Racket Assignment #3: Recursions in Racket

Learning Abstract:

This lab was very helpful for us to understand recursion in detail. We solved various problems using recursion only and that also helped us understand proper syntax and ways to use recursion in this particular language, Racket. I enjoyed doing this assignment and I implemented what I learnt in class here.

Task 1: Counting Down, Counting Up

Code

```
#lang racket
( define ( count-down n )
   ( cond
      ((= n 1 ) ( display n ))
      ( ( > n 0 ) ( display n )
                  ( display "\n" )
                  ( count-down (- n 1) ))
   )
   )

( define ( count-up num )
   ( cond
      ( ( = num 1 ) ( display num )
                  ( display "\n" )
                  )
      ( ( > num 1 ) ( count-up ( - num 1 ))
                  ( display num )
                  ( display "\n" )
                  )
      )
   )
```

Demo

```
> ( count-down 5 )
5
4
3
2
1
> ( count-down 10 )
10
9
8
7
6
5
4
3
2
1
> ( count-down 20 )
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
```

```
> ( count-up 5 )
1
2
3
4
5
> ( count-up 10 )
1
2
3
4
5
6
7
8
9
10
> ( count-up 20 )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Task 2: Triangle of Stars

Code

```
( define ( row-of-stars n )
   ( cond
      (( = n 0 ) ( display "\n" ))
      ((> n 0 ) ( display "*" ) ( row-of-stars (- n 1 ) ) )
      )
   )

( define ( triangle-of-stars int )
   ( cond
      ( ( = int 0 ) ( display "" ))
      ( ( > int 0 ) ( triangle-of-stars ( - int 1 ) )
                    ( row-of-stars int  )
                    )
      )
   )
```

Demo

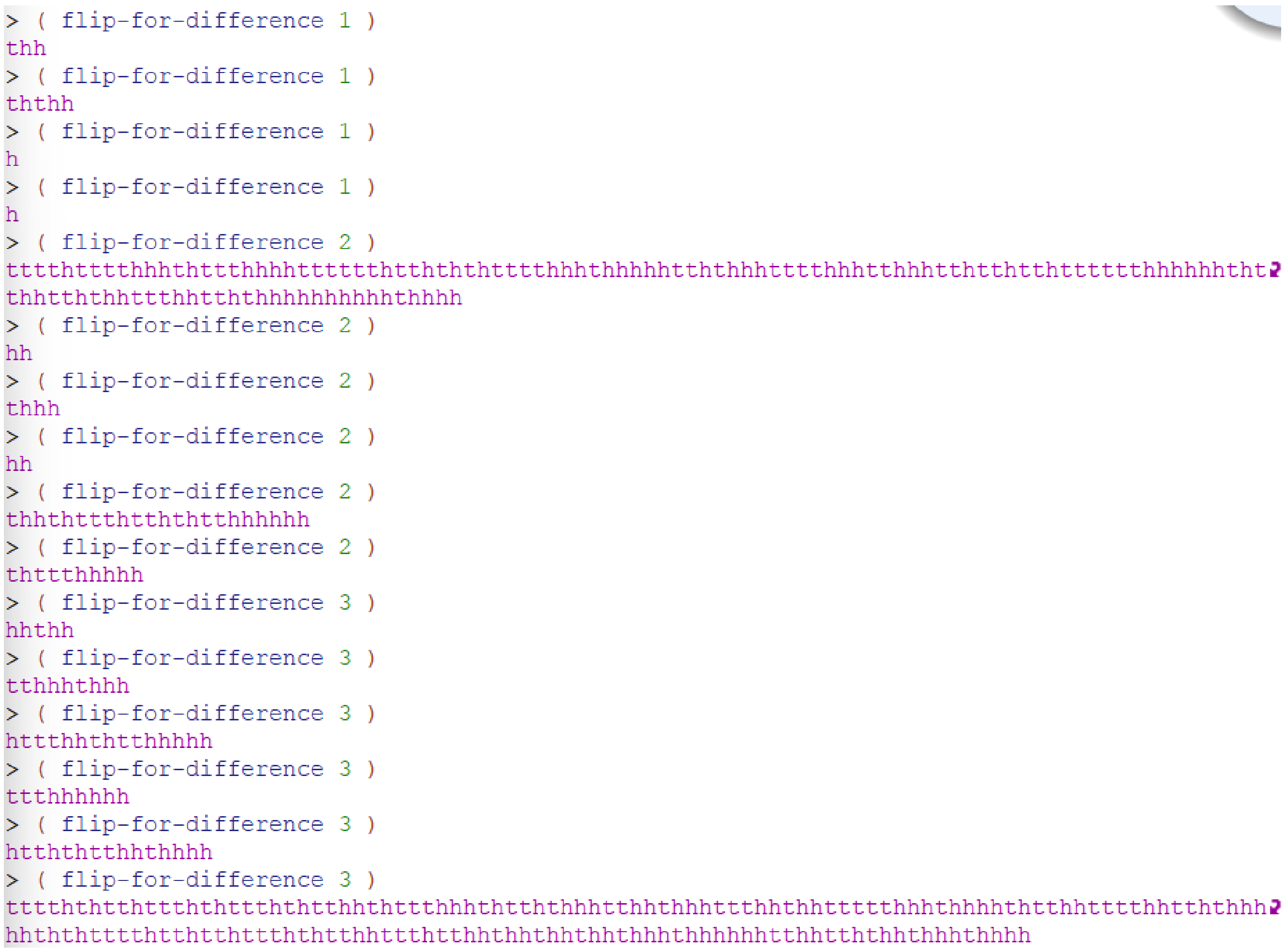
```
> ( triangle-of-stars 5 )
*
**
***
****
*****
> ( triangle-of-stars 0 )
> ( triangle-of-stars 15 )
*
**
***
****
*****
******
*******
********
*********
**********
***********
************
*************
**************
***************
>
```

Task 3: Flipping a Coin

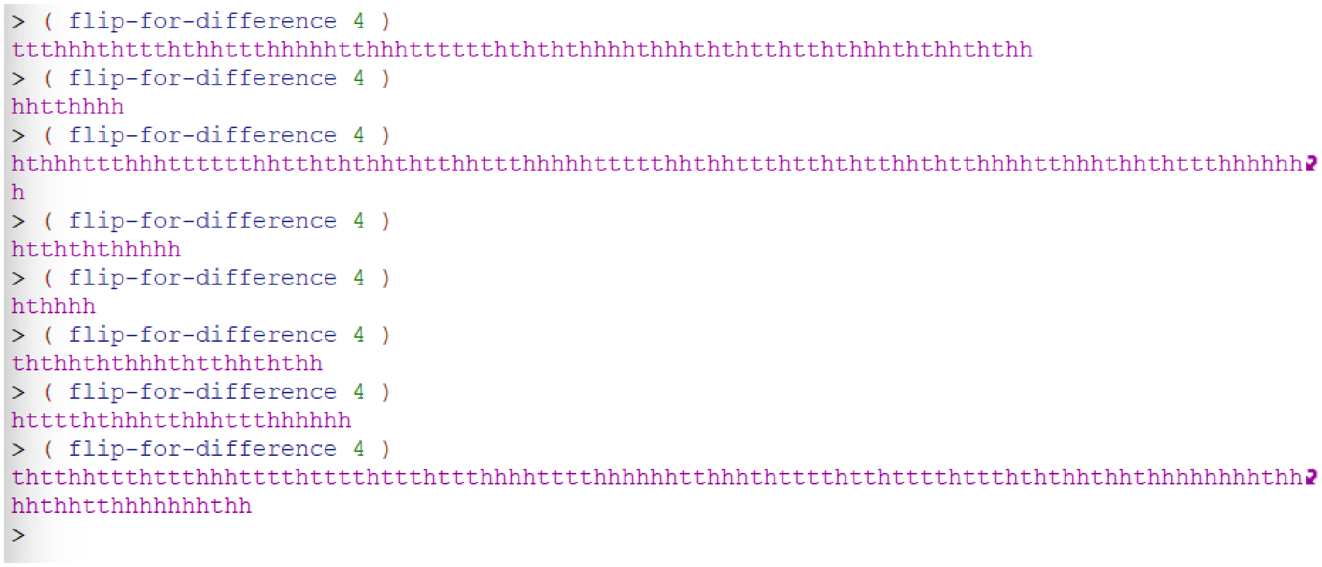
Code

```
#lang racket
( define head-count 0 )
( define tail-count 0 )


( define ( flip )
   ( define outcome ( random 2 ) )
   ( cond
      ((eq? outcome 0) (set! tail-count (+ tail-count 1 ) ) ( display "t" ) )
      ((eq? outcome 1) (set! head-count (+ head-count 1) ) ( display "h" ))
      )
   )



( define ( flip-for-difference diff )
   ( cond
      (( eq? diff (- head-count tail-count ) ) ( display "") (set! tail-count
 0) (set! head-count 0 ))
      ((not ( eq? diff (- head-count tail-count )))
       (flip)
       ( flip-for-difference diff )
      )
      )
   )
```

The Demo



```
> ( flip-for-difference 1 )
thh
> ( flip-for-difference 1 )
ththh
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
h
> ( flip-for-difference 2 )
ttttthttthhhthtttthhhhtttttthththththtttthhhthhhhhttththhhtttthhhtthhhththththttthtttttthhhhhhtht
thhtththhhtttthhhtththhhhhhhhhhhthhhh
> ( flip-for-difference 2 )
hh
> ( flip-for-difference 2 )
thhh
> ( flip-for-difference 2 )
hh
> ( flip-for-difference 2 )
thhthttththththtthhhhhh
> ( flip-for-difference 2 )
thttthhhhh
> ( flip-for-difference 3 )
hhthh
> ( flip-for-difference 3 )
tthhhthhh
> ( flip-for-difference 3 )
httthhthtthhhhh
> ( flip-for-difference 3 )
ttthhhhhh
> ( flip-for-difference 3 )
htthththhhthhhh
> ( flip-for-difference 3 )
ttttththtthttththtttththtthhthtttthhhthtththhhtthhthhhttthhthhhttttthhhthhhhthtthhttttthhttththhh
hhththttttthtthtthttththtthhtttthtthhthhthhthhthhhthhhhhhtthhttththhthhhhthhhh
```

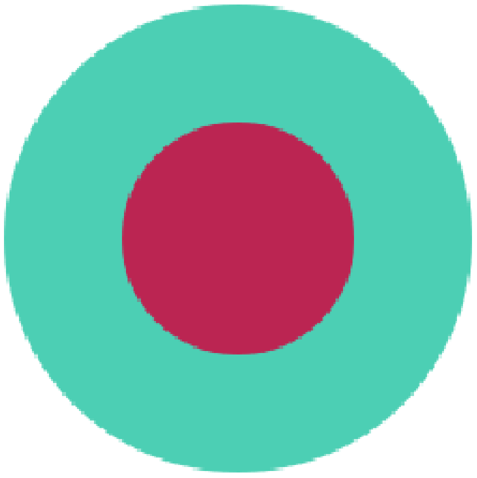


```
> ( flip-for-difference 4 )
ttthhhthtttthhhttthhhhhtthhhttttttththththhhhthhhthththttthththhhhththhhththh
> ( flip-for-difference 4 )
hhtthhhh
> ( flip-for-difference 4 )
hthhhtttthhhttttttthhtththththhthtthhttthhhhhtttttthhthhttththththtthhthtthhhhtthhhthhthtttthhhhhh
h
> ( flip-for-difference 4 )
httththhhhhh
> ( flip-for-difference 4 )
hthhhh
> ( flip-for-difference 4 )
ththhththhhhthtthhththh
> ( flip-for-difference 4 )
htttththhhtthhhtttthhhhhh
> ( flip-for-difference 4 )
thtthhtttthtttthhhttttthttttthtttthtttthhhhhttttthhhhhhtthhhthttttthtthttttthttththhthhthhhhhhhhhthh
hhthhtthhhhhhhhthh
>
```

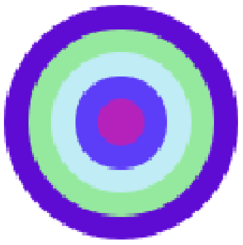
Task 4: Laying Down Colorful Concentric Disks

CCR Demo

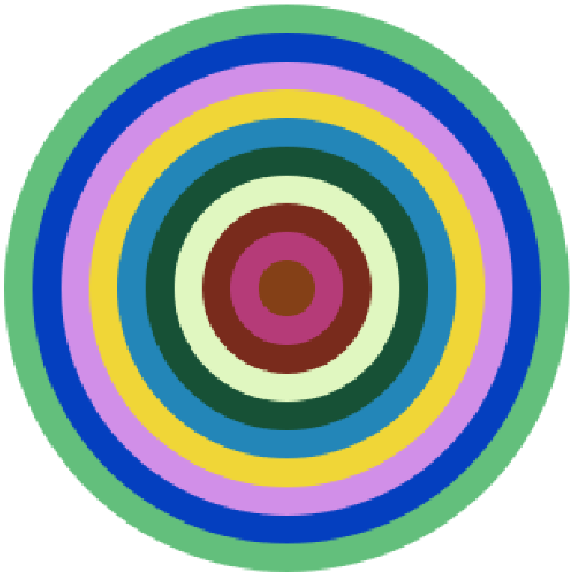
```
> ( ccr 100 50 )
```



```
> ( ccr 50 10 )
```



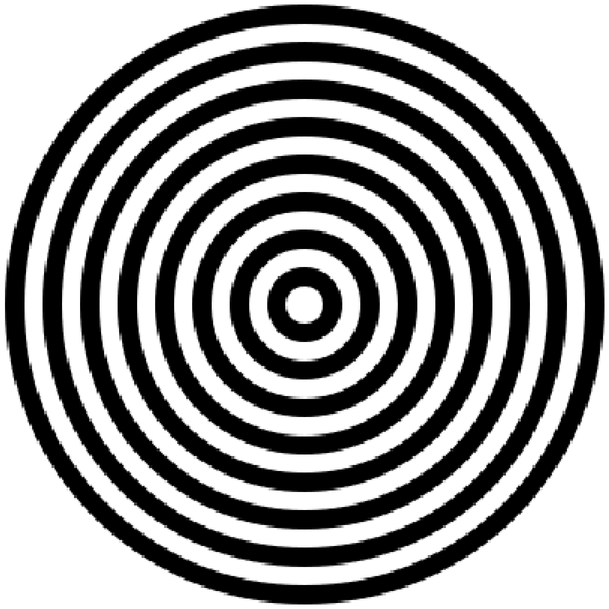
```
> ( ccr 150 15 )
```



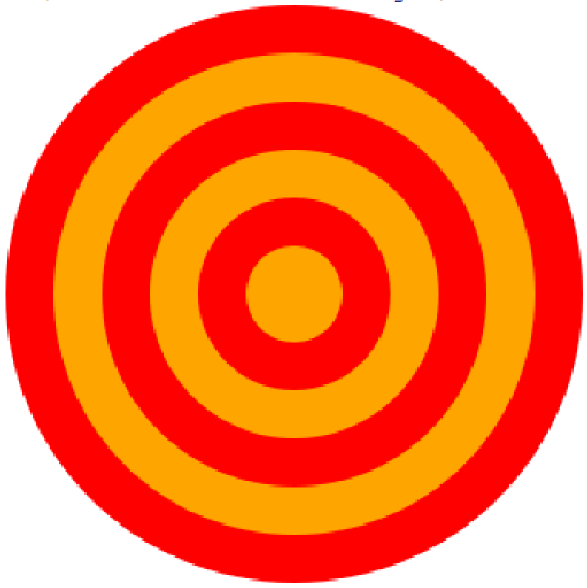
```
>
```

CCA Demo

```
> ( cca 160 10 'black 'white )
```



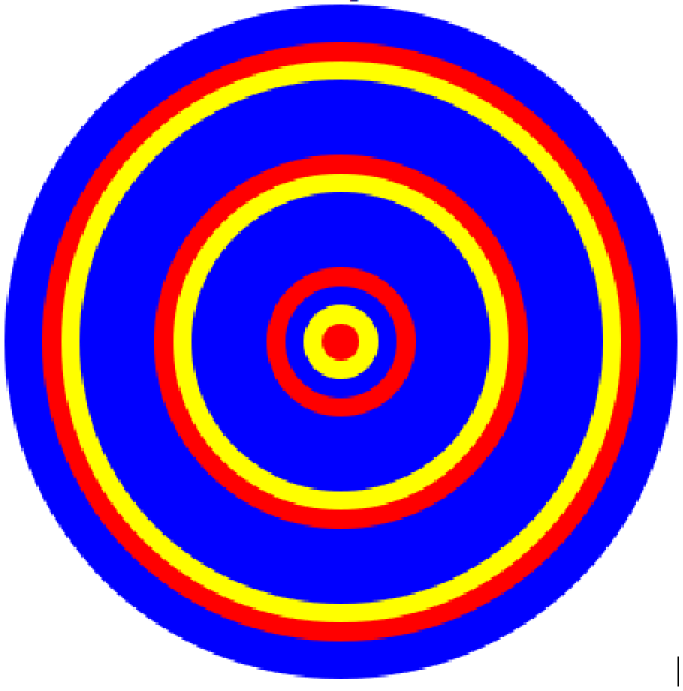
```
> ( cca 150 25 'red 'orange )
```



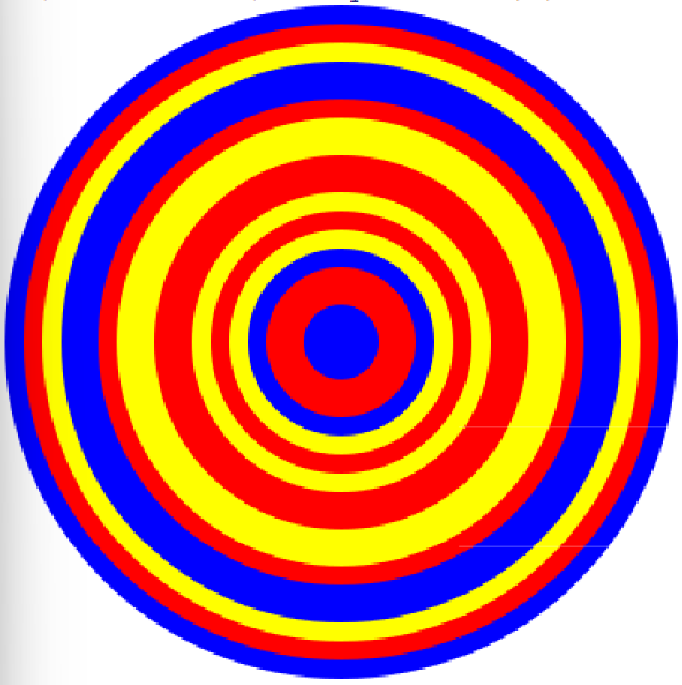
```
>
```

CCS Demo 1

```
> ( ccs 180 10 '( blue yellow red ) )
```



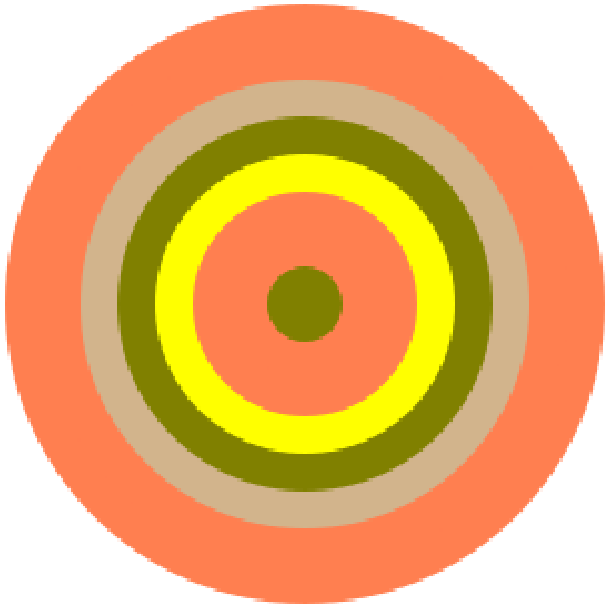
```
> ( ccs 180 10 '( blue yellow red ) )
```



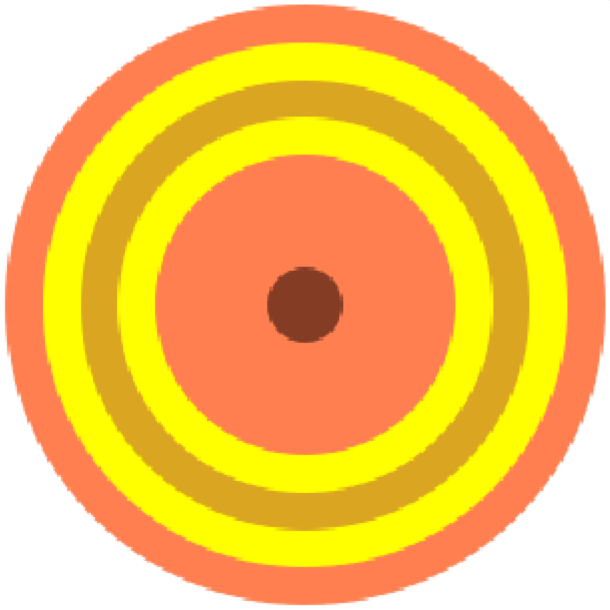
```
>
```

CCS Demo 2

```
> ( ccs 120 15 '( brown coral goldenrod yellow olive tan ) )
```



```
> ( ccs 120 15 '( brown coral goldenrod yellow olive tan ) )
```



```
>
```

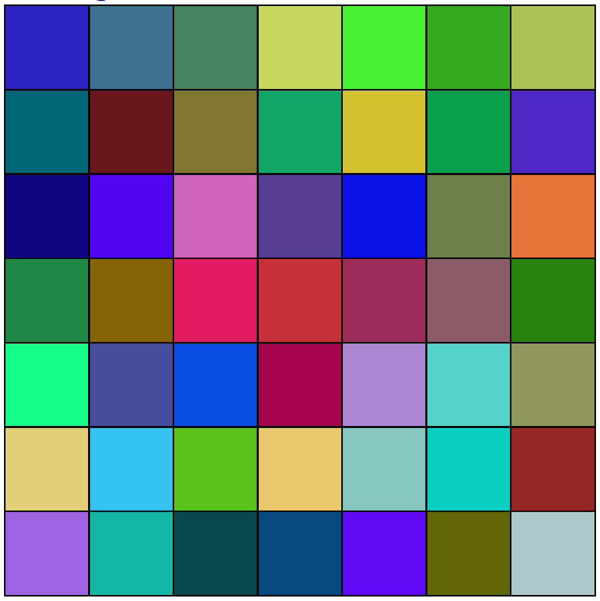
Code

```
( define ( random-color )
   ( color ( random 256 ) ( random 256 ) ( random 256 ) )
   )

( define (  ccr radius diff )
     ( cond ((<= radius 0 ) empty-image )
            ( else ( overlay  ( ccr ( - radius diff ) diff )
                     (  circle radius "solid" ( random-color ) )
                     )
                   )
            )
   )

( define ( cca radius difference color-1 color-2 )
   ( cond ( ( <= radius 0 ) empty-image )
          ( else ( overlay ( cca ( - radius difference ) difference color-2
   color-1 )
                           ( circle radius "solid" color-1 )
                           )
                 )
          )
   )

( define ( ccs radius difference color-list )
   ( cond ( ( <= radius 0 ) empty-image )
          ( else ( overlay ( ccs ( - radius difference ) difference color-list  )
                           ( circle radius "solid" ( list-ref color-list ( random
   ( length color-list ) ) ) )
                           )
                 )
          )
   )
```

Task 5: Variations on Hirst Dots

Random Colored Tile Demo

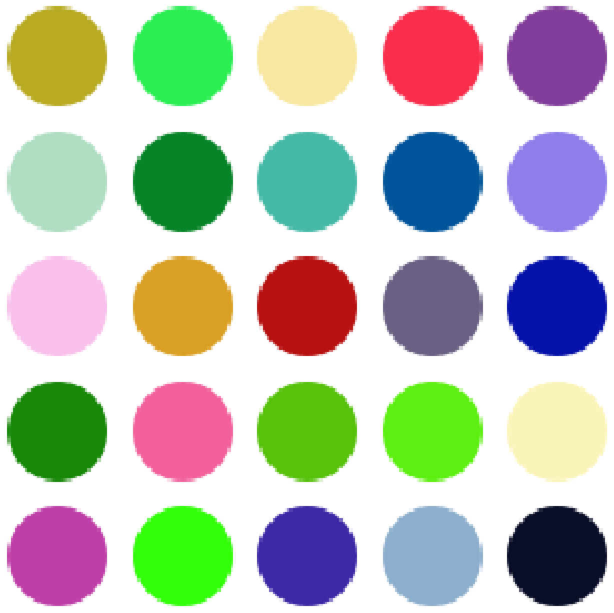
```
> ( square-of-tiles 7 random-color-tile )
```



```
>
```

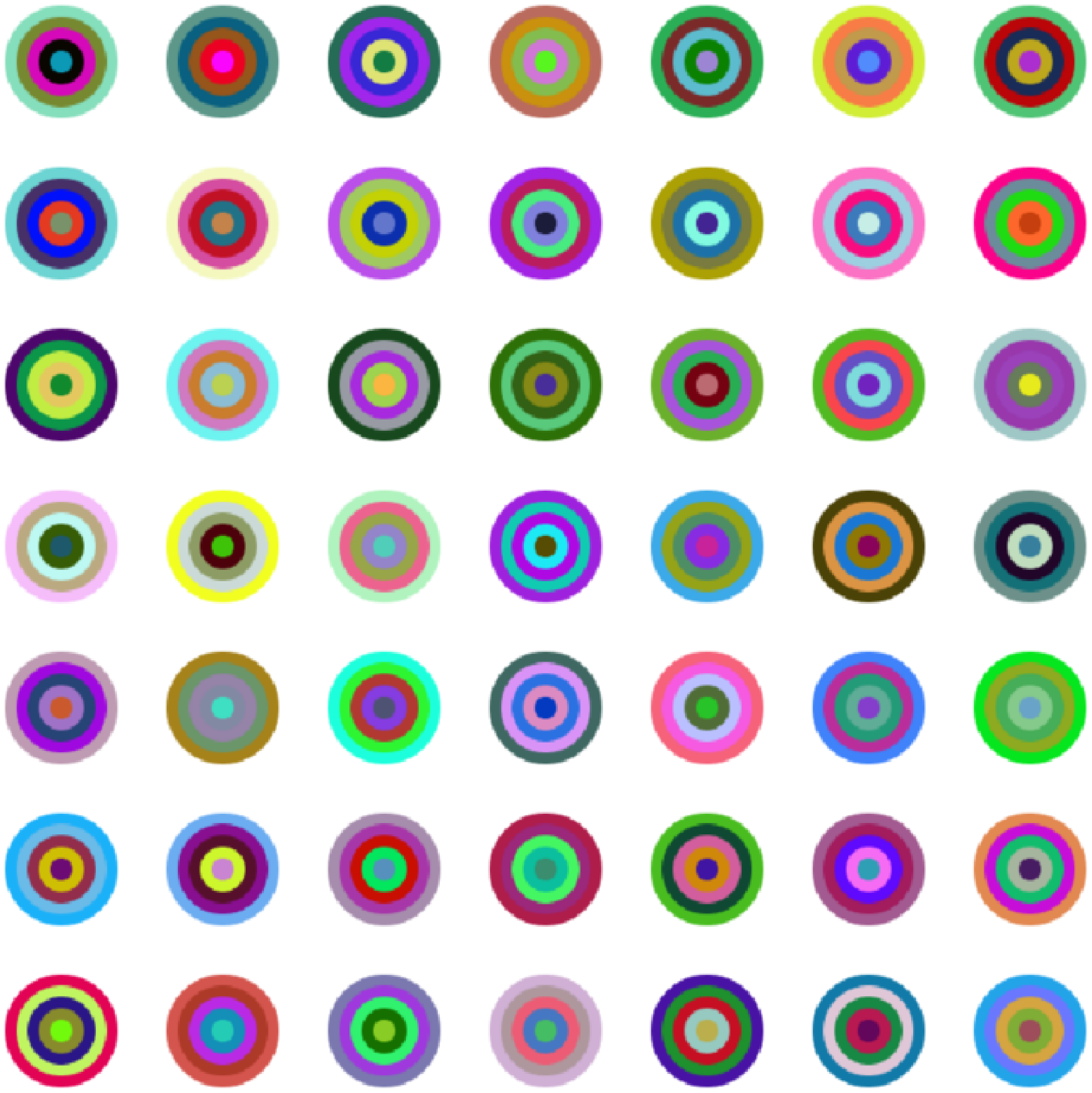
Hirst Dots Demo

```
> ( square-of-tiles 5 dot-tile )
```



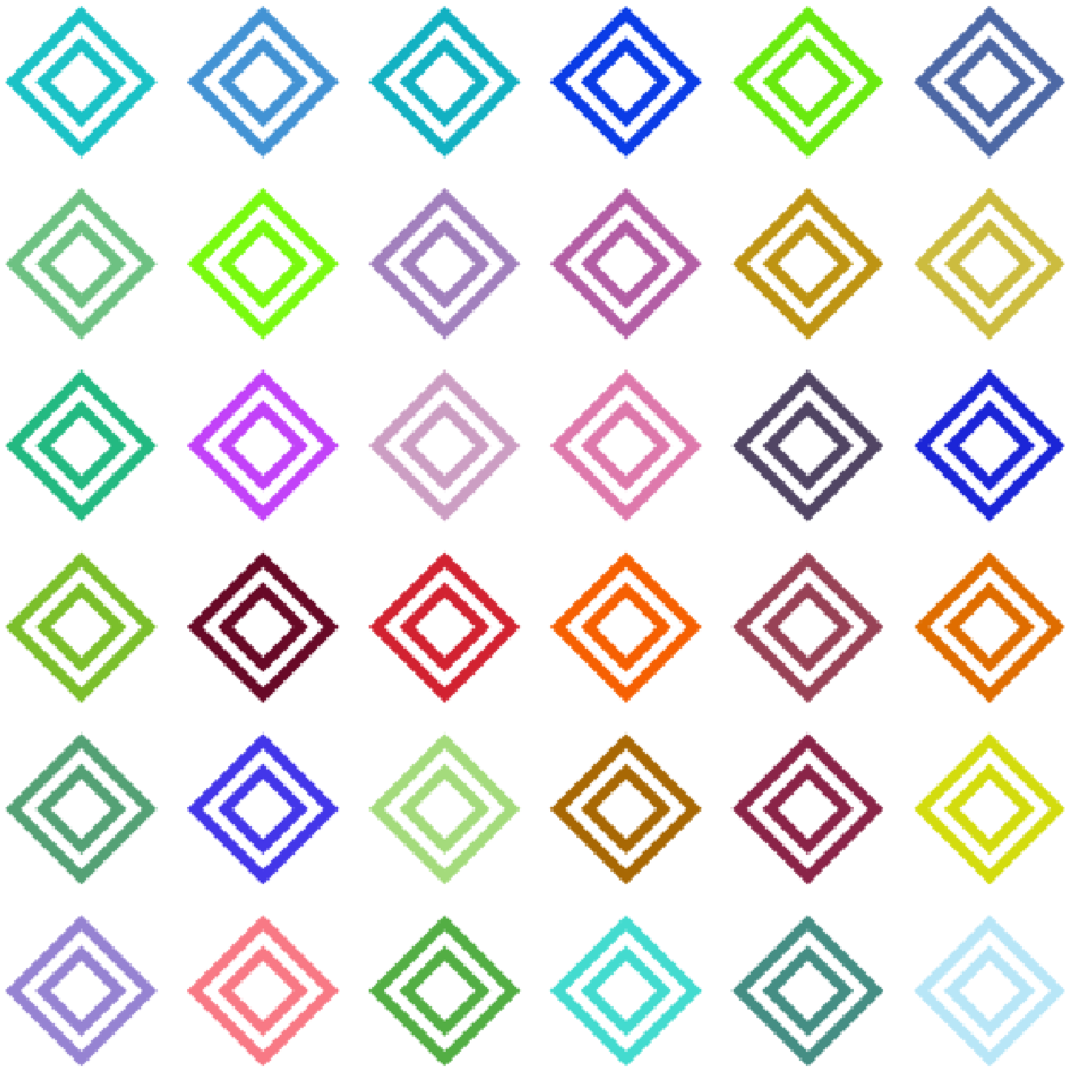
```
>
```

```
> ( square-of-tiles 7 ccs-tile )
```



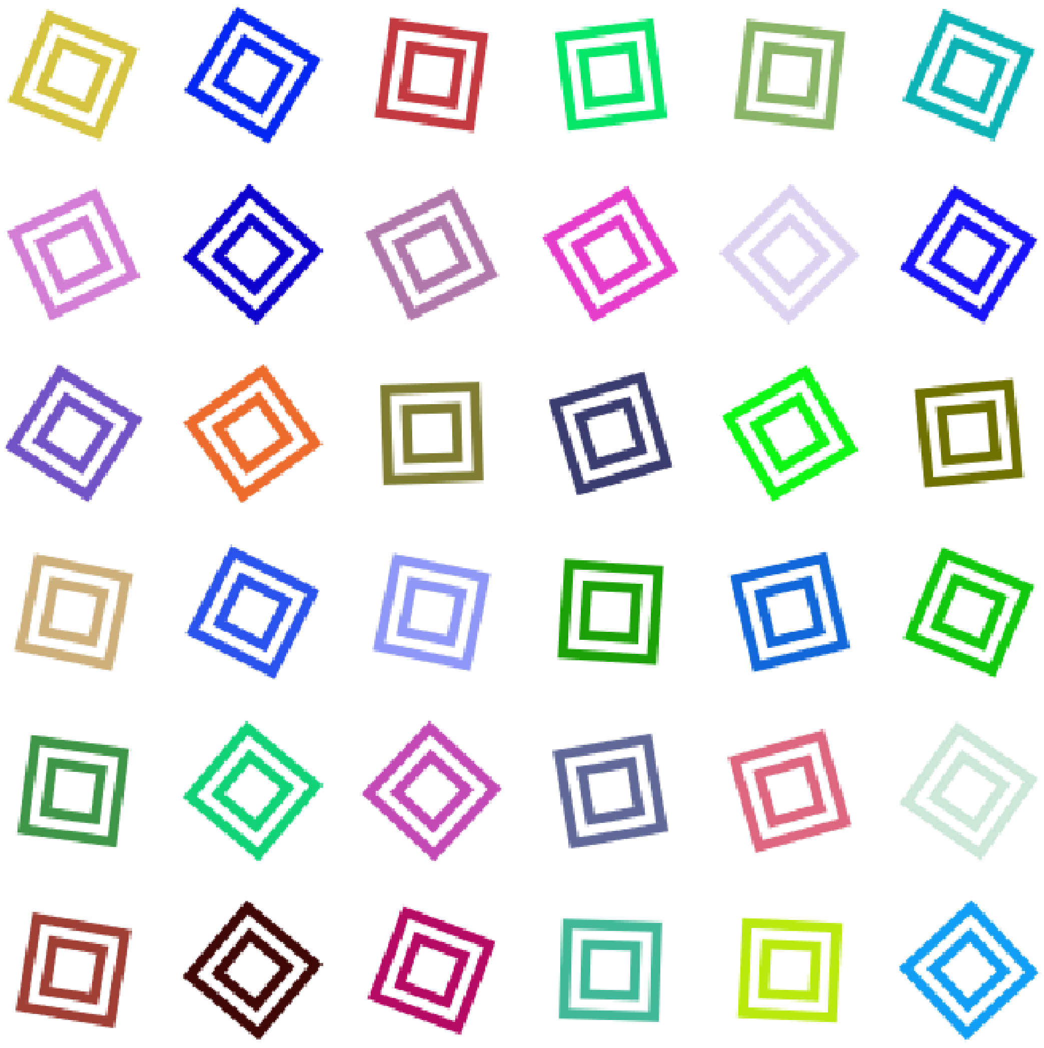
Nested Diamonds Demo

```
> ( square-of-tiles 6 diamond-tile )
```



```
>
```

```
> (square-of-tiles 6 wild-square-tile )
```



Code

```
( define ( random-color-tile )
   ( overlay
     ( square 40 "outline" "black" )
     ( square 40 "solid" ( random-color ) )
     )
   )

( define ( row-of-tiles n tile )
   ( cond
      ( ( = n 0 ) empty-image )
      ( ( > n 0 ) ( beside
                    ( row-of-tiles (- n 1 ) tile )
                    ( tile )
                    )
                  )
      )
   )

( define ( rectangle-of-tiles r c tile )
   ( cond
      ( ( = r 0 ) empty-image )
      ( ( > r 0 ) ( above
                    ( rectangle-of-tiles ( - r 1 ) c tile )
                    ( row-of-tiles c tile )
                    )
                  )
      )
   )

( define ( square-of-tiles n tile )
   ( rectangle-of-tiles n n tile  )
   )

( define ( dot-tile )
   ( overlay
     ( circle 20 "solid" (random-color) )
     ( square 50 0 "white " )
   )
   )

( define ( ccs-tile )
   ( define ( ccs-1 i j )
      ( cond ( ( = ( - i j ) 0 ) ( circle i "solid" ( random-color ) ) )
             ( else
               ( overlay ( ccs-1 ( - i j ) j )
                         ( circle i "solid" ( random-color ) ) )
               )
             )
      )
   ( overlay
     ( circle 50 "outline" "white" )
     ( ccs-1 35 7 )
     )
   )

( define ( diamond-tile )
   ( define clr ( random-color))
   ( overlay
        ( rotate 45 ( square 20 "solid" "white" ) )
        ( rotate 45 ( square 30 "solid" clr ) )
        ( rotate 45 ( square 40 "solid" "white" ) )
        ( rotate 45 ( square 50 "solid" clr ) )
        ( rotate 45 ( square 60 "solid" "white" ) )
        )
   )

( define ( wild-square-tile )
   ( define clr ( random-color))
   ( define rotation-a ( random 90 ) )
   ( overlay
        ( rotate rotation-a ( square 20 "solid" "white" ) )
        ( rotate rotation-a ( square 30 "solid" clr ) )
        ( rotate rotation-a ( square 40 "solid" "white" ) )
        ( rotate rotation-a ( square 50 "solid" clr ) )
        ( square 90 "solid" "white" )
        )
   )
```