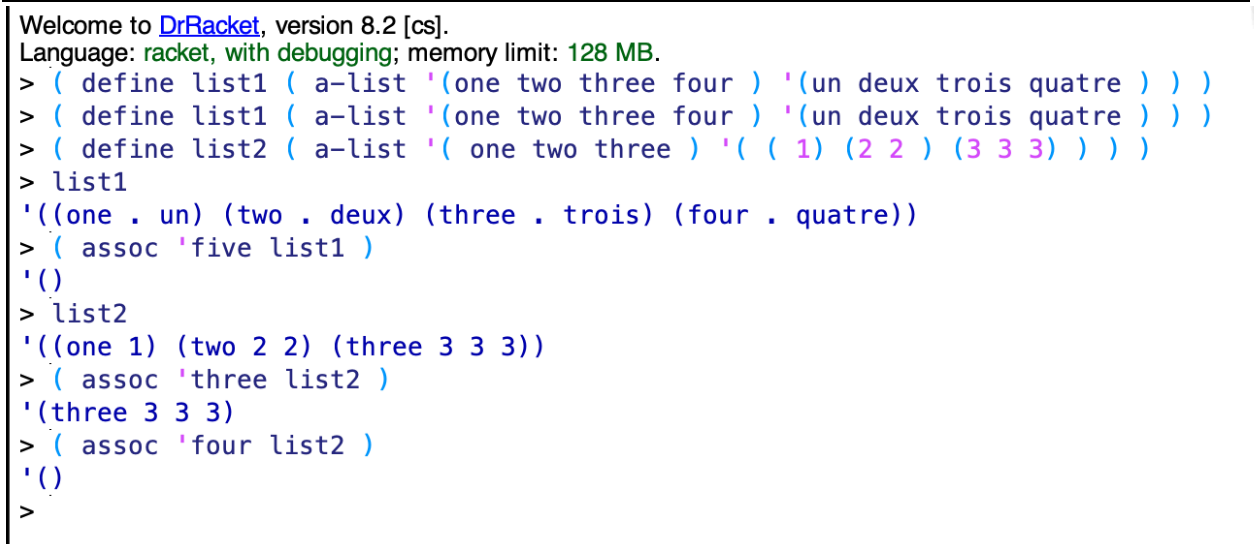
Demo

```
> ( define all ( a-list '(one two three four ) '(un deux trois quatre ) ) )
> ( a-list '(one two three four five ) '(un deux trois quatre cinq ) )
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
> ( 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 object assoclist )
   ( cond
      ( ( eq? (length assoclist ) 0 ) empty )

     ( ( eq? ( car (car assoclist ) ) object )
       ( car assoclist )
       )
      ( else
        ( assoc object ( cdr assoclist ) )
        )
      )
   )
```

Demo

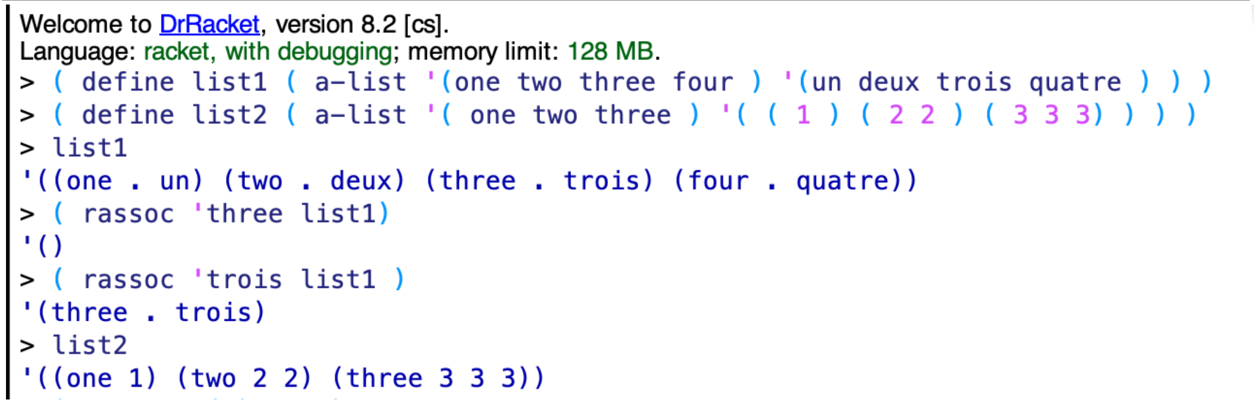


```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define list1 ( a-list '(one two three four ) '(un deux trois quatre ) ) )
> ( define list1 ( a-list '(one two three four ) '(un deux trois quatre ) ) )
> ( define list2 ( a-list '( one two three ) '( ( 1) (2 2 ) (3 3 3) ) ) )
> list1
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( assoc 'five list1 )
'()
> list2
'((one 1) (two 2 2) (three 3 3 3))
> ( assoc 'three list2 )
'(three 3 3 3)
> ( assoc 'four list2 )
'()
>
```

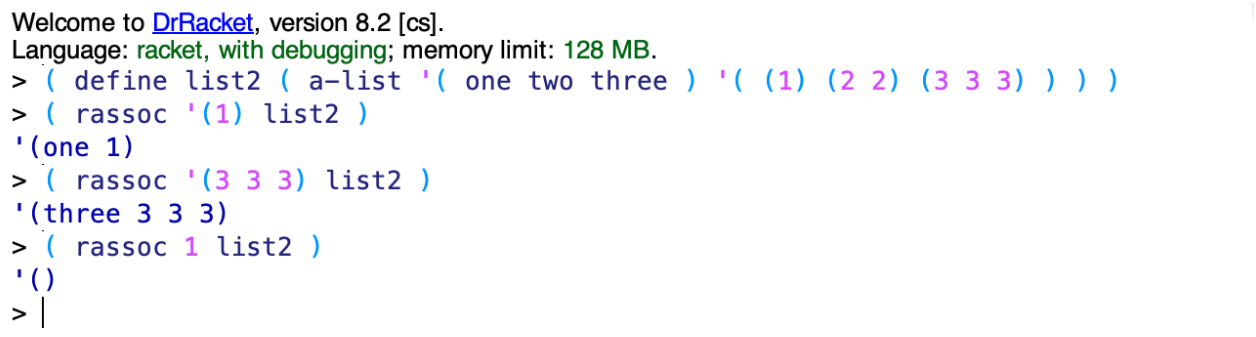
Task 4 : Rassoc

```
( define ( rassoc name assoclist )
   ( cond
      ( ( eq? ( length assoclist ) 0 ) empty )

      ( ( equal? ( cdr ( car assoclist ) ) name )
        ( car assoclist )
        )
      ( else
        ( rassoc name ( cdr assoclist ) )
        )
      )
   )
```

Demo



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define list1 ( a-list '(one two three four ) '(un deux trois quatre ) ) )
> ( define list2 ( a-list '( one two three ) '( ( 1 ) ( 2 2 ) ( 3 3 3) ) ) )
> list1
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( rassoc 'three list1)
'()
> ( rassoc 'trois list1 )
'(three . trois)
> list2
'((one 1) (two 2 2) (three 3 3 3))
```

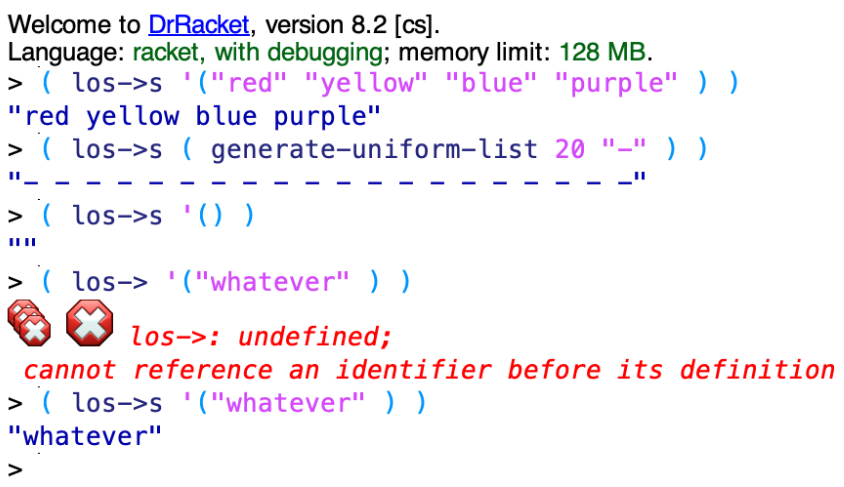


```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define list2 ( a-list '( one two three ) '( (1) (2 2) (3 3 3) ) ) )
> ( rassoc '(1) list2 )
'(one 1)
> ( rassoc '(3 3 3) list2 )
'(three 3 3 3)
> ( rassoc 1 list2 )
'()
> 
```

Task 5 : Los->5

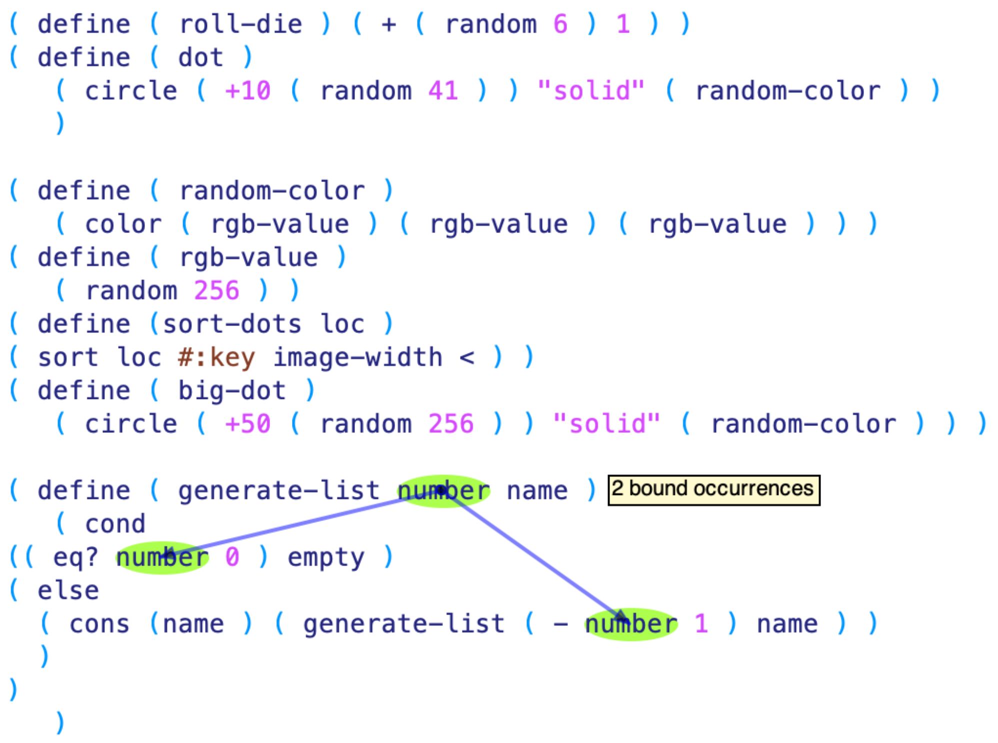
```
( define ( los->s lst )
   ( cond
      ( ( = ( length lst ) 0 ) "")
      ( ( = ( length lst ) 1 )
        ( car lst ) )
      ( ( > ( length lst 1 )
          ( string-append ( car lst ) " " ( los->s (cdr lst ) ) )
          )
        )
      )
   )
```

Demo



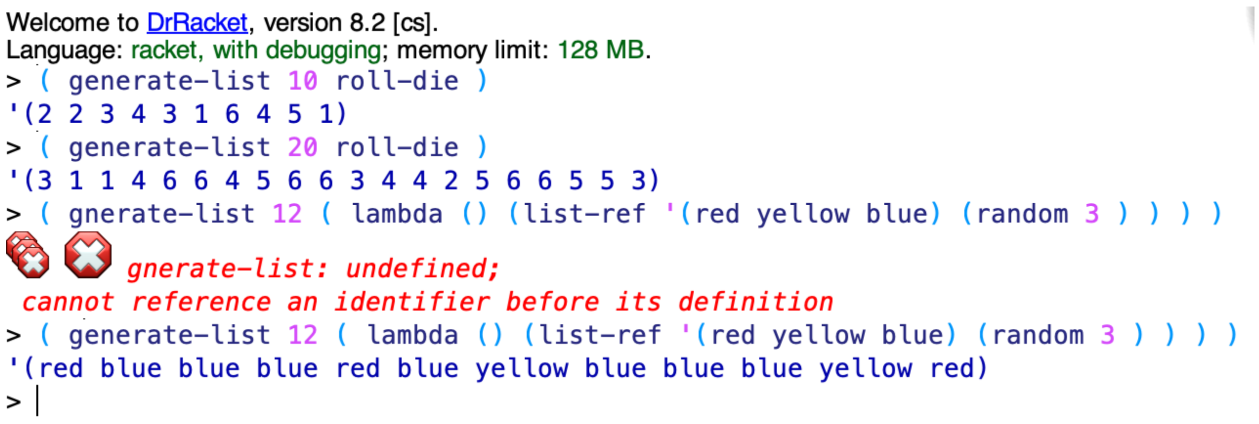
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( los->s '("red" "yellow" "blue" "purple" ) )
"red yellow blue purple"
> ( los->s ( generate-uniform-list 20 "-" ) )
"- - - - - - - - - - - - - - - - - - - -"
> ( los->s '() )
""
> ( los-> '("whatever" ) )
los->: undefined;
 cannot reference an identifier before its definition
> ( los->s '("whatever" ) )
"whatever"
>
```

Task 6 : Generate List



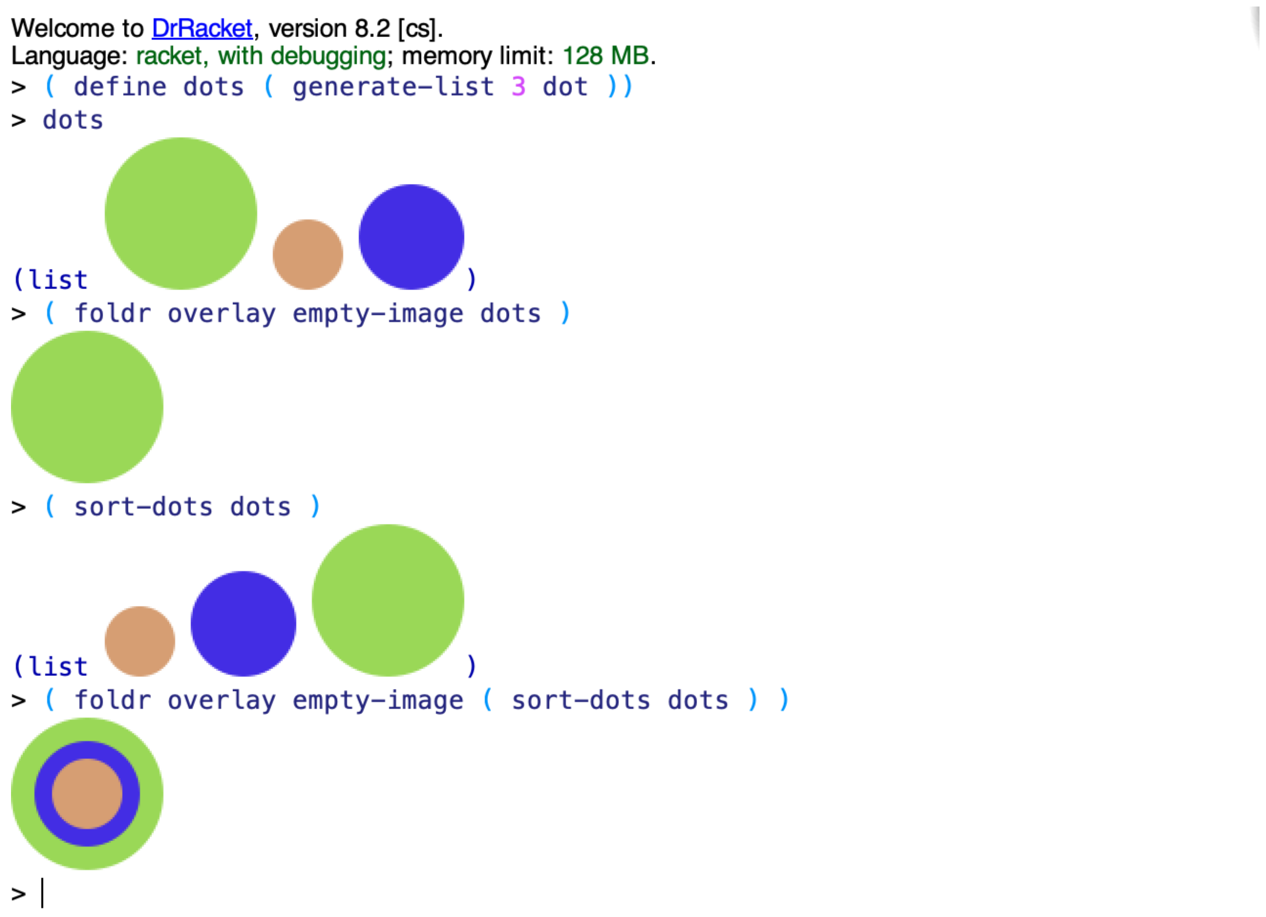
```
( define ( roll-die ) ( + ( random 6 ) 1 ) )
( define ( dot )
   ( circle ( +10 ( random 41 ) ) "solid" ( random-color ) )
   )

( 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 ( big-dot )
   ( circle ( +50 ( random 256 ) ) "solid" ( random-color ) ) )

( define ( generate-list number name )
   ( cond
(( eq? number 0 ) empty )
( else
  ( cons (name ) ( generate-list ( - number 1 ) name ) )
  )
)
   )
```

Demo



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( generate-list 10 roll-die )
'(2 2 3 4 3 1 6 4 5 1)
> ( generate-list 20 roll-die )
'(3 1 1 4 6 6 4 5 6 6 3 4 4 2 5 6 6 5 5 3)
> ( gnerate-list 12 ( lambda () (list-ref '(red yellow blue) (random 3 ) ) ) )
gnerate-list: undefined;
 cannot reference an identifier before its definition
> ( generate-list 12 ( lambda () (list-ref '(red yellow blue) (random 3 ) ) ) )
'(red blue blue blue red blue yellow blue blue blue yellow red)
> |
```

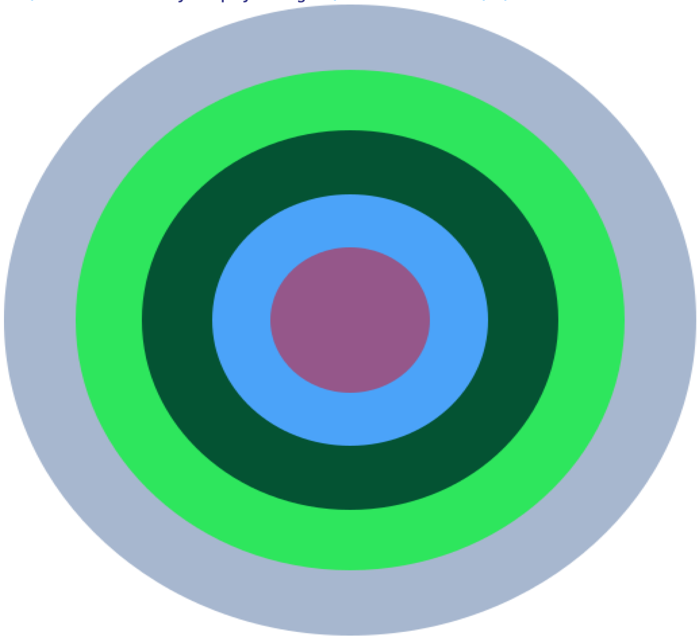
Demo 2



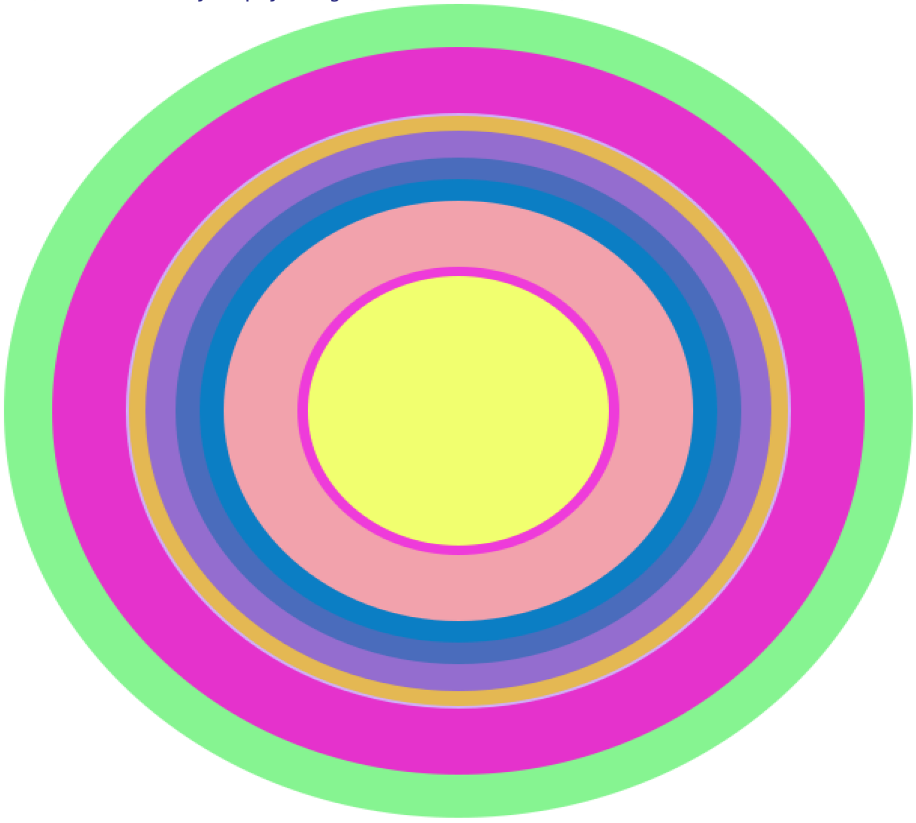
```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( define dots ( generate-list 3 dot ))
> dots
(list                         )
> ( foldr overlay empty-image dots )

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

> |
```

Demo 3

```
Welcome to DrRacket, version 8.2 [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 ) )
```

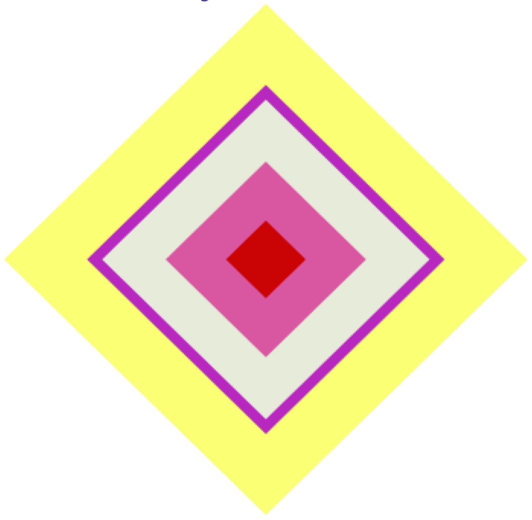


Task 7: The Diamond

```
( 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 ( big-dot )
   ( circle ( + 50 ( random 256 ) ) "solid" ( random-color ) ) )

( define ( generate-list number name )
   ( cond
(( eq? number 0 ) empty )
( else
  ( cons (name ) ( generate-list ( - number 1 ) name ) )
  )
)
   )
( define ( diamond )
   ( rotate 45 ( square ( random 20 400) "solid" ( random-color ) )
            )
   )
( define ( diamond-design n )
   ( define a ( generate-list n diamond ) )
   ( foldr overlay empty-image ( sort-dots a )
           )
   )
```

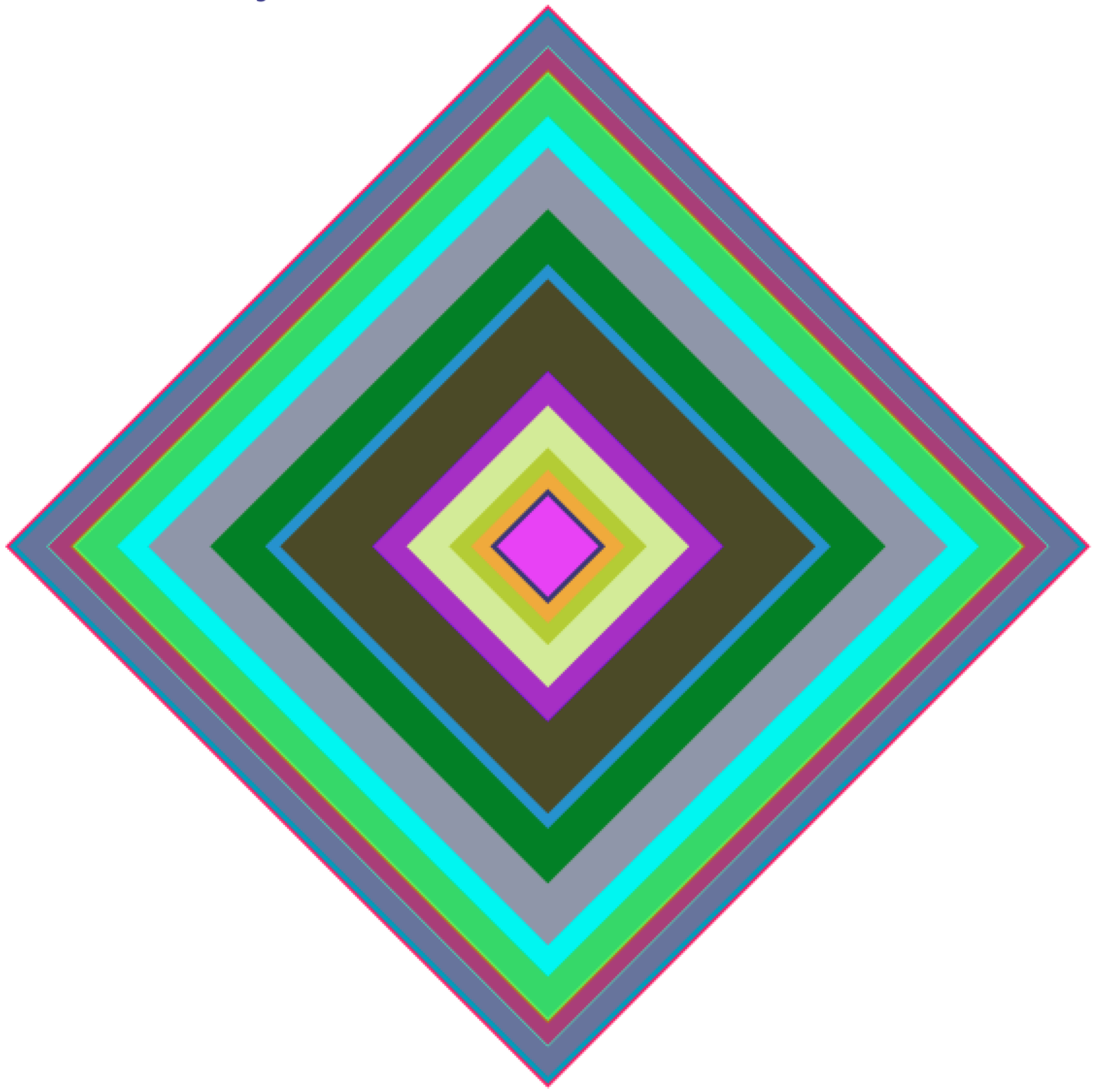
Demo

```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( diamond-design 5 )
```



Demo

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

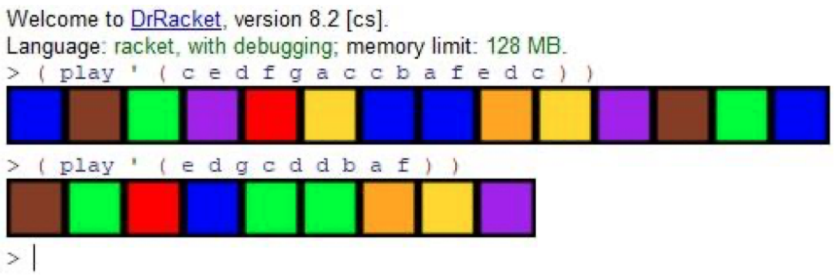


```
>
```

Task 8 - Chromesthetic Renderings

```
( 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 square-color )
   ( foldr beside empty-image ( map box ( map pc->color square-color) ) )
   )
```

Demo

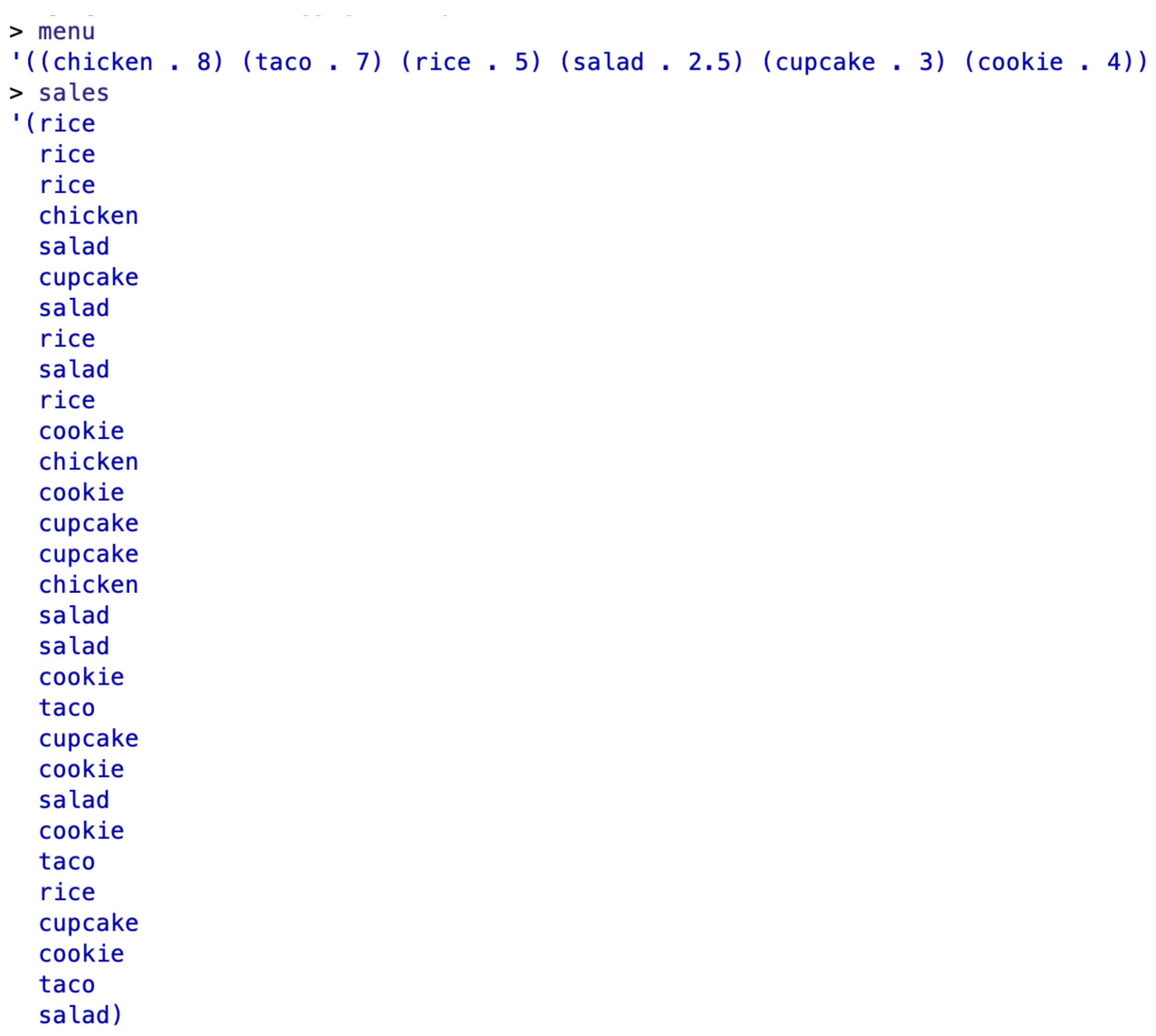


```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( play ' ( c e d f g a c c b a f e d c ) )
> ( play ' ( e d g c d d b a f ) )
> |
```

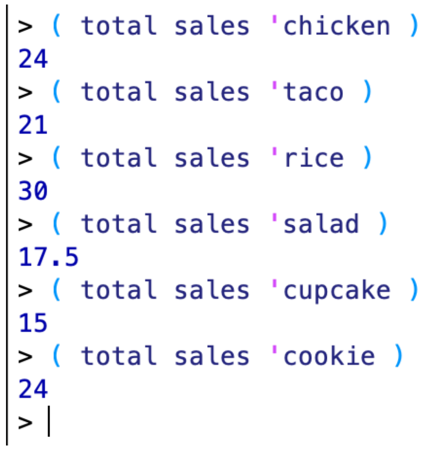
Task 9 - Diner

```scheme
( define food '(chicken taco rice salad cupcake cookie ) )
( define price '(8 7 5 2.5 3 4  ) )
( define menu ( a-list food price ) )

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

( define sales ( randomfood 30 ))

( define (prices n )
   ( cond
      ( ( eq? #f ( member n food ) ) 0 )
      ( else
        ( cdr ( assoc n menu ) )
        )
      )
   )

( define ( total n lst )
   ( foldr + 0 ( map prices ( filter ( lambda (x) (equal? x lst ) ) n ) ) )
   )
```

Demo



```
> menu
'((chicken . 8) (taco . 7) (rice . 5) (salad . 2.5) (cupcake . 3) (cookie . 4))
> sales
'(rice
  rice
  rice
  chicken
  salad
  cupcake
  salad
  rice
  salad
  rice
  cookie
  chicken
  cookie
  cupcake
  cupcake
  chicken
  salad
  salad
  cookie
  taco
  cupcake
  cookie
  salad
  cookie
  taco
  rice
  cupcake
  cookie
  taco
  salad)
```

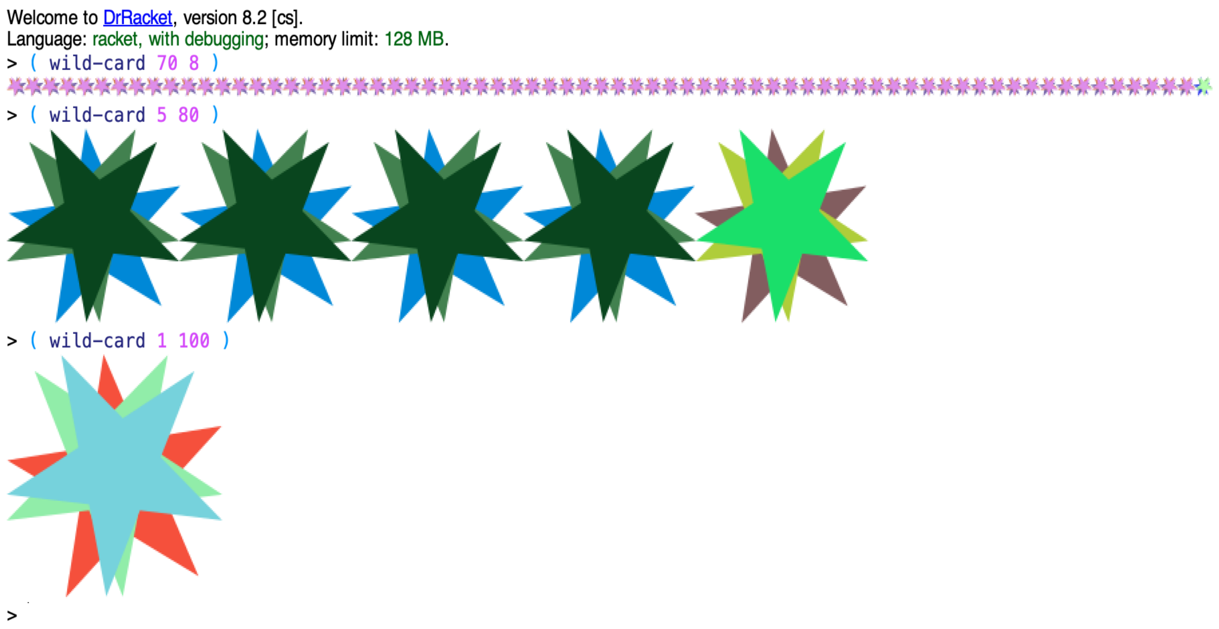


```
> ( total sales 'chicken )
24
> ( total sales 'taco )
21
> ( total sales 'rice )
30
> ( total sales 'salad )
17.5
> ( total sales 'cupcake )
15
> ( total sales 'cookie )
24
> |
```

Task 10 :

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

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




( define (arrow n s )
   ( define part ( rotate 30 ( star s "solid" ( random-color ) ) ))
   ( define head ( rotate 30 ( star s "solid" ( random-color ) ) ) )
   ( define stem ( foldr beside empty-image ( make-list ( - n 1 ) part ) ) )
   ( beside stem head )
   )

( define (part1 n s )
   ( define part1 ( rotate 30 ( star s "solid" ( random-color ) ) ))
   ( define second ( rotate 30 ( star s "solid" ( random-color ) ) ) )
   ( define together ( foldr beside empty-image ( make-list ( - n 1 ) part1 ) ) )
   ( beside together second )
   )

( define (part2 n s)
   ( define part2 ( rotate -30 ( star s "solid" ( random-color ) ) ))
   ( define second2 ( rotate -30 ( star s "solid" ( random-color ) ) ) )
   ( define together2 ( foldr beside empty-image ( make-list ( - n 1 ) part2 ) ) )
   ( beside together2 second2 )
   )
( define (part3 n s)
   ( define part3 ( rotate 80 ( star s "solid" ( random-color ) ) ))
   ( define second3 ( rotate 80 ( star s "solid" ( random-color ) ) ) )
   ( define together3 ( foldr beside empty-image ( make-list ( - n 1 ) part3 ) ) )
   ( beside together3 second3 )
   )

( define ( wild-card n s )
   ( overlay
     ( part1 n s)
     ( part2 n s)
     ( part3 n s)
     )
   )
```

Demo



```
Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( wild-card 70 8 )
```

```
> ( wild-card 5 80 )
```

```
> ( wild-card 1 100 )
```

```
>
```