Racket Assignment #3: Recursion

Cameron Francois
Feburary 28th, 2023
CSC 344

Abstract:

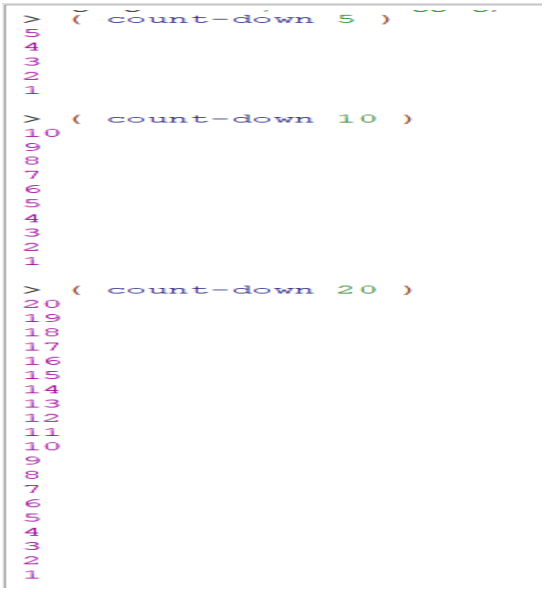
This Assignment demonstrates various ways recursion can be used in racket. As shown in task 1-3 we focus on counting up and down, displaying stars, and heads or tails. In the final two task we demonstrated the ability to produce various images while using our prior knowledge as we progressed through the tasks.

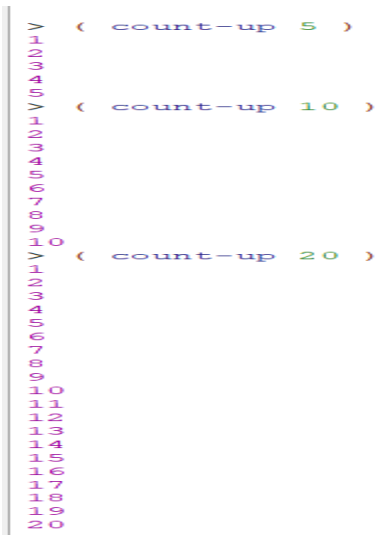
Task 1: Counting Down, Counting Up:

Code:

```
#lang racket

#| Task 1 |#

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

        ( ( >= 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:





Task 2: Triangle of Stars:

Code:

```
29  #| Task 2 |#
30
31  ( define ( star-rows n )
32     ( cond
33        ( ( > n 0 )
34          ( display "*" )
35          ( star-rows ( - n 1 ) ) )
36        )
37     )
38  )
39
40  ( define ( triangle-of-stars n )
41     ( cond
42        ( ( > n 0 )
43          ( triangle-of-stars ( - n 1 ) )
44          ( star-rows n )
45          ( display "\n" )
46        )
47     )
48  )
```

Demo:

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

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

Task 3: Flipping a Coin:

Code:

```
#lang racket

#| Task 3 |#

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

( define ( flip-for-difference n )
    ( flip-diff-function n ( * n 2 ) )
)

( define ( flip-diff-function value n )
    ( cond (
            ( not ( or ( = value 0 )
                       ( = value n ) ) )
            ( define outcomes ( flip-coin ) )
            ( display outcomes )
            ( display "" )
            ( cond
                ( ( eq? outcomes 'h )
                    ( flip-diff-function ( - value 1 ) n )
                )
                ( ( eq? outcomes 't )
                    ( flip-diff-function ( + value 1 ) n )
                )
            )
        )
    )
)
```

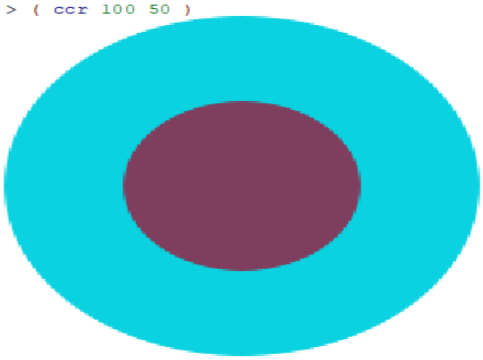
Demo:

```
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
t
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
h
> ( flip-for-difference 2 )
hh
> ( flip-for-difference 2 )
htthttt
> ( flip-for-difference 2 )
hh
> ( flip-for-difference 2 )
hh
> ( flip-for-difference 2 )
thtt
> ( flip-for-difference 2 )
tt
> ( flip-for-difference 3 )
htttthhtthhthhttththhthhh
> ( flip-for-difference 3 )
hhh
> ( flip-for-difference 3 )
hhttthhthhhthhh
> ( flip-for-difference 3 )
htttt
> ( flip-for-difference 3 )
thhhtthttthtthhthhtt
> ( flip-for-difference 3 )
tthhthhttthhhhhthh
> ( flip-for-difference 4 )
hthhttthhthttthhhhh
> ( flip-for-difference 4 )
hhhthhthhh
> ( flip-for-difference 4 )
tthhtthttt
> ( flip-for-difference 4 )
thtttt
> ( flip-for-difference 4 )
hhhh
> ( flip-for-difference 4 )
ththhhhthttthhhh
> ( flip-for-difference 4 )
hthhththhhhttththhhhh
> ( flip-for-difference 4 )
htthhhhh
>
```

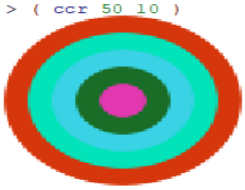
Task 4: Laying Down Colorful Conentric Disks:

CCR Demo:

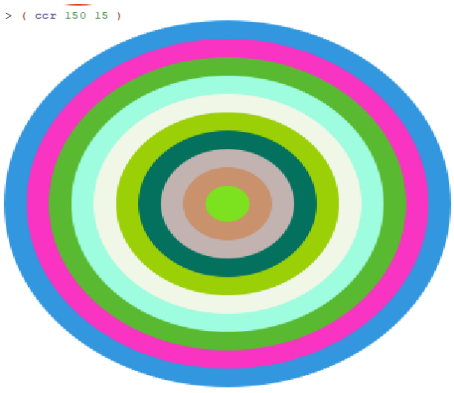
```
> ( ccr 100 50 )
```



```
> ( ccr 50 10 )
```

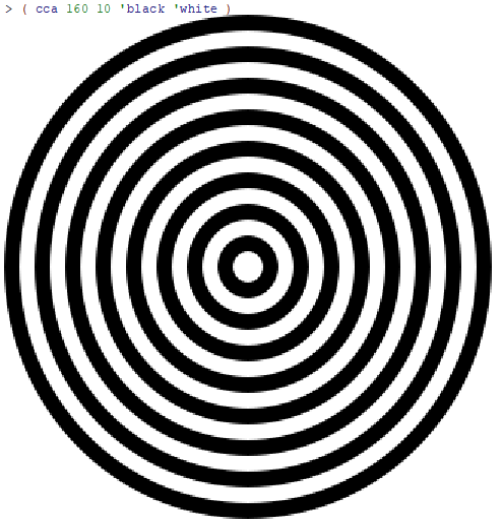


```
> ( ccr 150 15 )
```

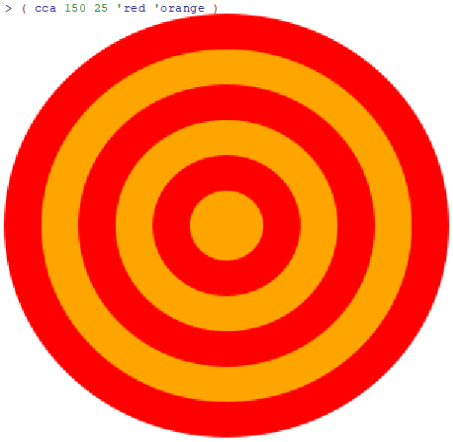


CCA Demo:

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



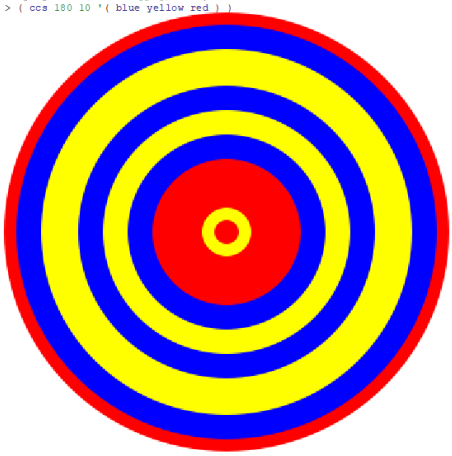
```
> ( cca 150 25 'red 'orange )
```



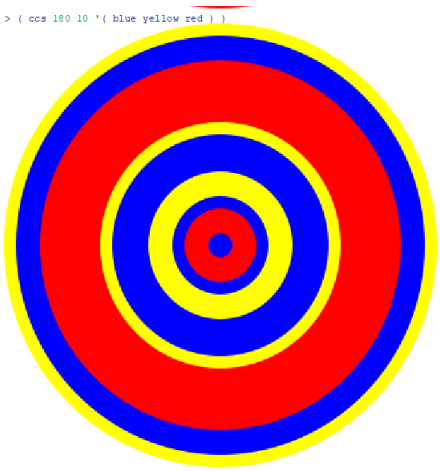
```
>
```

CCS Demo 1:

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

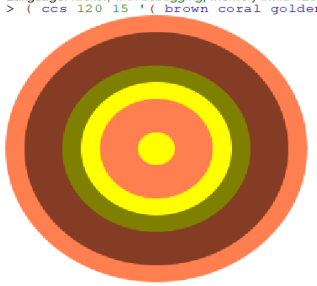


```
> ( ccs 180 10 '( blue yellow red ) )
```



CCS Demo 2:

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



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

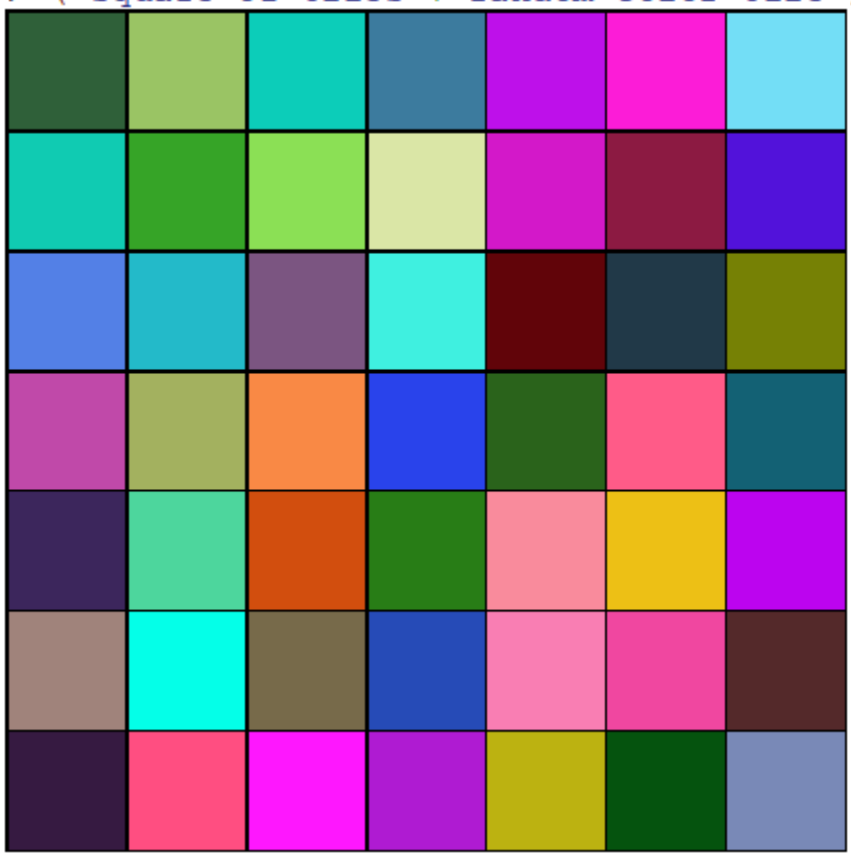
Code:

```
#lang racket

#| Task 4 |#

( require 2htdp/image )

(define ( rdm-color )
  ( color ( random 256 ) (random 256 ) ( random 256 ) ) )

( define ( ccr r diff )
   ( cond ( ( > r 0 )
            ( overlay ( ccr ( - r diff ) diff ) ( circle r 'solid ( rdm-color ) ) )
           )
          ( ( = r 0 ) empty-image )
    )
  )

( define ( cca r diff c1 c2 )
   ( cca-func r diff c1 c2 1 )
 )

( define ( cca-func r diff c1 c2 c-num )
   ( cond ( ( > r 0 )
          ( cond ( ( = c-num 1 )
                 ( overlay ( cca-func ( - r diff ) diff c1 c2 2 )
                           ( circle r 'solid c1 ) ) )
                 ( ( = c-num 2 )
                 ( overlay ( cca-func ( - r diff ) diff c1 c2 1 )
                           ( circle r 'solid c2 ) ) )
                 )
          )
         ( ( = r 0 ) empty-image )
      )
   )
( define ( ccs r diff colors )
   ( define num-colors ( length colors ) )
   ( ccs-func r diff colors num-colors )
 )

( define ( ccs-func r diff colors num-colors )
   ( cond ( ( > r 0 )
     ( define (c-num ) ( random num-colors ) )
     ( define color ( list-ref colors (c-num ) ) )
     ( overlay ( ccs-func ( - r diff ) diff colors num-colors )
               ( circle r 'solid color ) )
     )
      ( ( = r 0 ) empty-image )
     )
   )
```

Task 5: Variations on Hirst Dots:

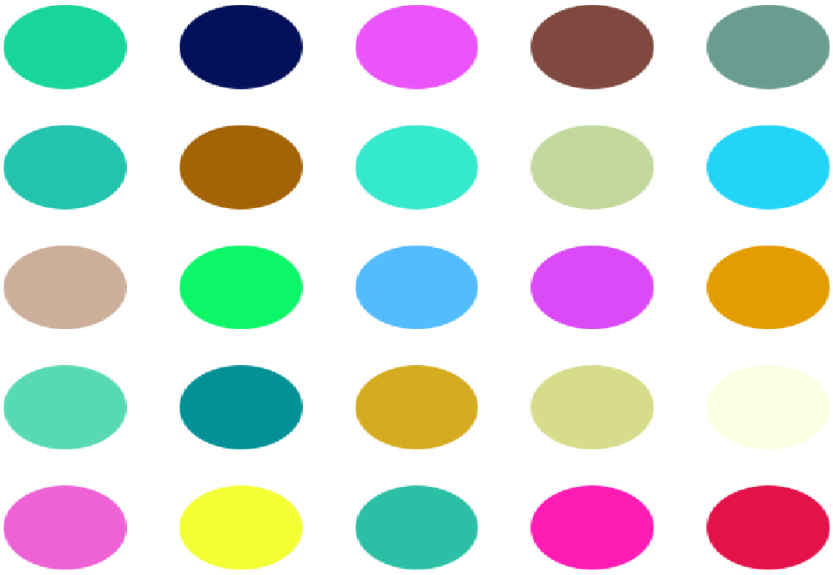
Random Colored Tile Demo:

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



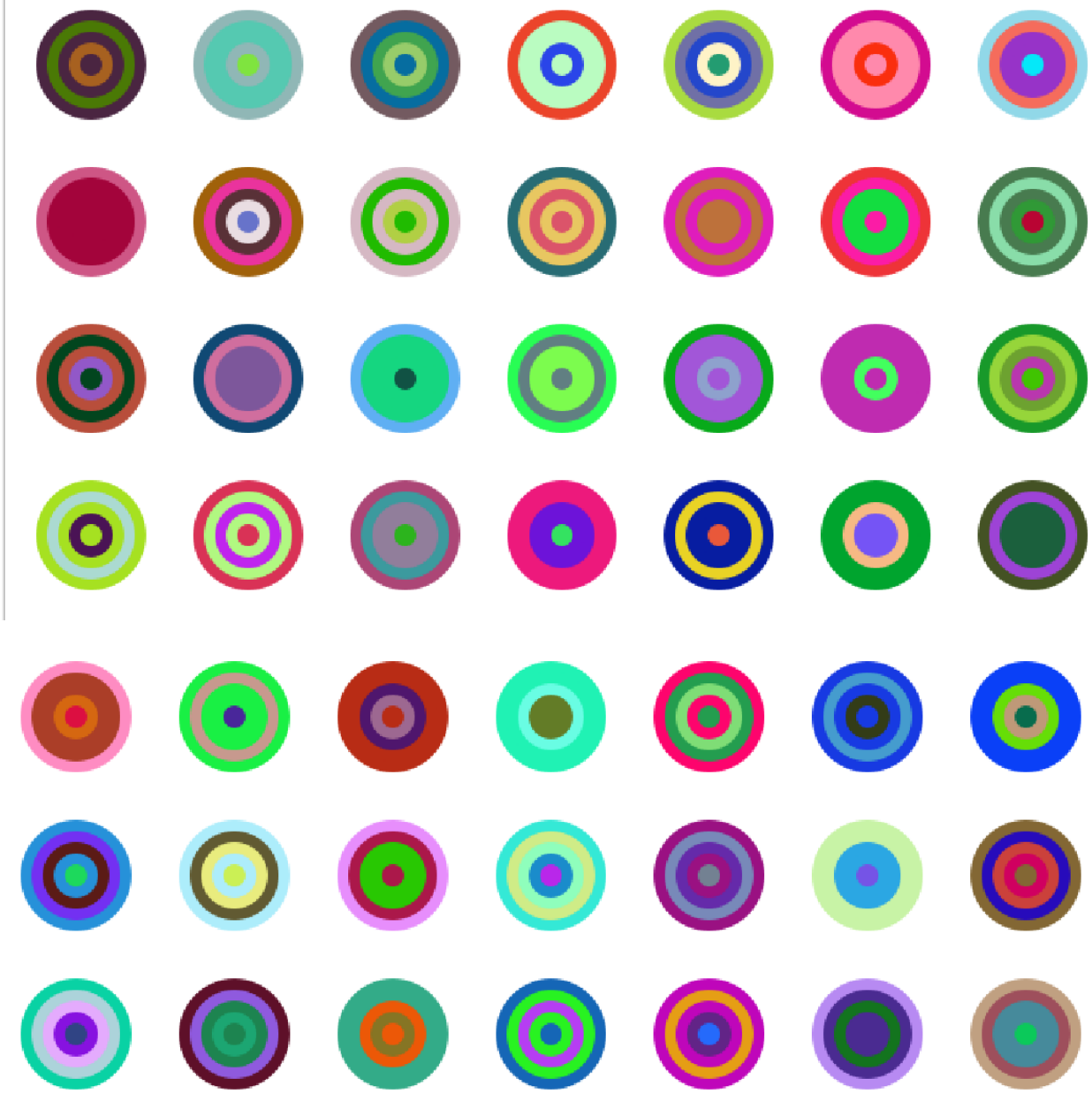
Hirst Dots Demo:

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



CCS Dots Demo:

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



Nested Diamonds Demo:

Language: racket, with debugging; memory limit: 128 MB.

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

Unruly Squares Demo:

Language: racket, with debugging; memory limit: 128 MB.

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

Code:

```
#lang racket
#| Task 5 |#
( require 2htdp/image )



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

( define ( dots-tile )
   ( overlay
     ( circle 35 "solid" ( rdm-color ) )
     ( square 100 "solid" "black" )
     )
 )

( define ( color-maker x )
   ( let loop ( ( y x ) ( colors '() ) )
   ( cond
      ( ( = y 0 ) ( reverse colors ) )
      ( else ( loop ( - y 1 ) ( cons ( rdm-color ) colors ) ) ) ) ) )

( define ( ccs-tile )
   ( define colors ( color-maker 5 ) )
   ( overlay
     ( ccs 35 7 colors )
     ( square 100 "solid" "black" )
     )
   )


   )

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

( define ( wild-square-tile )
   ( rotate ( random 180 ) ( diamond-tile ) )
   )

( define ( rows-of-tiles x tile )
  ( cond
     ( ( = x 0 ) empty-image )
     ( ( > x 0 )
       ( beside ( rows-of-tiles ( - x 1 ) tile ) ( tile ) ) )
     )
   )

( define ( rec-of-tiles c f tile )
   ( cond
      ( ( = c 0 ) empty-image )
      ( ( > c 0 )
        ( above
          ( rec-of-tiles ( - c 1 ) f tile ) ( rows-of-tiles f tile ) ) )
     )
   )

( define ( square-of-tiles x tile )
   ( rec-of-tiles x x tile )
   )
```