Racket Assignment #3: Recursions in Racket

Learning Abstract

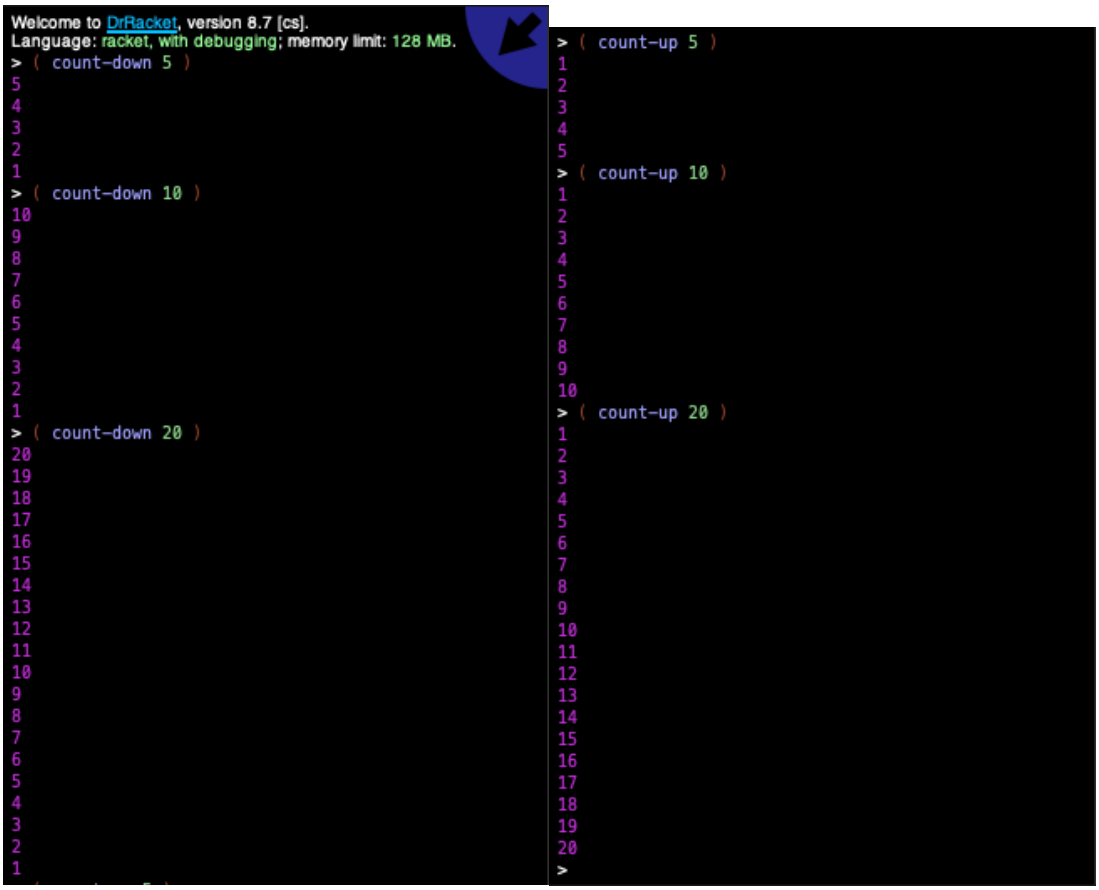
In this assignment, I will be making recursion functions for a variety of applications, ranging from simple number and star recursions all the way to more complex, and graphical shapes reminiscent of Racket Assignment #2.

Task 1: Counting Down, Counting Up

Code

```
#lang racket
( define ( count-down i )
   ( cond
      ( ( > i 0 )
      ( display i )( display "\n" )
      ( count-down ( - i 1 ) )
      )
   )
)

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

Demo



Task 2: Triangle of Stars

Code

```
#lang racket
( define ( triangle-of-stars i )
   ( cond
      ( ( > i 0 )
         ( define ( line n )
            ( cond
               ( ( <= n i )
                 ( define ( starring j )
                    ( cond
                       ( ( <= j n )
                          ( display "*" )
                          ( starring ( + j 1 ))
                       )
                    )
                 )
                 ( starring 1 )
                 ( display "\n" )
```

```
                    ( line ( + n 1 ) )
                )
            )
        )
        ( line 1 )
    )
)
)
```

Demo

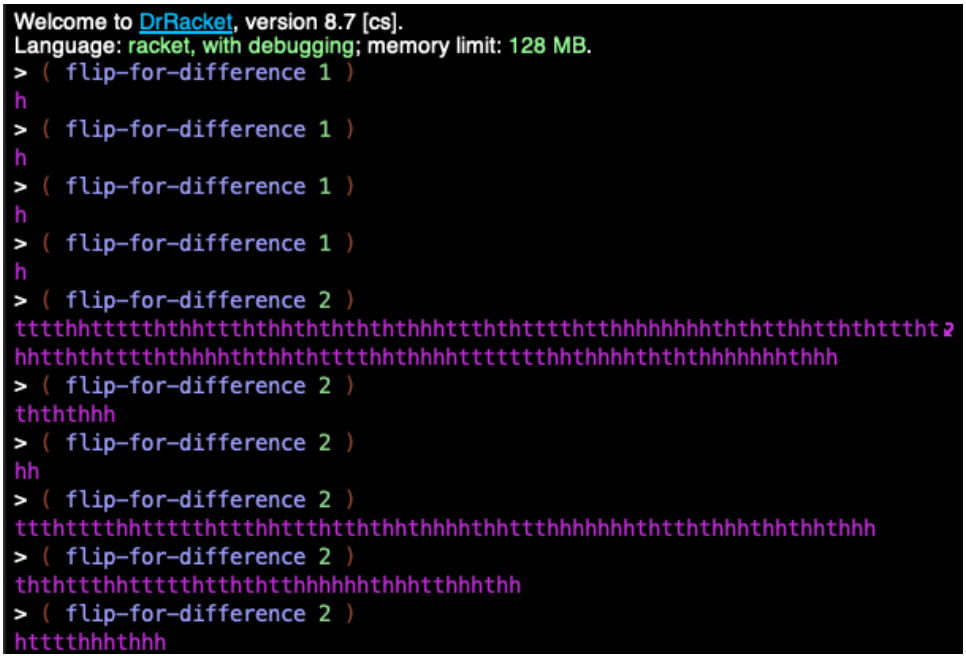
Task 3: Flipping a Coin

Code

```
#lang racket

( define hCount 0 )
( define tCount 0 )

( define ( flip-for-difference n )
```

```
    ( define ( flipCoin target )
      ( set! hCount 0 )
      ( set! tCount 0 )
      ( let loop ()
        ( if ( not ( equal? ( - hCount tCount ) target ) )
            ( begin
              ( if ( equal? ( random 2 ) 0 )
                  ( begin ( display "h" )
                     ( set! hCount ( + hCount 1 ) )
                  )
                  ( begin ( display "t" )
                     ( set! tCount ( + tCount 1 ) )
                  )
              )
              (loop)
            )
        (void)
        )
      )
    )
    (flipCoin n)
)
```

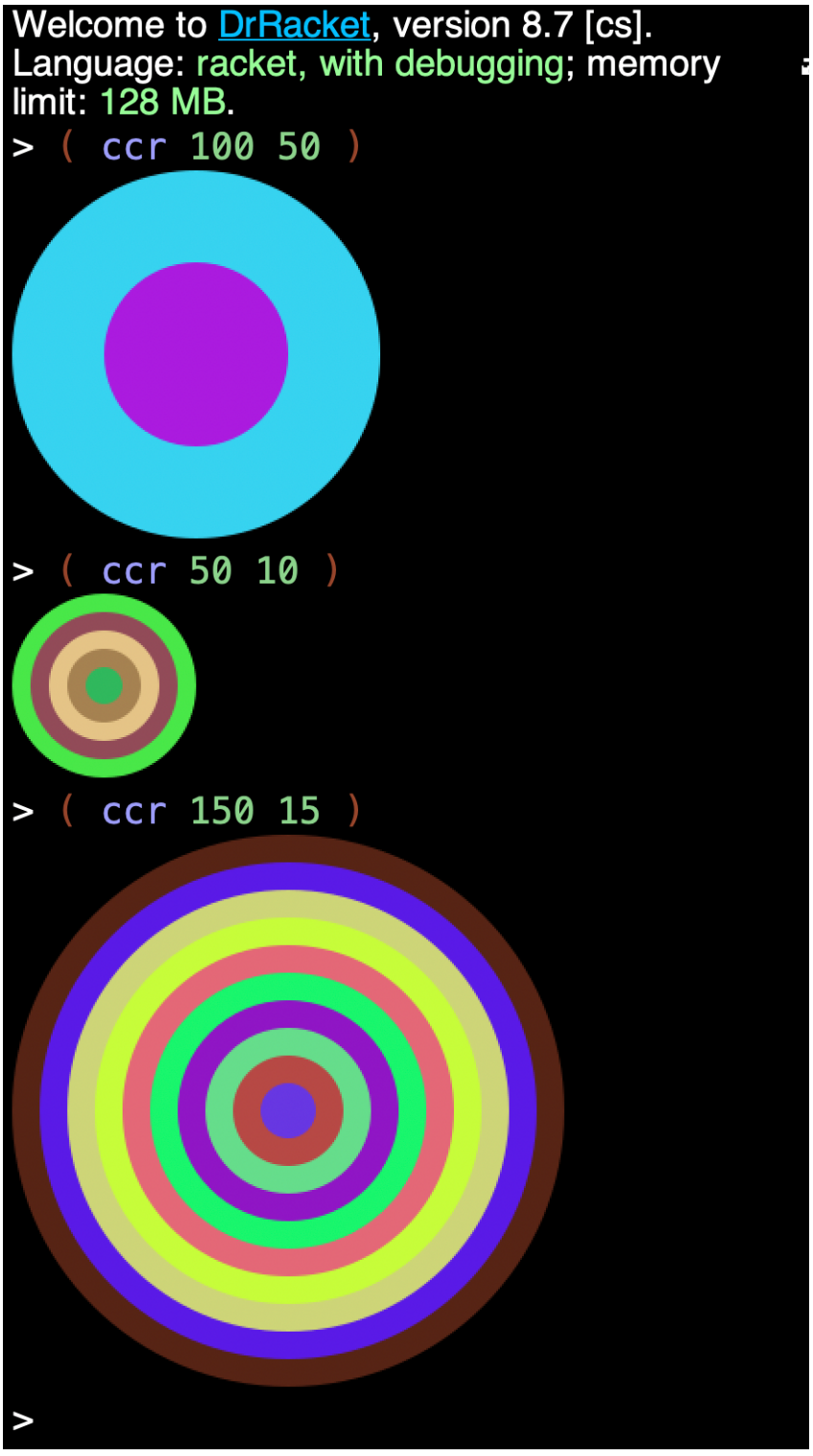
Demo




> (flip-for-difference 3)
htthhthttthhhh
> (flip-for-difference 3)
hhttthhttttthhttttthhhthhthhthhttthhhhhhthhhttttthhhhthhthhthhthhh
> (flip-for-difference 3)
hhthttthhh
> (flip-for-difference 3)
htthhhh
> (flip-for-difference 3)
> (flip-for-difference 4)
> (flip-for-difference 4)
> (flip-for-difference 4)
> (flip-for-difference 4)
> (flip-for-difference 4)
> (flip-for-difference 4)
> (flip-for-difference 4)
>

Task 4: Laying Down Colorful Concentric Disks

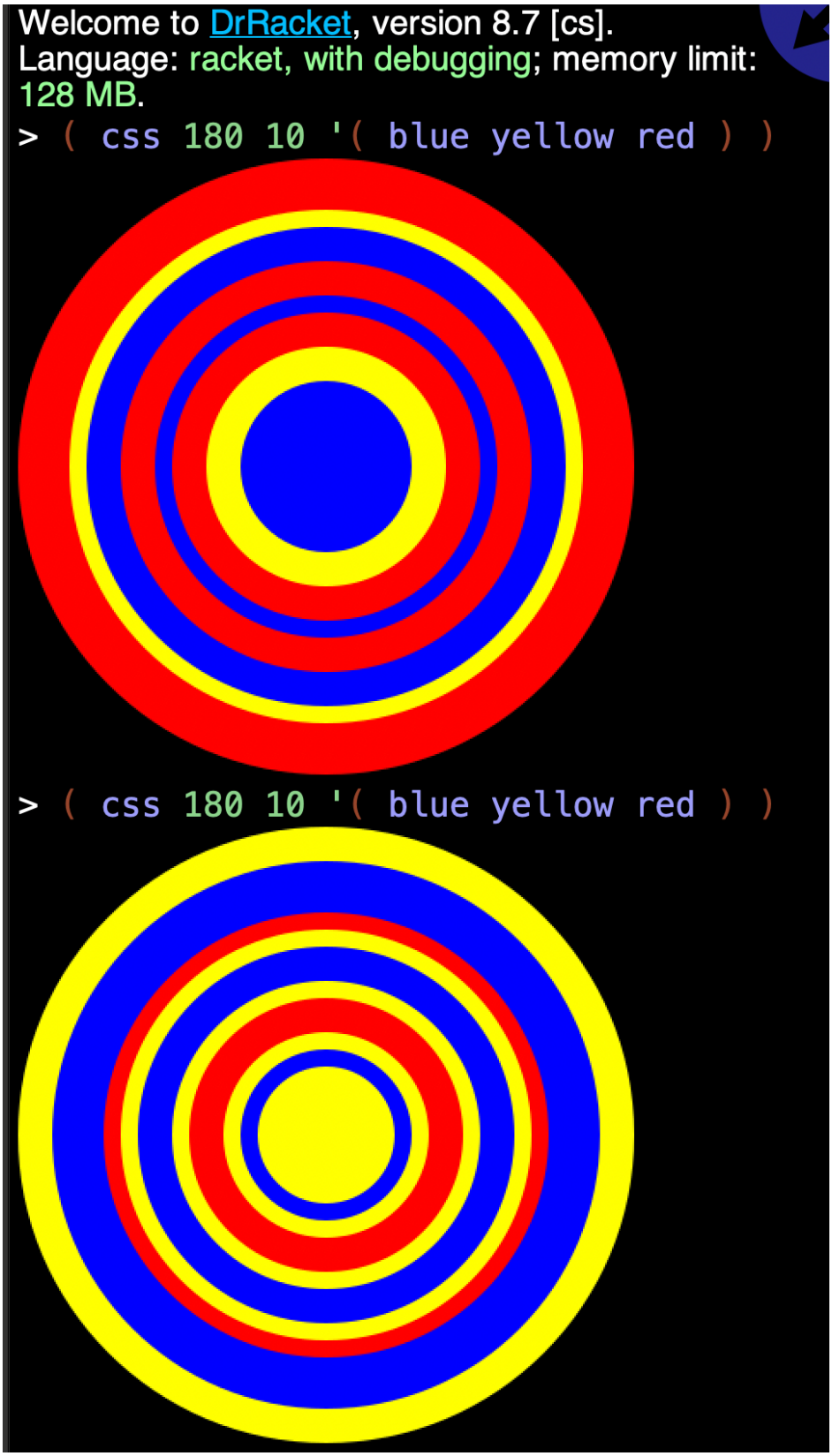
CCR Demo



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 Demo 1


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

CCS Demo 2


Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (css 120 15 '(brown coral goldenrod yellow olive tan))
> (css 120 15 '(brown coral goldenrod yellow olive tan))
>

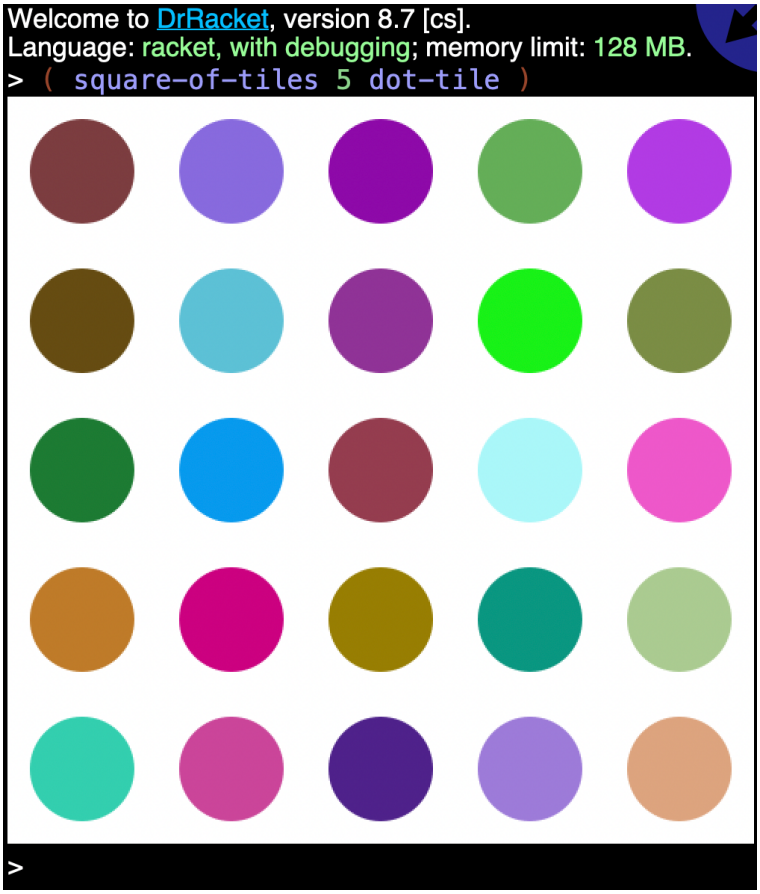
Code

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

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

( define ( ccr r1 d1 )
   ( cond
      ( ( <= r1 0 ) empty-image )
        ( ( > r1 0)
          ( overlay ( ccr ( - r1 d1 ) d1 )
          ( circle r1 "solid" ( rc ) )
          )
        )
   )
)

( define ( cca r1 d1 color1 color2 )
   ( cond
      ( ( <= r1 0 ) empty-image )
        ( ( > r1 0)
          ( overlay ( cca ( - r1 d1 ) d1 color1 color2 )
             ( circle r1 "solid"
                ( if ( even? ( quotient r1 d1 ) ) color1 color2 )
             )
          )
        )
   )
)

( define ( css r1 d1 colorList )
   ( cond
      ( ( <= r1 0 ) empty-image )
        ( ( > r1 0)
          ( let ( ( currColor ( list-ref colorList ( random ( length
colorList ) ) ) ) )
             ( overlay ( css ( - r1 d1 ) d1 colorList )
                ( circle r1 "solid" currColor)
             )
          )
       )
   )
)
```

Task 5: Variations on Hirst Dots

Random Colored Tile Demo

Hirst Dots Demo



CCS Dots Demo

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

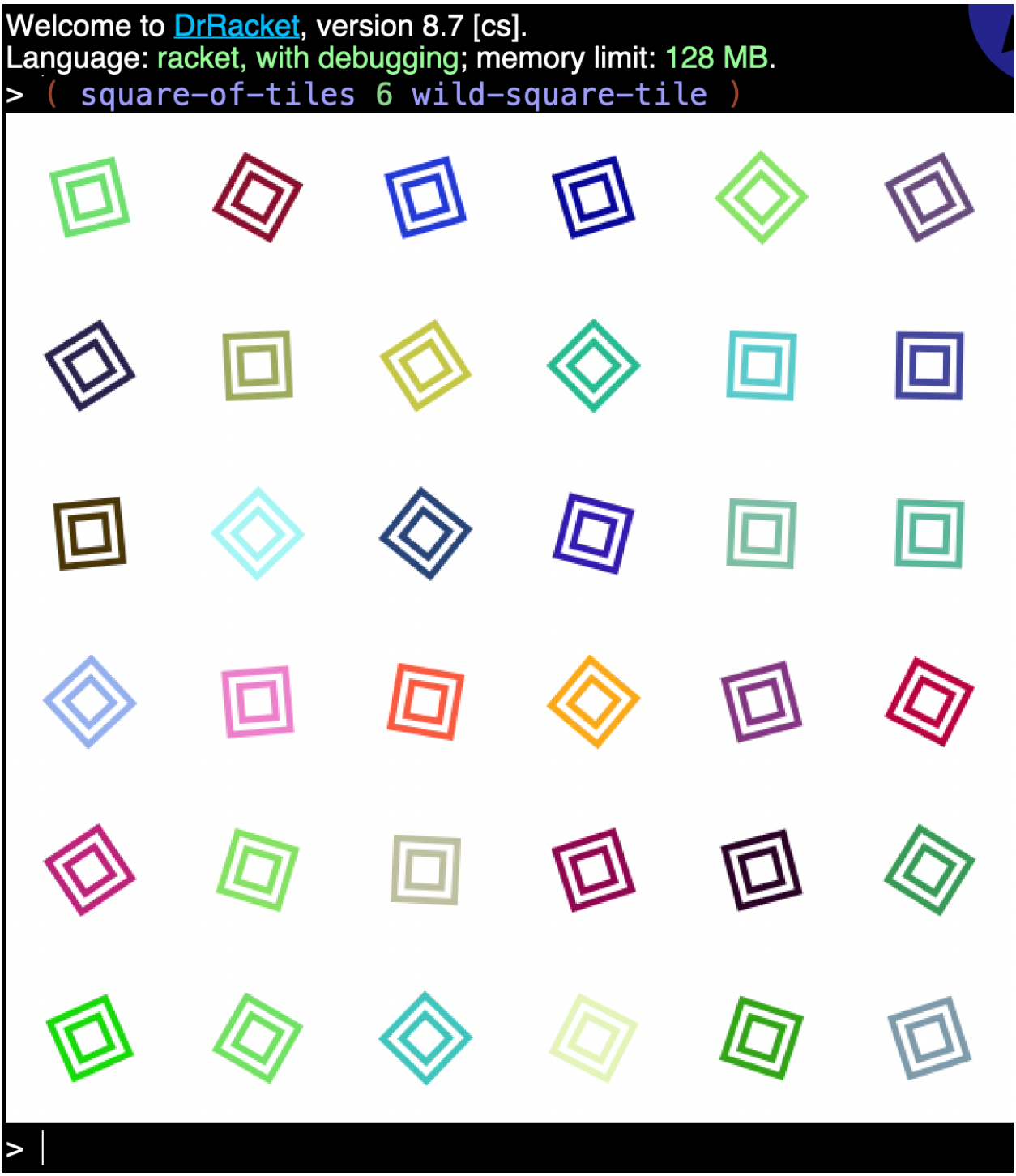
```
>
```

Nested Diamonds Demo

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

```
>
```

Unruly Squares Demo



Code

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

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

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

( 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 r1 d1 )
   ( cond
      ( ( <= r1 0 ) empty-image )
        ( ( > r1 0)
          ( overlay ( ccs ( - r1 d1 ) d1 )
          ( circle r1 "solid" ( random-color ) )
          )
        )
   )
)

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

( define ( diamond-tile )
   ( define c1 "white" )
   ( define c2 ( random-color ) )
   ( overlay
     ( rotate 45 ( square 16 "solid" c1 ) )
     ( rotate 45 ( square 24 "solid" c2 ) )
     ( rotate 45 ( square 32 "solid" c1 ) )
     ( rotate 45 ( square 40 "solid" c2 ) )
     ( square 100 "solid" c1 )
   )
)

( define ( random-degree )
   ( random 360 )
)

( define ( wild-square-tile )
   ( define c1 "white" )
   ( define c2 ( random-color ) )
   ( define d1 ( random-degree ) )
   ( overlay
     ( rotate d1 ( square 16 "solid" c1 ) )
     ( rotate d1 ( square 24 "solid" c2 ) )
     ( rotate d1 ( square 32 "solid" c1 ) )
     ( rotate d1 ( square 40 "solid" c2 ) )
     ( square 100 "solid" c1 )
   )
)
```