Title: Racket Assignment #3: Recursion in Racket

Abstract:

In this assignment I wrote my first non-first Racket programs for this semester.

Task 1:

Code:

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

(define (count-up n)
  (count-upp 1 n)
  )

(define (count-upp nn n)
  (cond
    ((> nn n)(display " "))
    ((< nn (+ n 1))
     (display nn)
     (display "\n")
     (count-upp (+ nn 1) 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:

Code:

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

(define (triangle-of-stars n)
  (tos 1 n)
  )

(define (tos nn n)
  (cond
    ((> nn n)(display " "))
    ((< nn (+ n 1))
     (row-of-stars nn)
     (tos (+ nn 1) n)
           )
       )
  )
```

Demo:

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

> (triangle-of-stars 7)
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *

> (triangle-of-stars 14)
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
```

Task 3:

Note: I took the liberty of adding stars between t's and h's, purely for the sake of readability. To get rid of the stars, simply delete *(display "*")* from the code

Code:

```
(define (flip)
  (define x (random 0 2))
  (cond
    ((= x 0)"t")
    ((= x 1)"h")
       )
  )

(define (flip-for-difference n)
  (flip-for-dif 0 n (* n -1)))

(define (flip-for-dif dif n nn)

  (cond
    ((and (> dif nn) (< dif n))
       (define r (flip))
       (display "*")(display r)
     (cond
       ((eq? r "t")
        (define new-dif (- dif 1))
        (flip-for-dif new-dif n nn)
        )
       ((eq? r "h")
        (define new-dif (+ dif 1))
        (flip-for-dif new-dif n nn)
        )
      )

     )
    )
  )
```

Demo:

```
> (flip-for-difference 1)
*t
> (flip-for-difference 1)
*t
> (flip-for-difference 1)
*h
> (flip-for-difference 1)
*h
> (flip-for-difference 2)
*t*t
> (flip-for-difference 2)
*t*t
> (flip-for-difference 2)
*t*h*h*t*t*h*h*t*t*h*t*t
> (flip-for-difference 2)
*t*h*t*t
> (flip-for-difference 2)
*h*t*h*h
> (flip-for-difference 2)
*t*h*h*h
```

```
> (flip-for-difference 3)
*t*h*t*h*t*h*h*t*h*h*t*t*t*t*t
> (flip-for-difference 3)
*h*h*t*t*t*t*h*h*h*h*t*t*h*t*t*t*t*h*h*t*h*t*h*t*h*t*h*h*t*t*h*h*h*t*h*t*h*h
> (flip-for-difference 3)
*h*t*t*t*h*h*t*t*t
> (flip-for-difference 3)
*t*h*t*t*t
> (flip-for-difference 3)
*h*t*h*h*h
> (flip-for-difference 3)
*h*h*t*h*h
> (flip-for-difference 4)
*h*t*h*h*h*t*h*h
> (flip-for-difference 4)
*h*t*t*h*h*h*t*t*h*h*t*t*h*h*t*h
> (flip-for-difference 4)
*t*h*t*h*h*h*h*h
> (flip-for-difference 4)
*h*t*h*h*h*t*h*h
> (flip-for-difference 4)
*h*t*h*h*t*t*t*t*t*h*t*t*h*h*t*t*t*h*h*h*h*h*h*t*t*t*h*h*t*t*t*h*t*t*t*h*h*t*t*t*t*t*h*t*t
> (flip-for-difference 4)
*h*t*t*h*t*t*t*t
> (flip-for-difference 4)
*h*h*h*t*h*h
> (flip-for-difference 4)
*t*t*h*t*t*t
>
```

Task 4:

Code:

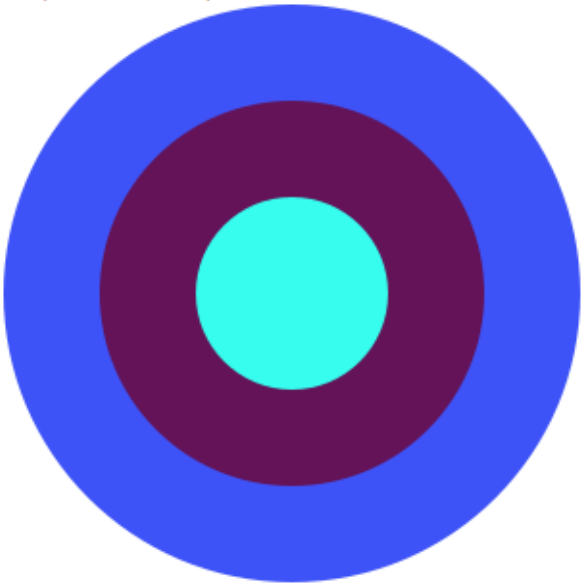
```
(define (ccr n d)
  (define c (circle n "solid" (rc)))
  (ccrr c n d)
  )

(define (ccrr c n d)
  (define nn (- n d))
  (cond
    ((= n 0)
     (display c)
           )
    ((> n 0)
     (define newc (circle nn "solid" (rc)))
     (define combo (overlay newc c))
     (ccrr combo nn d)
           )
       )
  )

(define (cca n d color1 color2)
  (define c (circle n "solid" color1))
  (ccaa c n d color2 color1)
  )
```

```
(define (ccaa c n d color1 color2)
  (define nn (- n d))
  (cond
    ((= n 0)
     (display c)
          )
    ((> n 0)
     (define newc (circle nn "solid" color1))
     (define combo (overlay newc c))
     (ccaa combo nn d color2 color1)
          )
       )
  )

(define (ccs n d colors)
  (define len (length colors))
  (define nn (- n d))
  (define c (circle n "solid" (list-ref colors (random 0 len))))
  (ccss c len nn d colors)
  )

(define (ccss c len nn d colors)

  (cond
    ((or (> nn 0) (= nn 0)) c )
    ((> nn 0)
     (define cc (circle nn "solid" (list-ref colors (random 0 len))))
     (define combo (overlay cc c))
     (define nnn (- nn d))
     (ccss combo len nnn d colors)
          )
       )
  )
```

Demo: (next page)

```
(ccr 100 50)
```

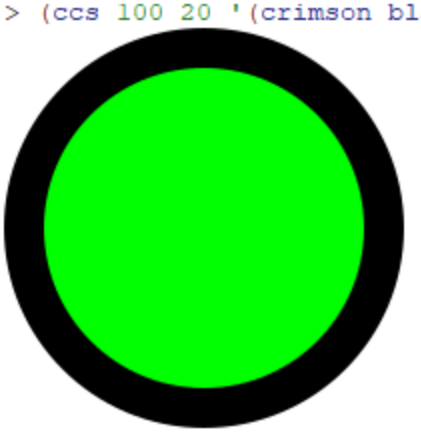
```
> (ccr 150 50)
```



```
> (cca 150 25 "dark blue" "gold")
```

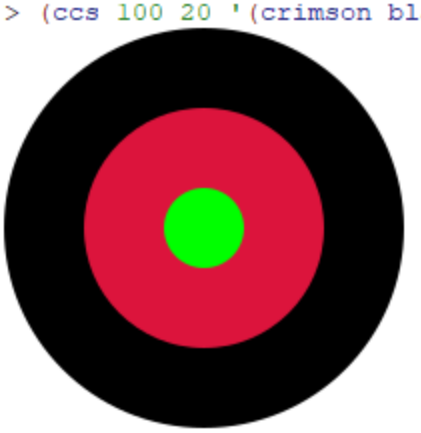
```
> (cca 120 20 "orange" "gold")
```

```
> (ccs 100 20 '(crimson black green))
```



```
> (ccs 100 20 '(crimson black green))
```

```
> (ccs 100 20 '(crimson black green))
```



```
> (ccs 100 20 '(crimson black green))
```

```
> (ccs 80 10 '(plum purple navy))
```

```
> (ccs 80 10 '(plum purple navy))
```

```
> (ccs 80 4 '(firebrick black green gold))
```

```
> (ccs 80 4 '(firebrick black green gold))
```

Task 5:

Code:

```
#lang racket

(require 2htdp/image)

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

(define (types-of-tile)
  (display "standard-tile ")
  (display "random-color-tile ")
  (display "random-red-tile ")
  (display "random-green-tile ")
  (display "random-yellow-tile ")
  (display "random-cyan-tile ")
  (display "random-magenta-tile\n")
 )

(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 (standard-tile) (square 40 "outline" "black"))

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

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

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

(define (random-green-tile)
  (overlay
```

```
   (square 40 "outline" "black")(square 40 "solid" (random-green-color))
   )
  )

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

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

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



(define (dot-tile)
  (overlay
   (circle 35 "solid" (rc))(square 100 "solid" "white")
   )
  )

(define (random-blue-dot-tile)
  (overlay
   (circle 35 "solid" (random-blue-color))(square 100 "solid" "white")
   )
  )
```

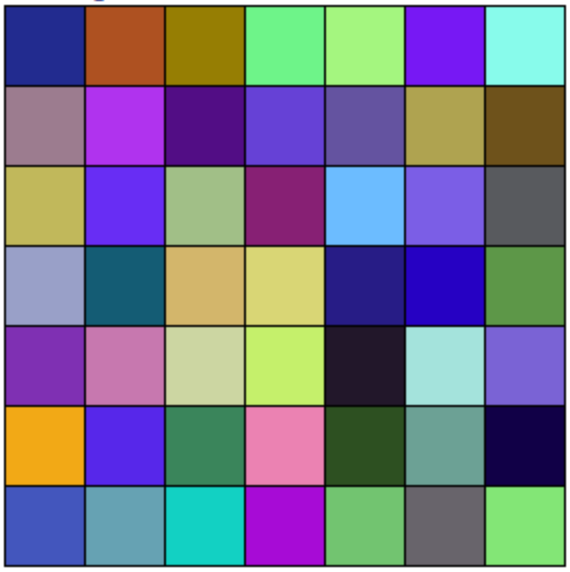
```
(define (random-red-dot-tile)
  (overlay
   (circle 35 "solid" (random-red-color))(square 100 "solid" "white")
   )
  )

(define (random-green-dot-tile)
  (overlay
   (circle 35 "solid" (random-green-color))(square 100 "solid" "white")
   )
  )

(define (random-yellow-dot-tile)
  (overlay
   (circle 35 "solid" (random-yellow-color))(square 100 "solid" "white")
   )
  )

(define (random-magenta-dot-tile)
  (overlay
   (circle 35 "solid" (random-magenta-color))(square 100 "solid" "white")
   )
  )

(define (random-cyan-dot-tile)
  (overlay
   (circle 35 "solid" (random-cyan-color))(square 100 "solid" "white")
   )
  )


(define (random-blue-color) (color 0 0 255 (dark-rgb-value)))
(define (random-red-color) (color 255 0 0 (dark-rgb-value)))
(define (random-green-color) (color 0 255 0 (dark-rgb-value)))
```

```
(define (random-yellow-color) (color 255 255 0 (dark-rgb-value)))
(define (random-magenta-color) (color 255 0 255 (dark-rgb-value)))
(define (random-cyan-color) (color 0 255 255 (dark-rgb-value)))

(define (dark-rgb-value) (+ (random 128) 128))



(define (ccs-tile)
  (define a (random-color))(define b (random-color))(define c
(random-color))
  (define x (list a b c))
  (define circ (ccs 35 7 x))
  (overlay circ (square 100 "solid" "white"))
  )

(define (ccs n d colors)
  (define len (length colors))
  (define nn (- n d))
  (define c (circle n "solid" (list-ref colors (random 0 len))))
  (ccss c len nn d colors)
  )

(define (ccss c len nn d colors)

  (cond
    ((or (< nn 0) (= nn 0)) c)
    ((> nn 0)
     (define cc (circle nn "solid" (list-ref colors (random 0 len))))
     (define combo (overlay cc c))
     (define nnn (- nn d))
     (ccss combo len nnn d colors)
          )
       )
  )
```

```
(define (diamond-tile)

  (define a (random-color))
  (define back (square 100 "solid" "white"))
  (define big (square 70 "solid" a))
  (define mid (square 60 "solid" "white"))
  (define smol (square 50 "solid" a))
  (define tiny (square 40 "solid" "white"))
  (rotate 45 (overlay tiny smol mid big back))

  )

(define (wild-square-tile)

  (define a (random-color))
  (define back (square 100 "solid" "white"))
  (define big (square 70 "solid" a))
  (define mid (square 60 "solid" "white"))
  (define smol (square 50 "solid" a))
  (define tiny (square 40 "solid" "white"))
  (define angle (random 0 360))
  (rotate angle (overlay tiny smol mid big back))

  )
```

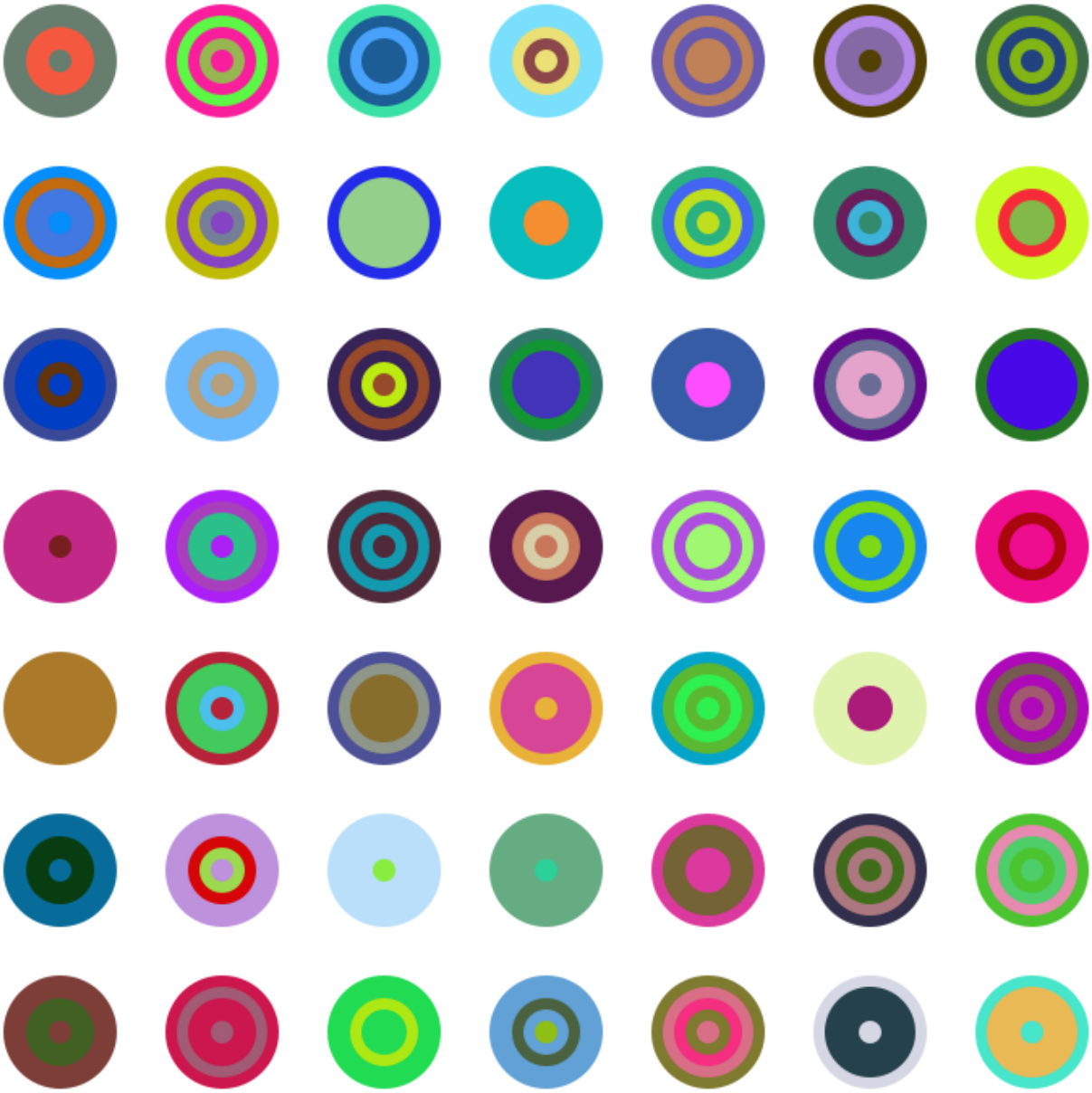
Demo:

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



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

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



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

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