Racket Assignment #3: Recursions in Racket

Learning Abstract:

This assignment helps us to use the recursive method using simple codes. This also helps to use the recursive method to create square of diamonds, wild squares, circles and many more which helps to know many concepts and ways to solve it.

Task 1: Counting Down, Counting Up

Code

```
(define (count-down n)
(cond
  ((zero? n)(display "\n"))
  (else
   (display n)(display "\n")(count-down (- n 1 )))
      )
)


(define (count-up n)
 (cond
  ((eq? n 1)(display 1))
  (else (count-up(- n 1))(display "\n")(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 (stars n)
  (cond
    ((zero? n)(display ""))
    (else (display "* ")(stars (- n 1)))
    )
  )

(define (triangle-of-stars n)
  (cond
    ((zero? n)(display ""))
    (else
    (triangle-of-stars (- n 1))(stars n)(display "\n")
      )
  )
  )
```

Demo

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

```
> (triangle-of-stars 0)
> (triangle-of-stars 10)
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```

Task 3: Flipping a coin

Code

```
(define(flip-for-difference n)
  (define (test h t)
  (let ((result (if (zero? (random 2)) 'h 't)))
    (cond
      ((= (abs (- h t)) n)(void))
      (else (display result)(test (+ h (if(eq? result 'h) 1 0))
            (+ t (if(eq? result 't) 1 0)))))))
(test 0 0)
  )
```

Demo

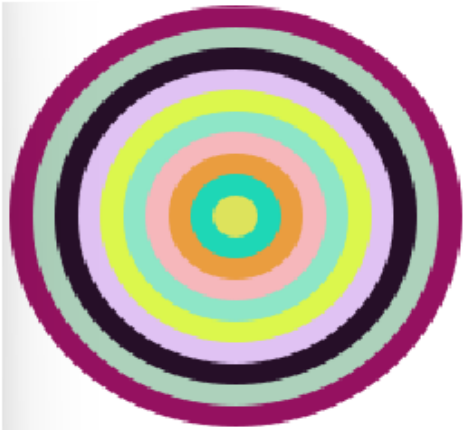
```
> (flip-for-difference 1)
t
> (flip-for-difference 1)
t
> (flip-for-difference 1)
t
```

> (flip-for-difference 1)
h
> (flip-for-difference 2)
hthhttthtt
> (flip-for-difference 2)
hthh
> (flip-for-difference 2)
thththththhhttt
> (flip-for-difference 2)
hh
> (flip-for-difference 2)
tt
> (flip-for-difference 2)
hthhthtthhttt
> (flip-for-difference 3)
hhttthhhttthtthtt
> (flip-for-difference 3)
ttt
> (flip-for-difference 3)
thhthhttthttt
> (flip-for-difference 3)
htththhhh
> (flip-for-difference 3)
thhthhttttt
> (flip-for-difference 3)
hthhh
> (flip-for-difference 3)
hthhthhtthhthtthhthtthhhthhhhthh
> (flip-for-difference 3)
htttt

Task 4: Laying Down Colorful Concentric Disks

CCR Demo

(ccr 100 10)

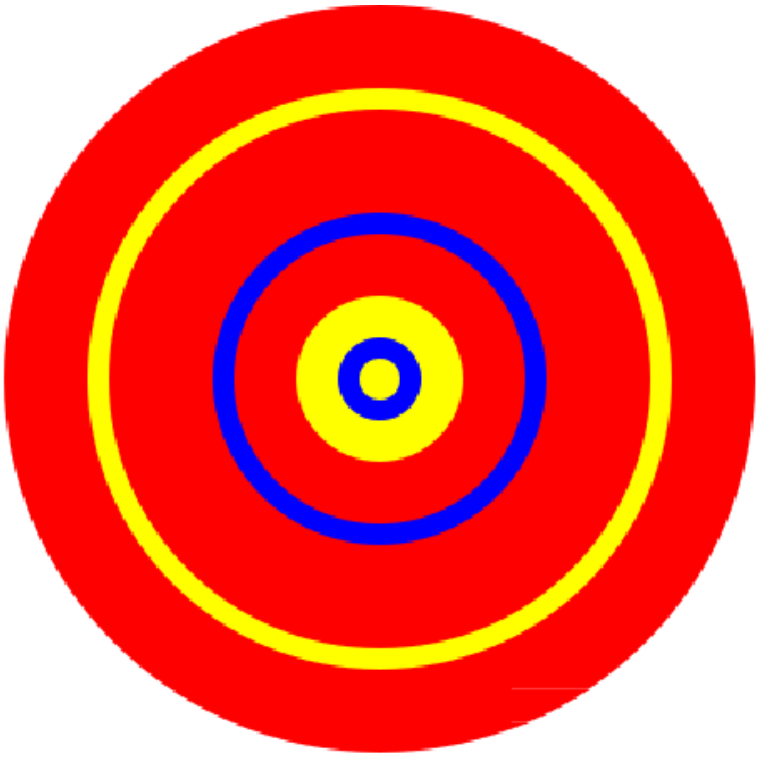


CCA Demo

(cca 120 20 ‘red ‘yellow)

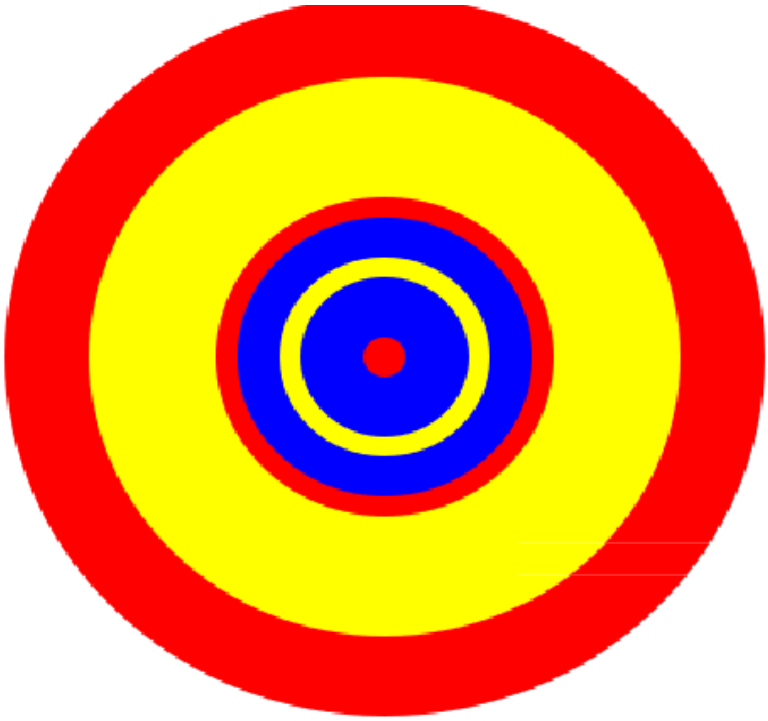
CCS Demo 1

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



CCS Demo 2

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



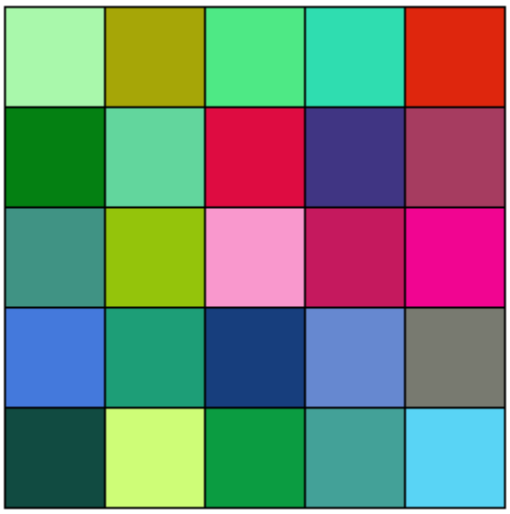
Code

```
; ccr
(define (ccr a b)
  (define (c)
  (make-color (random 256) (random 256) (random 256))
  )
  (define cr(circle a "solid" (c)))
  (cond
    ((zero? (- a b)) cr)
    (else (overlay (ccr (- a b) b) cr))
    )
  )

;cca
(define (cca n1 n2 c1 c2)
  (define cr(circle n1 "solid" c1))
  (cond
    ((zero? (- n1 n2)) cr)
    (else (overlay (cca (- n1 n2) n2 c2 c1) cr))
    )
  )

;ccs
(define (ccs n1 n2 l)
  (define (c)
    (list-ref l (random (length l)))
    )
  (define cr(circle n1 "solid" (c)))
  (cond
    ((zero? (- n1 n2)) cr)
    (else (overlay (ccs (- n1 n2) n2 l) cr))
    )
  )
```

Task 5: Variations on Hirst Dots

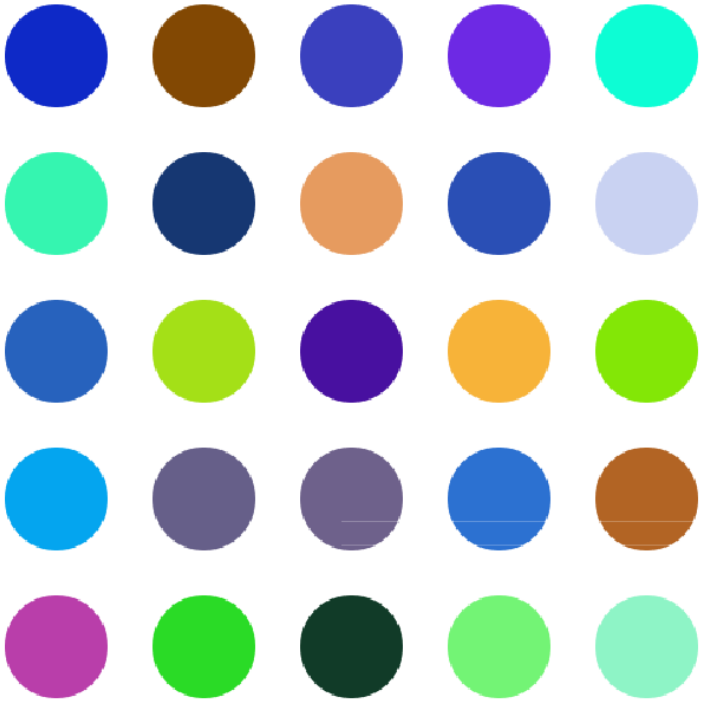
Random Colored Tile Demo

> (square-of-tiles 5 random-color-tile)



Hirst Dots Demo

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



CCS Dots Demo

> (square-of-tiles 6 ccs-tile)

Nested Diamonds Demo

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

Unruly Squares Demo

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

Code

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

;making a rectangle
(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))
    )
   )
  )

;making a square
(define (square-of-tiles n tile)
  (rectangle-of-tiles n n tile))

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

;making random color
(define (random-color)
  (make-color (random 256) (random 256) (random 256))
  )

;making dot tile
(define (dot-tile)
  (overlay
   (circle 50 "outline" "white")
   (circle 35 "solid" (random-color))
   )
  )

;making ccs dots
(define (ccs-tile)
   (define (ccs n1 n2)
     (cond
       ((= (- n1 n2) 0)(circle n1 "solid" (random-color)))
       (else
        (overlay (ccs (- n1 n2) n2) (circle n1 "solid" (random-color)))
       )
       )
```

```
    )
  (overlay
   (circle 50 "outline" "white")
  (ccs 35 7)
   )
  )

(define (diamond-tile)
  (define r1(random-color))
   (define (diamonds n1 n2 r1 r2)
     (cond
       ((= (- n1 n2) 5)(rotate 45 (square n1 "solid" "white") ))
       (else (overlay
              (diamonds (- n1 n2) n2 r2 r1)
              (rotate 45 (square n1 "solid" r1))
              ))
       )
     )
  (diamonds 30 5 "white" r1)
  )

(define (wild-square-tile)
  (define r3(random 90))
  (define r1(random-color))
   (define (diamonds n1 n2 r1 r2 r3)
     (cond
       ((= (- n1 n2) 5)(rotate r3 (square n1 "solid" "white") ))
       (else (overlay
              (diamonds (- n1 n2) n2 r2 r1 r3)
              (rotate r3 (square n1 "solid" r1))
              (square 50 "outline" "white")
              ))
       )
     )
  (diamonds 30 5 "white" r1 r3)
  )
```