Racket Assignment #3: Recursion in Racket

Written by **David Hennigan**

Learning Abstract

This assignment is composed of 5 programming tasks all utilizing recursion. The tasks were varying in difficulty with the assignment becoming more involved in the final tasks. During this assignment I was able to make my own recursive random color list generator which I enjoyed making quite thoroughly. The overarching goals of the assignment were to develop some important recursion practices and to further familiarize with 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 )
      ( 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 ( line-of-stars n )
  ( cond
    ( ( > n 0 )
      ( display "*" )
      ( line-of-stars ( - n 1 ) )
    )
  )
)

( define ( triangle-of-stars n )
  ( cond
    ( ( > n 0 )
      ( triangle-of-stars ( - n 1 ) )
      ( line-of-stars n )
      ( display "\n" )
    )
   )
)
```

Demo

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

Task 3: Flipping a Coin

Code

```
( define ( flip-coin )
  ( define rv ( random 2 ) )
  ( cond
    ( ( = rv 0 ) 'h )
    ( ( = rv 1 ) 't)
  )
```

```
)

( define ( flip-for-difference stopping-point )
  ( flip-for-difference-helper stopping-point ( * stopping-point 2 ) )
)

( define ( flip-for-difference-helper current-value stopping-point )
  ( cond (
       ( not ( or ( = current-value 0 ) ( = current-value stopping-point ) ) )
       ( define outcome ( flip-coin ) )
       ( display outcome ) ( display " " )
       ( cond (
            ( eq? outcome 'h )
            ( flip-for-difference-helper ( - current-value 1 ) stopping-point)
           )

           (
            ( eq? outcome 't )
            ( flip-for-difference-helper ( + current-value 1 ) stopping-point )
           )
        )
      )
  )
)
```

Demo

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

> (flip-for-difference 1)

t

> (flip-for-difference 1)

h

> (flip-for-difference 2)

t t

> (flip-for-difference 2)

t h t t

> (flip-for-difference 2)

h h

> (flip-for-difference 2)

h t h h

> (flip-for-difference 2)

h t t t

> (flip-for-difference 2)

t t

> (flip-for-difference 3)

h h h

> (flip-for-difference 3)

h h h

> (flip-for-difference 3)

h h h

> (flip-for-difference 3)

h h h

> (flip-for-difference 3)

t h h t t t t

> (flip-for-difference 3)

h t t h t t t

> (flip-for-difference 4)

t h h t h t h t t t t t

> (flip-for-difference 4)

t h t t t t

> (flip-for-difference 4)

t h h h h h

> (flip-for-difference 4)

t t t t

> (flip-for-difference 4)

h t t t t t

> (flip-for-difference 4)

t t t h h h t t t h t h h t h h t h h h t h h h

> (flip-for-difference 4)

t h h t t t t h h h t h h t t t h h t t t t h t t t

> (flip-for-difference 4)

t t t t

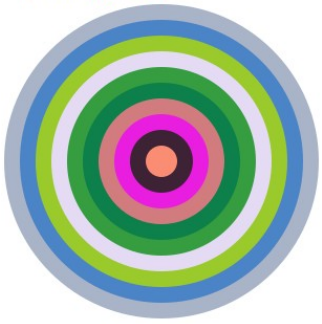
Task 4: Laying Down Colorful Concentric Disks

CCR Demo

> (ccr 100 50)

> (ccr 50 10)

> (ccr 150 15)

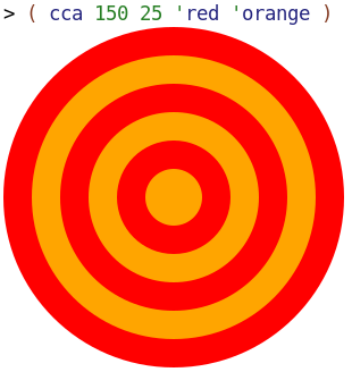


>

CCA Demo

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

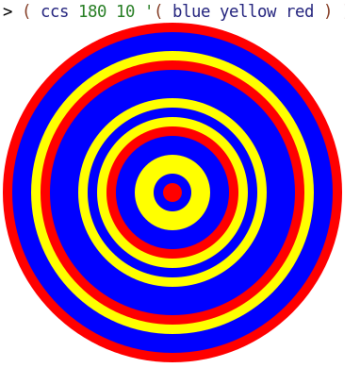
```
> ( cca 150 25 'red 'orange )
```



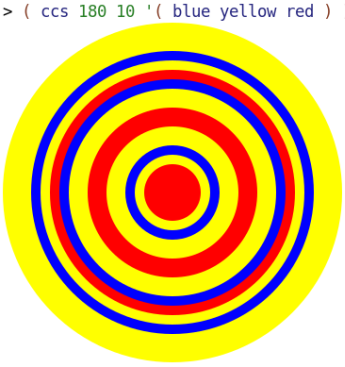
```
>
```

CCS Demo 1

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



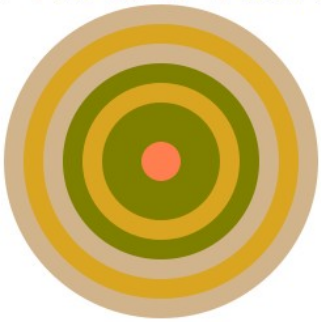
```
> ( ccs 180 10 '( blue yellow red ) )
```



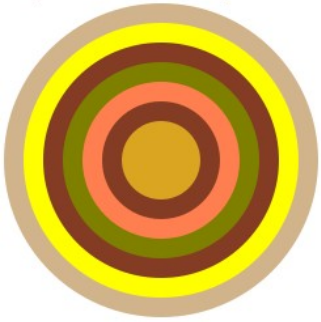
```
>
```

CCS Demo 2

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



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



```
>
```

Code

```
( define ( 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 color1 color2 )
  ( cca-helper radius difference color1 color2 1 )
)
```

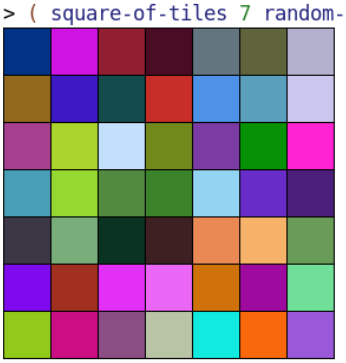
```
( define ( cca-helper radius difference color1 color2 current-color-num )
  ( cond ( ( > radius 0 )
          ( cond ( ( = current-color-num 1 )
                  ( overlay ( cca-helper ( - radius difference ) difference color1 color2 2 )
                                        ( circle radius 'solid color1 ) ) )
                 ( ( = current-color-num 2 )
                  ( overlay ( cca-helper ( - radius difference ) difference color1 color2 1 )
                                        ( circle radius 'solid color2 ) ) )
          )
          )
          ( ( = radius 0 ) empty-image )
  )
)

( define ( ccs radius difference colors )
  ( define numColors ( length colors ) )
  ( ccs-helper radius difference colors numColors )
)



( define ( ccs-helper radius difference colors numColors )
       ( cond (( > radius 0 )
         ( define ( colorNum ) ( random numColors ) )
         ( define color ( list-ref colors ( colorNum ) ) )
         ( overlay ( ccs-helper ( - radius difference ) difference colors numColors )
                                ( circle radius 'solid color ) )
        )
        ( ( = radius 0 ) empty-image )
       )
```

)

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 )
```

```
>
```

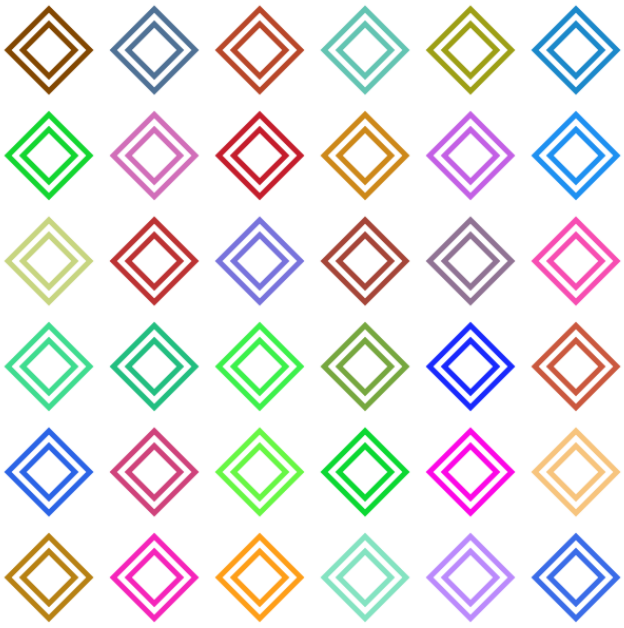
CCS Dots Demo

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

```
>
```

Nested Diamonds Demo

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



```
>
```

Unruly Squares Demo

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

```
>
```

Code

```
; Generate a rectangle of tiles of a specified row count, column count,
; and tile type

( 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 ) )
      )
  )
)

; Generate a square of tiles of a specified side length and tile type

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

; Generator for a randomly colored tile

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

; Generator for a random color

( define ( random-color )
  ( define ( rgb ) ( random 0 256 ) )
  ( color ( rgb ) ( rgb ) ( rgb ) )
)

; Generator for a randomly colored dot tile
```

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

; Generator for a ccs tile

( define ( ccs-tile )
   ( define colors ( random-colors 3 ) )
   ( overlay
     ( ccs 35 5 colors )
     ( square 100 "solid" "white" )
   )
)

; randomly generate a list of colors of size n

( define ( random-colors n )
   ( cond ( ( > n 0 )
            ( cons ( random-color ) ( random-colors ( - n 1 ) ) )
        )
        ( ( = n 0 ) empty )
    )
)

; generates a randomly colored diamond tile

( define ( diamond-tile )
```

```
  ( define diamondColor ( random-color ) )
  ( overlay ( rotate 45 ( square 30 "solid" "white" ) )
        ( rotate 45 ( square 40 "solid" diamondColor ) )
        ( rotate 45 ( square 50 "solid" "white" ) )
        ( rotate 45 ( square 60 "solid" diamondColor ) )
        ( square 100 "solid" "white" )
  )
)

; generates a randomly colored and angled square within a white tile

( define ( wild-square-tile )
  ( define squareColor ( random-color ) )
  ( define angle ( random 0 90 ) )
  ( overlay
    ( rotate angle ( square 30 "solid" "white" ) )
    ( rotate angle ( square 40 "solid" squareColor ) )
    ( rotate angle ( square 50 "solid" "white" ) )
    ( rotate angle ( square 60 "solid" squareColor ) )
    ( square 100 "solid" "white" )
  )
)

( define ( ccs radius difference colors )
  ( define numColors ( length colors ) )
  ( ccs-helper radius difference colors numColors )
)



( define ( ccs-helper radius difference colors numColors )
```

```
    ( cond (( > radius 0 )
      ( define ( colorNum ) ( random numColors ) )
      ( define color ( list-ref colors ( colorNum ) ) )
      ( overlay ( ccs-helper ( - radius difference ) difference colors numColors ) ( circle radius 'solid color ) )
     )
     ( ( = radius 0 ) empty-image )
    )
)
```