Tyler Cullen
CSC 344
2/18/2023

Racket Programming Assignment

Interactions, Definitions, Applications

Abstract

This assignment will focus on recursive programming in order to get us more comfortable with the concept. The constraint during the assignment is that any non recursive iterations are not allowed.

Task 1 - Counting Down | Counting Up

Count-Down

```
> ( 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

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

Code

```
#lang racket

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

( define ( count-up num )
  (cond
    ( ( > num 0 )
      ( count-up (- num 1 ))
      (display num ) (display "\n")
      )
    )
  )
```

Task 2: Triangle of Stars

Demo

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

Code

```
#lang racket
( define (stars num )
  (cond
    (( > num 0)
     (display "*")
     ( stars (- num 1 ))
     )
    )
  )

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

Task 3- Flipping a Coin

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)
```

```
t t
> (flip-for-difference 2)
t t
> (flip-for-difference 2)
t h t t
> (flip-for-difference 2)
h t t t
> (flip-for-difference 3)
h h t h h
> (flip-for-difference 3)
h h t t t t h t t
> (flip-for-difference 3)
h t h t t t t
> (flip-for-difference 3)
h t h t h h h
> (flip-for-difference 4)
t h h h t t h t t h t h h h h t h h
> (flip-for-difference 4)
t h h t t t h t t t
> (flip-for-difference 4)
t h t h h t h t h h h h
> (flip-for-difference 4)
t h t t h h t h h h h h
> (flip-for-difference 4)
h t h h h t h t h h
>
```

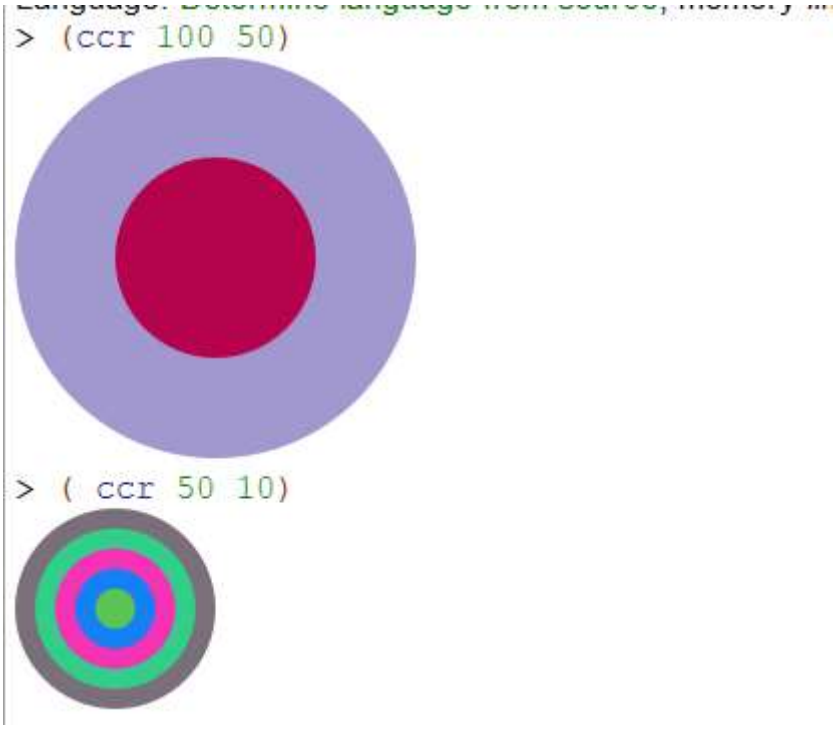
Code

```
#lang racket
(define (coin-flip)
 (define answer ( random 2 ))
 ( cond
   (( = answer 0 ) 't)
   (( = answer 1 ) 'h )
   )
```

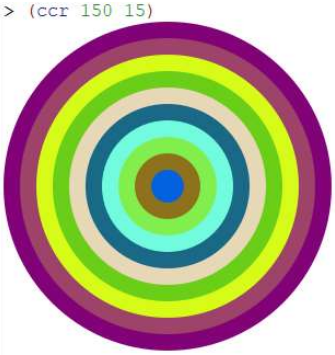
```
 )
(define (flip-for-difference num)
 (define ( flip-helper num count )
  (define negative ( * num -1 ) )
  ( define flip ( coin-flip ) )
  ( cond
    ( ( and ( < count num ) ( > count negative ) )
     ( cond
       ( ( eq? flip 't ) ( display "t ")
                 (flip-helper num (- count 1 ) ) )
       ( ( eq? flip 'h ) (display "h " )
                 ( flip-helper num (+ count 1 ) ) )
       )
     )
    ( else ( display "") ) )
  )
 (flip-helper num 0 )
 )
```

Task 4 - Concentric Disks

CCR Demo



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



```
>
```

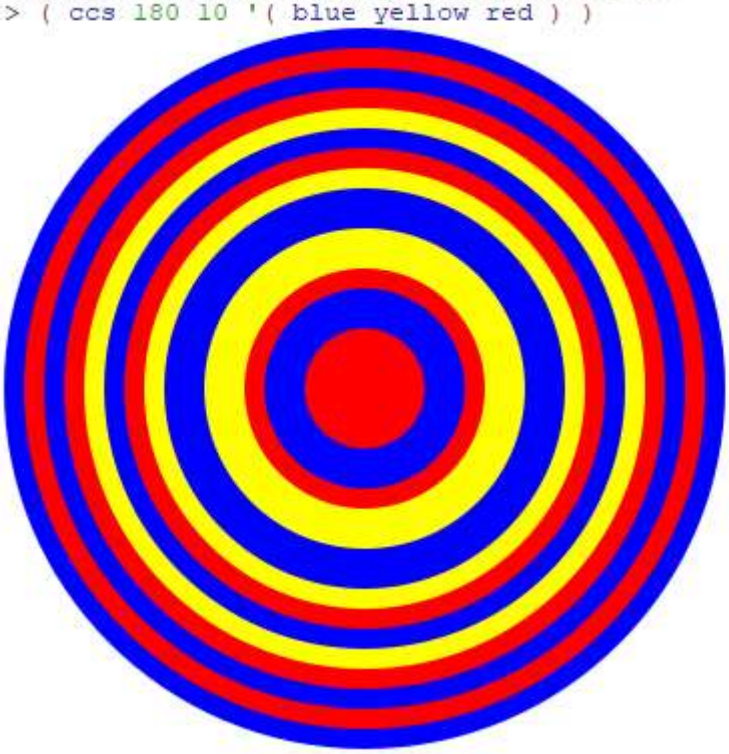
CCA Demo

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

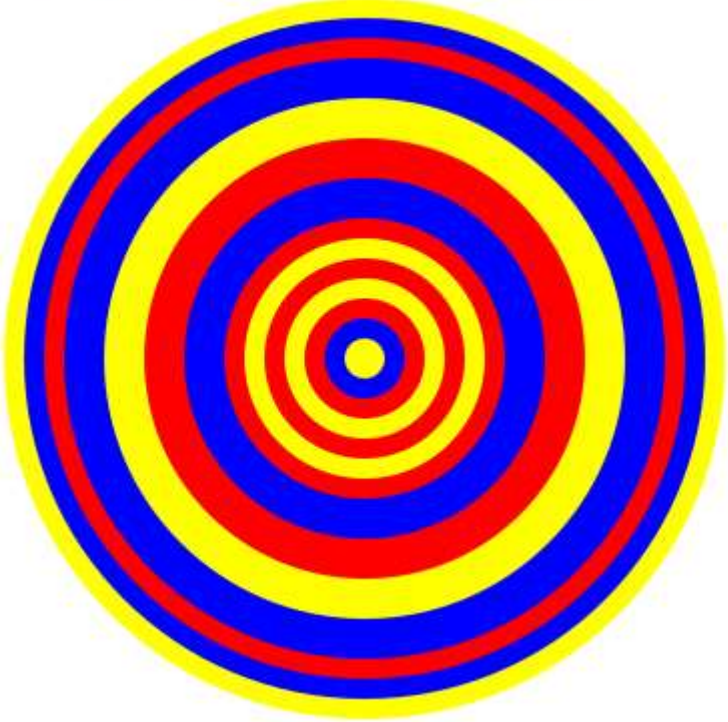
```
> (cca 150 25 'red 'orange )
```

CCS Demo

```
Language: racket, with debugging; memory limit: 128 MB.
> ( ccs 180 10 '( blue yellow red ) )
```



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

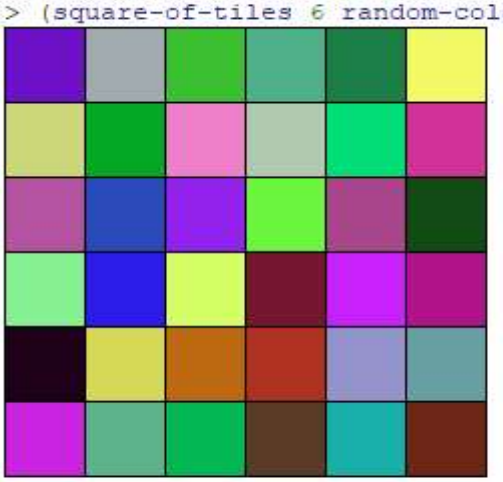


Code

```
#lang racket

(require 2htdp/image)
( define (rand-color)
  (color (random 256) (random 256) ( random 256 )
         )
  )

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

 )

(define ( cca radius diff c1 c2)
 (cond
  (( <= radius 0 ) empty-image)
  (( > radius 0 )
   ( overlay ( cca (- radius diff) diff c2 c1) ( circle radius "solid" c1) )
   )
  )
 )


(define ( ccs radius diff color )
 (cond ((<= radius 0 ) empty-image)
       ((> radius 0 )
        (define (ran-color listing )
        (list-ref color (random (length color))))
        ( overlay ( ccs ( - radius diff ) diff color) ( circle radius "solid" (ran-color color))) ) ) )
```

Random Colored Tile Demo

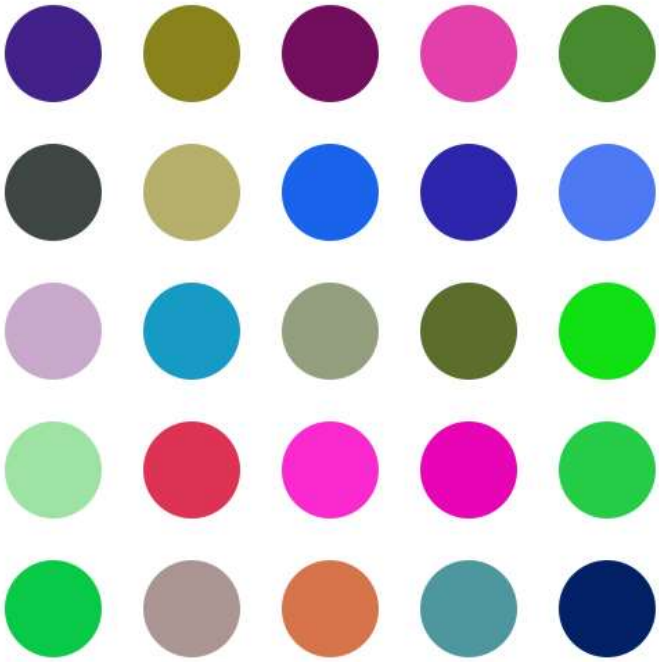
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (square-of-tiles 6 random-color-tile )
```



```
>
```

Hirst Dots

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( square-of-tiles 5 dot-tile)
```



CCS Dots

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( square-of-tiles 7 ccs-tile )
```

Nested Diamonds

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

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

```
>
```

Unruly Squares

Code

```
#lang racket
(require 2htdp/image)

(define ( ccs radius diff color )
 (cond ((<= radius 0 ) empty-image)
       ((> radius 0 )
        (define (ran-color listing )
```

```
        (list-ref color (random (length color))))
        ( overlay ( ccs ( - radius diff ) diff color) ( circle radius "solid" (ran-color
color)))
         )
         )
        )

( define ( row-of-tiles num tile)
  (cond
    ( ( = num 0) empty-image )
    ( ( > num 0 )
      ( beside (row-of-tiles ( - num 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 num tile )
  (rectangle-of-tiles num num tile ) )

( define ( random-color-tile )
```

```
  ( overlay
    (square 40 "outline" "black")
    (square 40 "solid"  (random-color)  )
    )
  )
( define ( random-color  )
  ( define( ranColor ) ( random 0 256 ) )
  ( color ( ranColor ) ( ranColor ) ( ranColor ) )
  )

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

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

( define ( random-colors num )
  ( cond
    ( ( > num 0 )
      ( cons ( random-color) ( random-colors (- num 1 ) ) )
      )
```

```
    ( ( = num 0 ) empty ) ) )

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


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