Samuel Ghent

Racket Assignment #3: Recursion in Racket

Learning Abstract: Our goal this assignment is to display our knowledge and practice writing recursive functions in Racket.

Task 1: Counting Down, Counting Up

Code:

```
#lang racket

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

( define ( count-up n )
   (cond
     ( ( > n 0 )
       ( define ( count-up-compare i )
          (cond
            (( <= i n )
             ( display i ) ( display "\n" )
             ( count-up-compare (+ i 1 )

                                )
             )
            )
          )
       ( count-up-compare 1 )
       )
     )
   )
```

Demo:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( 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 n)
  (cond
    ((= n 0)
     (display "")
     )
    ((> n 0)
     (triangle-of-stars (- n 1))
     (row-of-stars n)
     )
    )
  )
```

Demo:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( triangle-of-stars 5 )
*
**
***
****
*****
> ( triangle-of-stars 0 )
> ( triangle-of-stars 15 )
*
**
***
****
*****
******
*******
********
*********
**********
***********
************
*************
**************
***************
> |
```

Task 3: Flipping a Coin

Code:

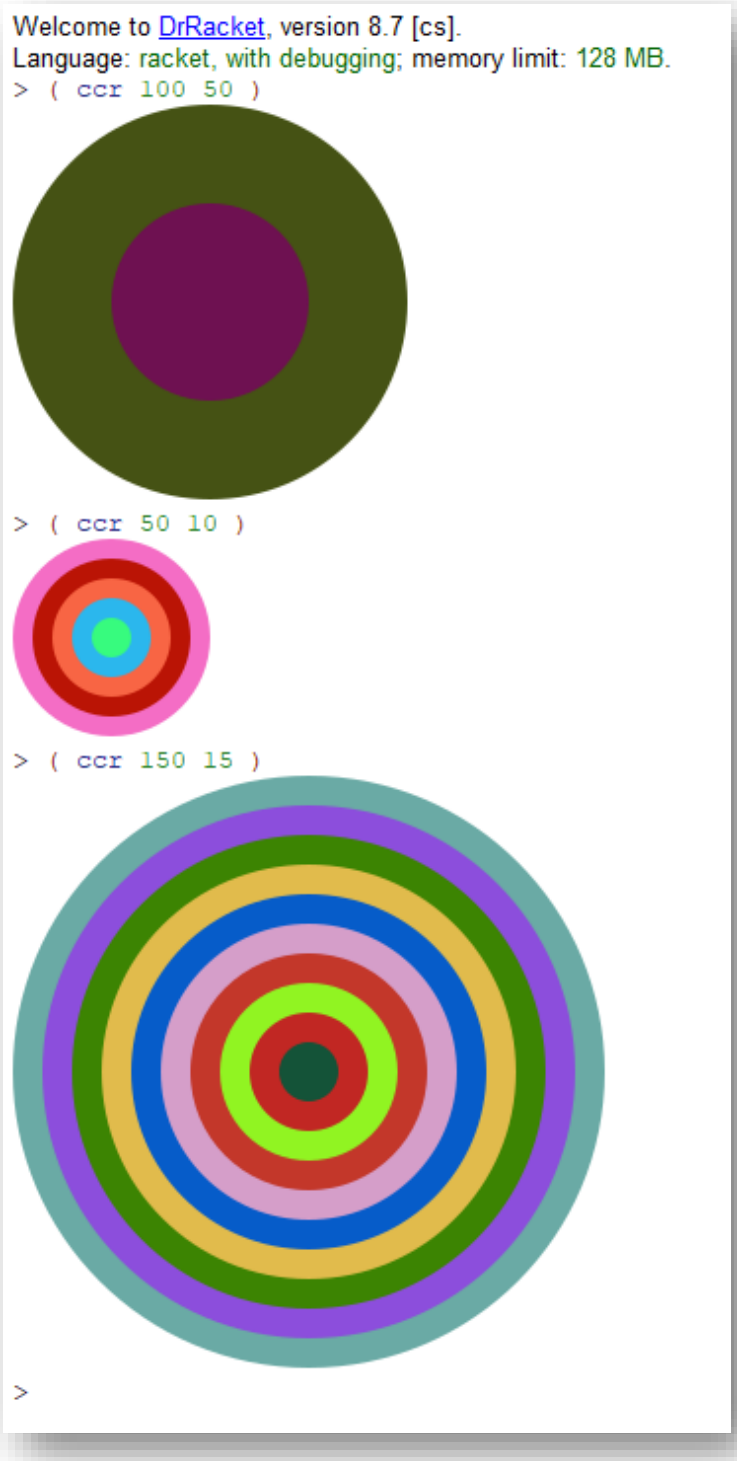
```
(define (flip-for-difference n)
  ( define ( flip-coin )
     ( define result ( random 2 ) )
     ( cond
        ( ( = result 1 ) 'h )
        ( ( = result 0 ) 't ) ) )
  ( define ( flip-helper heads tails )
     ( let ( [ flips ( flip-coin ) ] )
       ( display ( if ( eq? flips 'h ) "h " "t " ) )
       ( if ( = ( abs ( - ( + ( if ( eq? flips 'h ) 1 0 ) heads )
                          ( + ( if ( eq? flips 't ) 1 0 ) tails )
                          )
                       )
                 n )
          ( displayln "" )
          ( flip-helper ( + ( if ( eq? flips 'h ) 1 0 ) heads )
                        ( + ( if ( eq? flips 't ) 1 0 ) tails )
                        )
          )
       )
     )
  ( flip-helper 0 0 )
  )
```

Demo:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
h
> ( flip-for-difference 2 )
t t
> ( flip-for-difference 2 )
h t t t
> ( flip-for-difference 2 )
t t
> ( flip-for-difference 2 )
t h h h
> ( flip-for-difference 2 )
t h h t h t h t t t
> ( flip-for-difference 2 )
h h
> ( flip-for-difference 3 )
t h t h h h t t t t t
> ( flip-for-difference 3 )
h h h
> ( flip-for-difference 3 )
h h t h h
> ( flip-for-difference 3 )
t h h h h
> ( flip-for-difference 3 )
h t h h t h h
> ( flip-for-difference 3 )
t h t h h h t t h t h h h
> ( flip-for-difference 4 )
t t t h t t
> ( flip-for-difference 4 )
h h t t t t t t
> ( flip-for-difference 4 )
h t h h h h
> ( flip-for-difference 4 )
h h h t t t h t h h h h
> ( flip-for-difference 4 )
t t t t
> ( flip-for-difference 4 )
t h t t t t
> ( flip-for-difference 4 )
t h h h t h t h t h t h t h t h t t t t t h h t t t
> ( flip-for-difference 4 )
t t h h h t h h h t t h t h h t h t t h t t h t t h h h h h
>
```

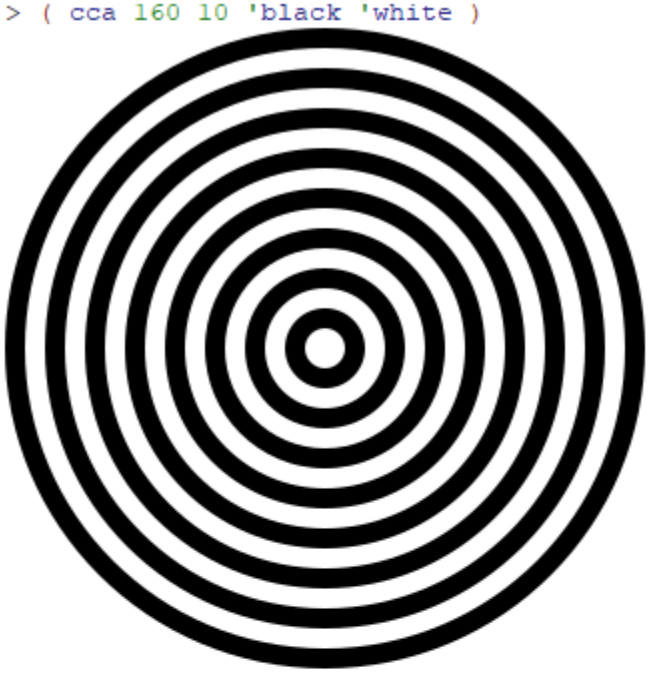
Task 4: Laying Down Colorful Concentric Disks

CCR Demo:



CCA Demo:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( cca 160 10 'black 'white )
```

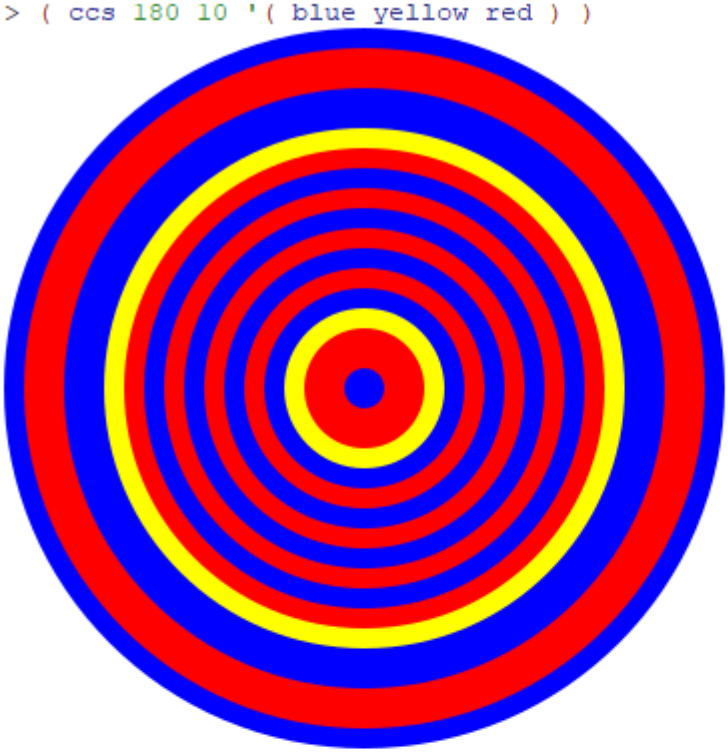


```
> ( cca 150 25 'red 'orange )
```

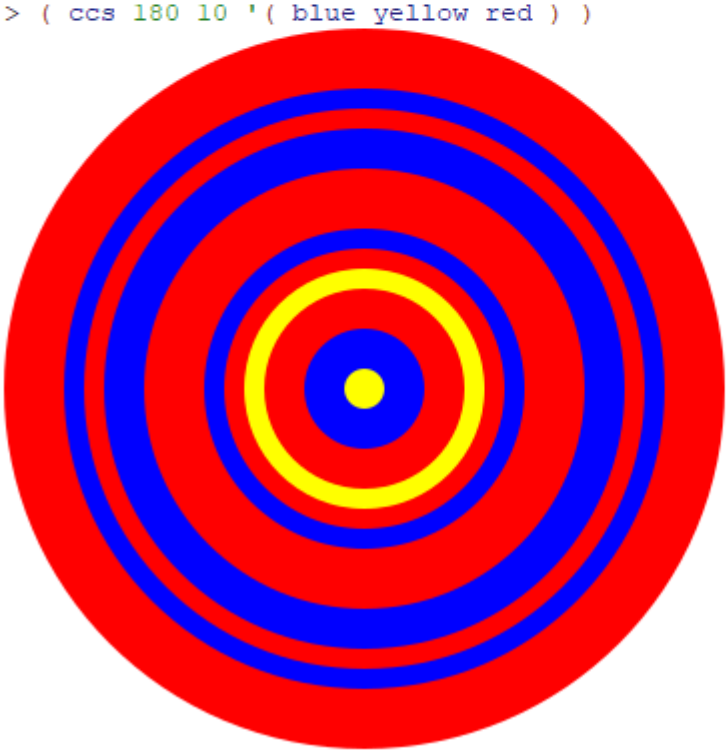
```
>
```

CCS 1 Demo:

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( ccs 180 10 '( blue yellow red ) )
```



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



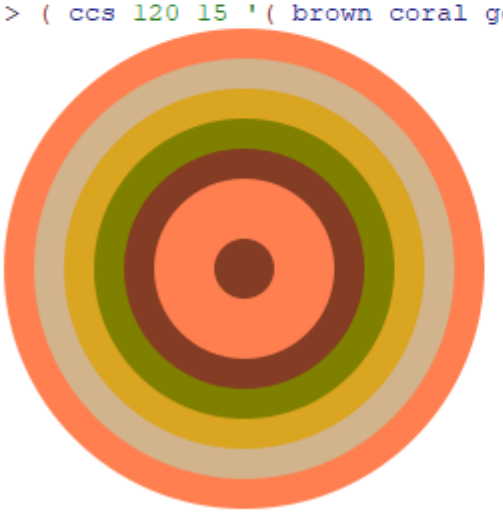
```
>
```

CCS 2 Demo

Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.

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

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



```
>
```

Code:

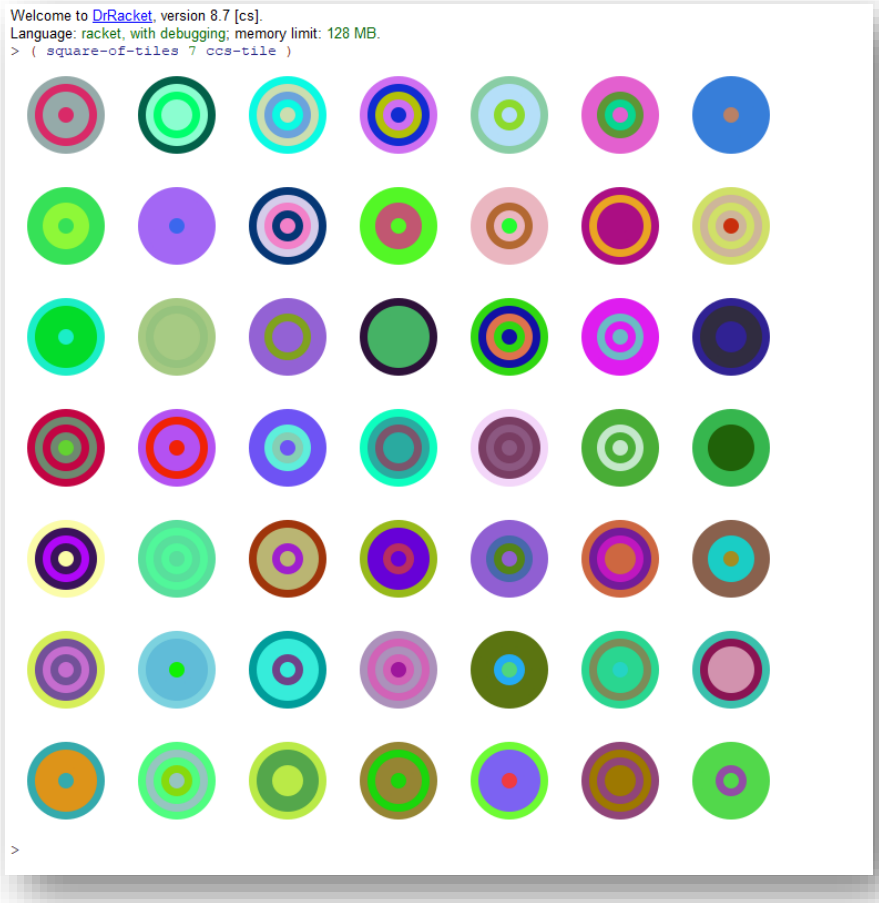
```
( require 2htdp/image )
( define ( ccr radius difference )
   ( cond (( > radius 0 )
           ( define ( rgb ) ( random 0 256 ) )
           ( define ( rc ) ( color ( rgb ) ( rgb ) ( rgb ) ) )
           ( overlay ( ccr ( - radius difference ) difference) ( circle radius 'solid ( rc ) ) )
           )
          ( ( = radius 0 ) empty-image )
          )
   )

( define ( cca radius difference c1 c2 )
   ( cond
      ( ( <= radius 0 ) empty-image )
      ( ( > radius 0)
        ( overlay ( cca ( - radius difference ) difference c1 c2 )
                  ( circle radius "solid"
                           ( if ( even? ( quotient radius difference ) ) c1 c2 )
                           )
                  )
        )
      )
)
( define ( ccs radius difference cList )
   ( cond
      ( ( <= radius 0 ) empty-image )
      ( ( > radius 0)
        ( let ( ( cItem ( list-ref cList ( random ( length
                                                     cList ) ) ) ) ) )
           ( overlay ( ccs ( - radius difference ) difference cList )
                     ( circle radius "solid" cItem)
                     )
           )
        )
      )
   )
```

Task 5: Variations on Hirst Dots

Solid Randomly Color Tiles with Borders

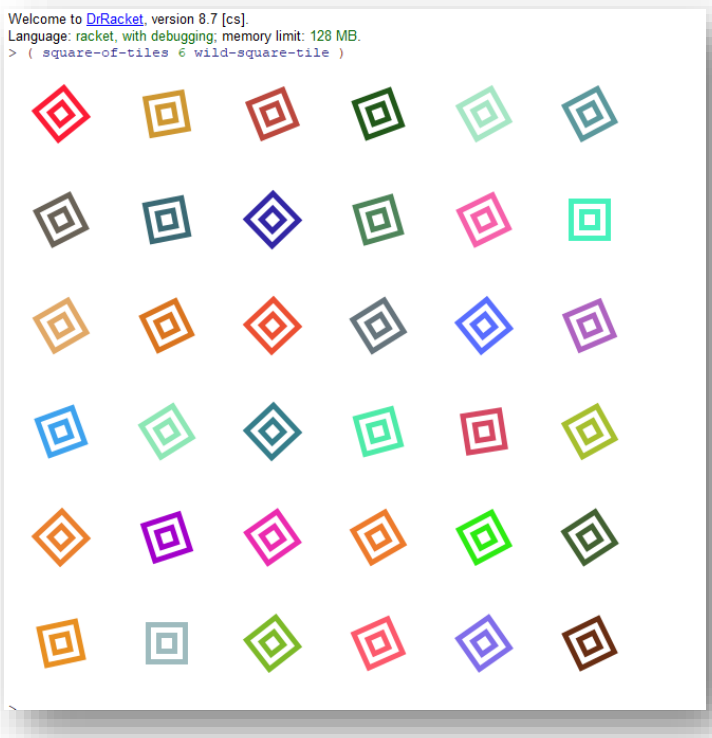
Hirst Dots:

CCS Dots:



Nested Diamonds:

Unruly Squares:



Code:

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

( define ( rc n )
   ( cond
      ( ( > n 0 )
        ( cons ( random-color ) ( rc ( - n 1 ) ) )
        )
      ( ( = n 0)
        empty
        )
      )
   )
( 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 35 "solid" (random-color ) )
     ( square 100 "solid" "white" )
     )
   )

( define ( ccs-tile )
   ( define color ( rc 3 ) )
   ( overlay
     ( ccs 35 7 color )
     ( square 100 "solid" "white" )
     )
   )

( define ( diamond-tile )
   ( define dColor ( random-color ) )
   ( overlay
     ( rotate 45 ( square 10 "solid" "white" ) )
     ( rotate 45 ( square 20 "solid" dColor ) )
     ( rotate 45 ( square 30 "solid" "white" ) )
     ( rotate 45 ( square 40 "solid" dColor ) )
     ( square 100 "solid" "white" )
     )
   )

( define (wild-square-tile )
   ( define sqColor ( random-color ) )
   ( define angle ( random 0 45 ) )
   ( overlay
     ( rotate angle ( square 10 "solid" "white" ) )
     ( rotate angle ( square 20 "solid" sqColor ) )
     ( rotate angle ( square 30 "solid" "white" ) )
     ( rotate angle ( square 40 "solid" sqColor ) )
     ( square 100 "solid" "white" )
     )
   )
```