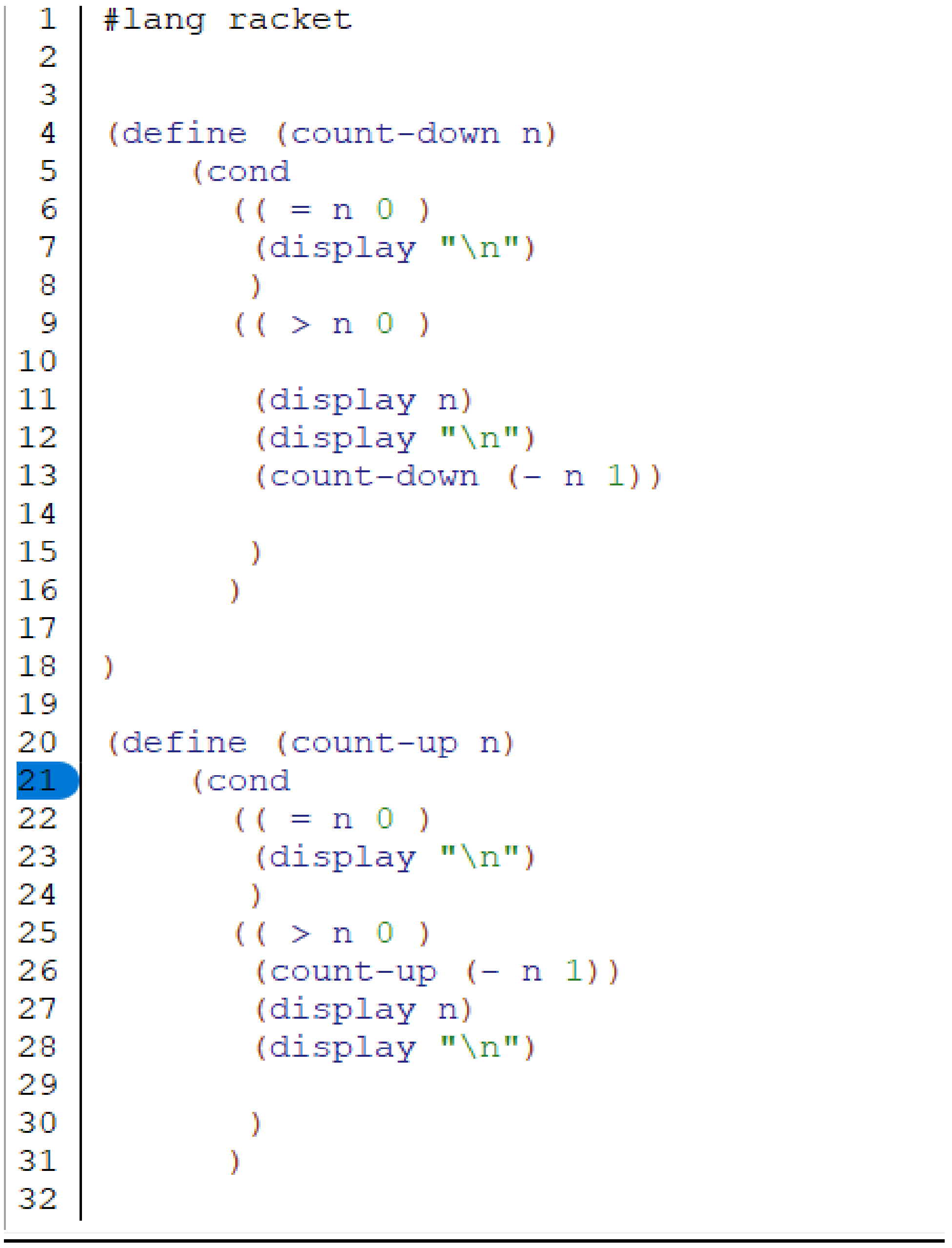
Racket Assignment #3 Recursions in Raacket

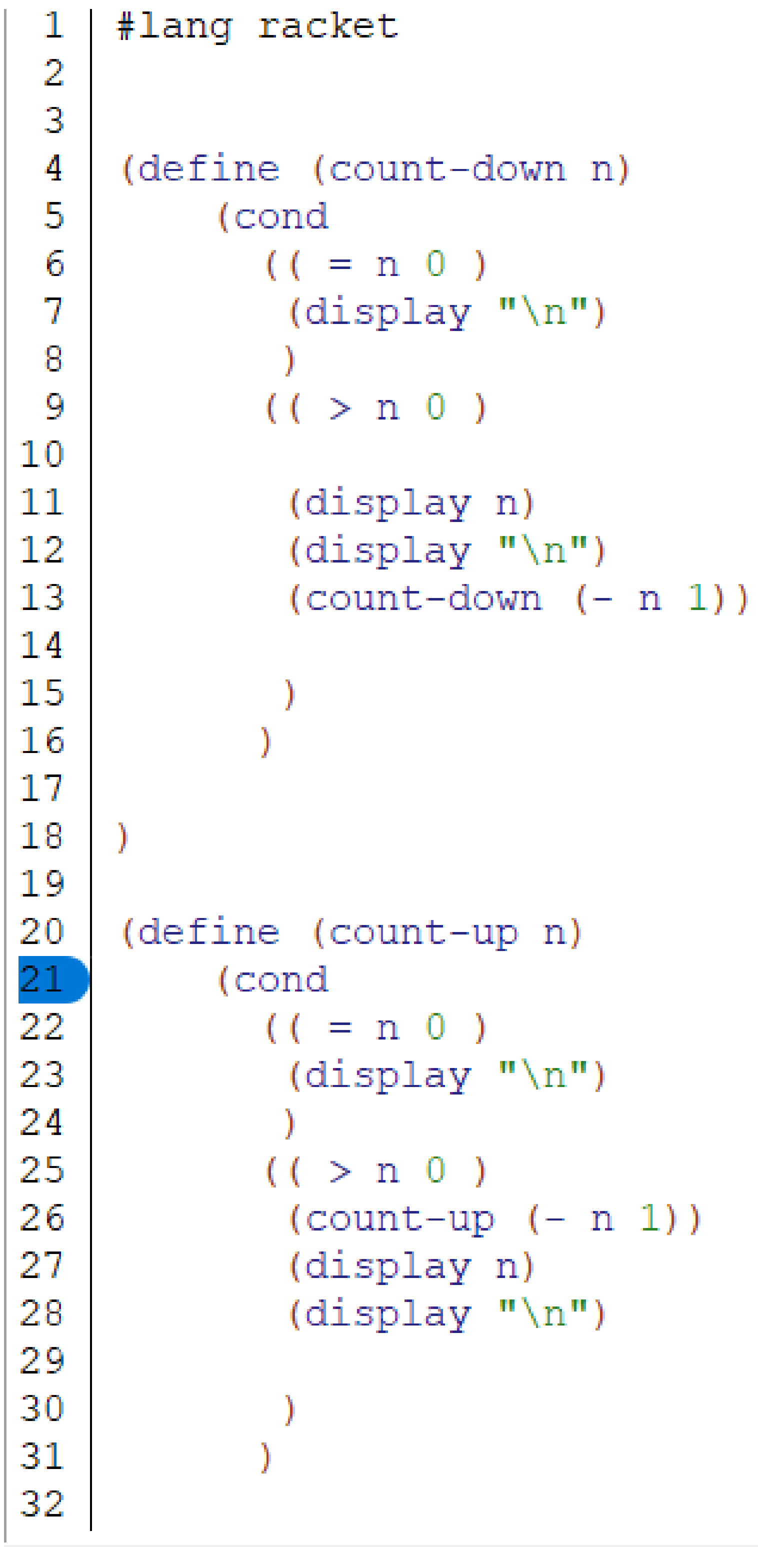
Task 1: Counting Down, Counting Up

Code

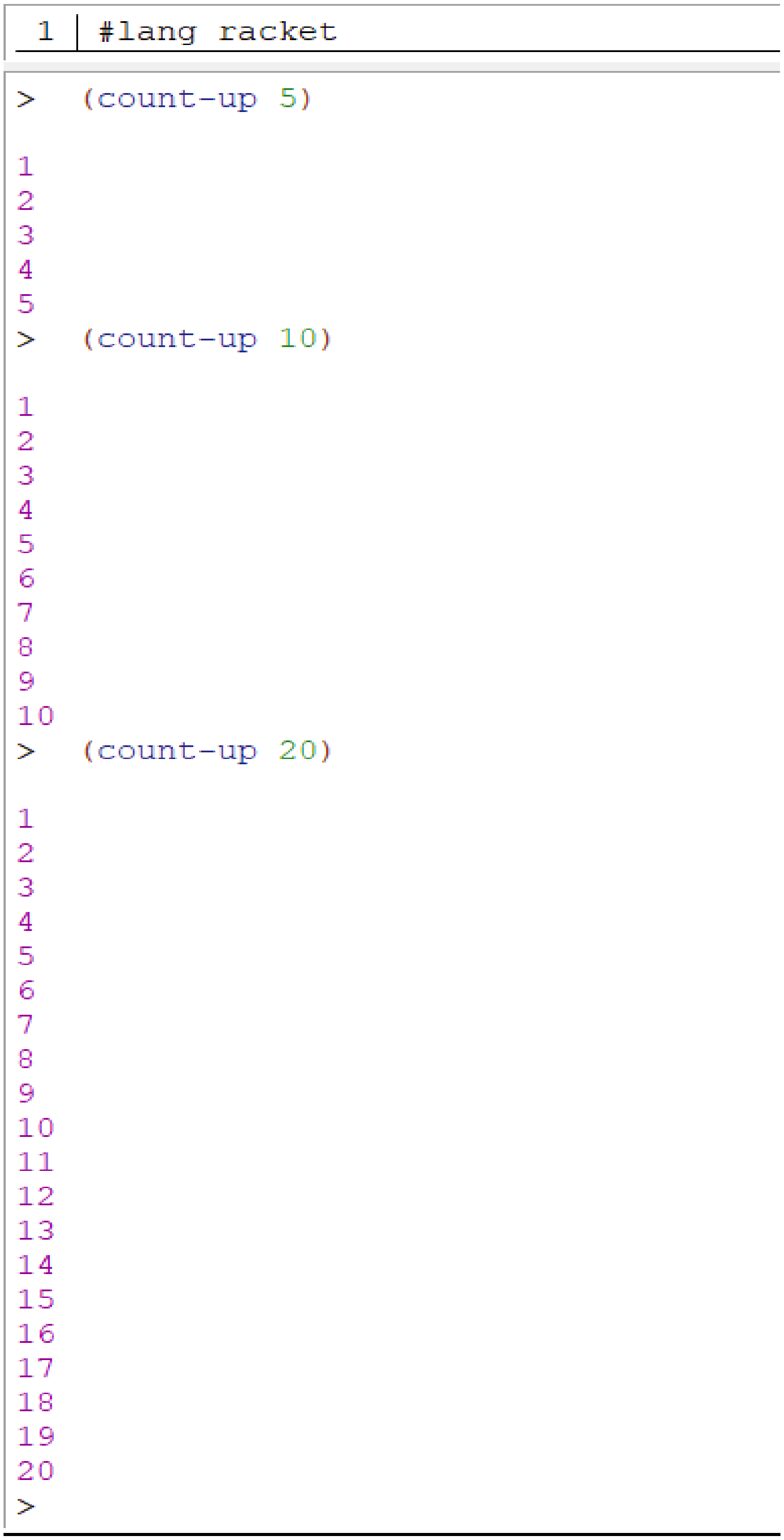


```racket
#lang racket


(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 )
       (display "\n")
       )
      (( > n 0 )
       (count-up (- n 1))
       (display n)
       (display "\n")

       )
      )

```

Demo



```racket
#lang racket


(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 )
       (display "\n")
       )
      (( > n 0 )
       (count-up (- n 1))
       (display n)
       (display "\n")

       )
      )

```

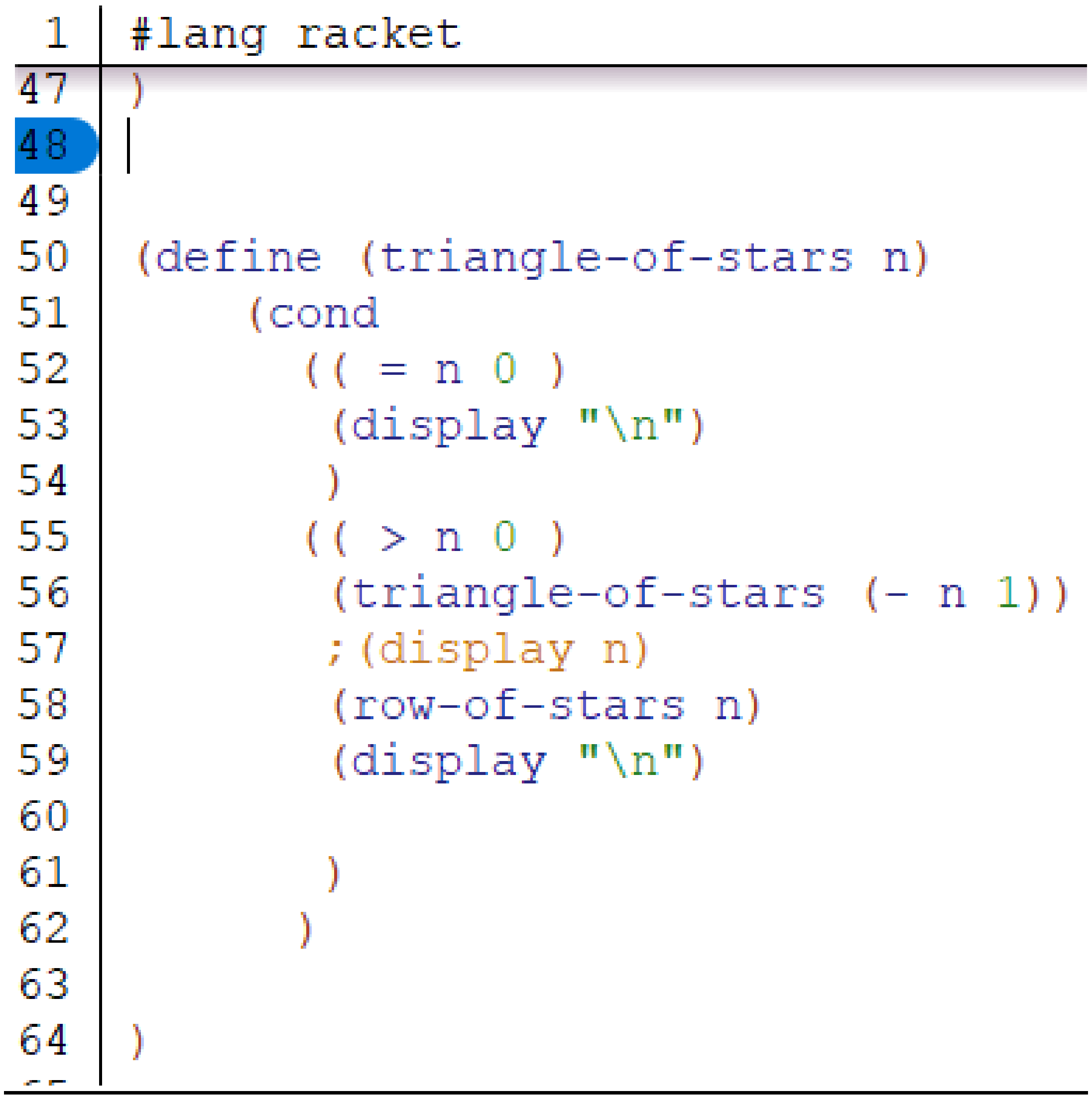


```
#lang racket
```

```
>  (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
>
```

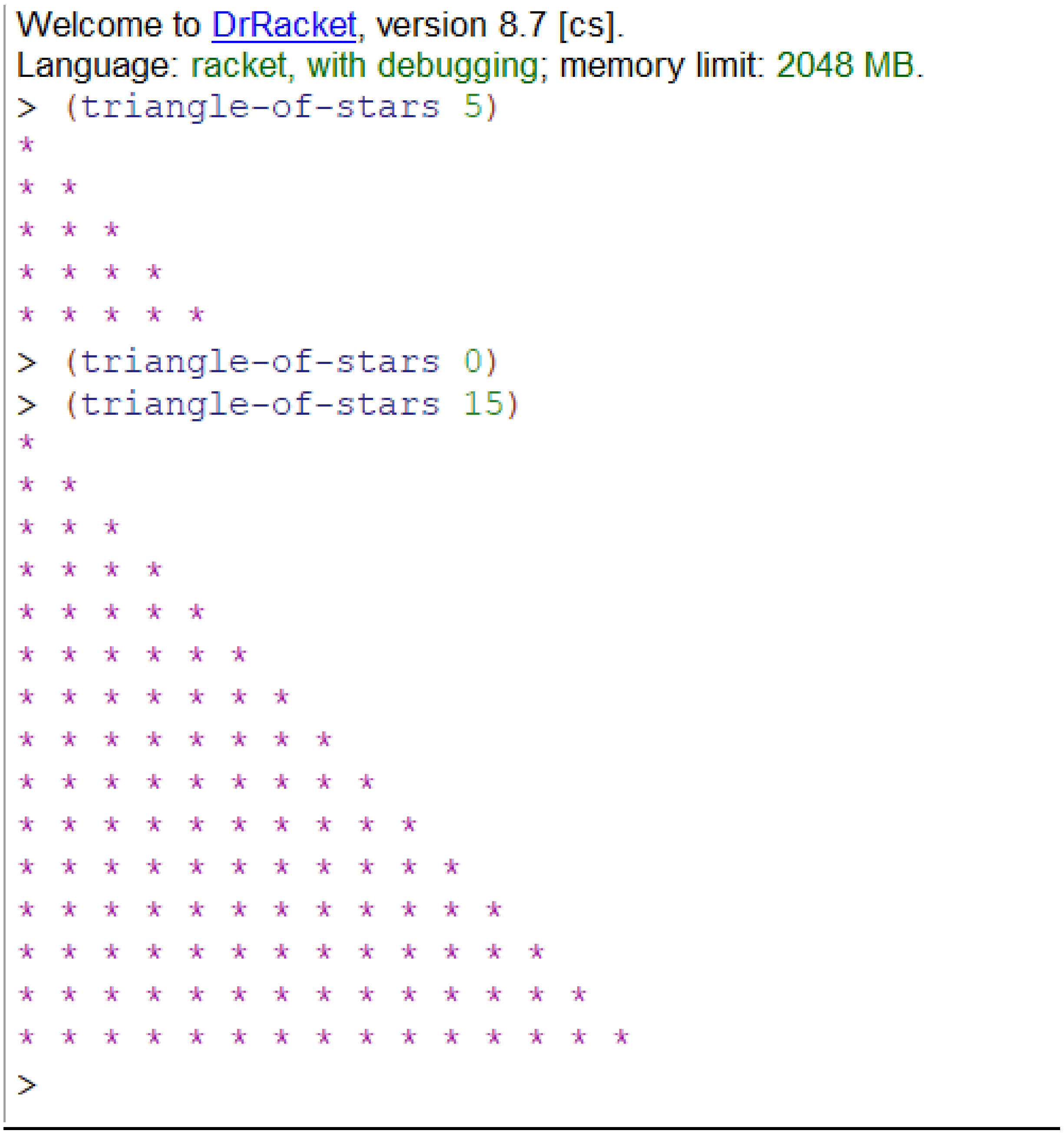
Task 2: Triangle of Stars

Code



```
#lang racket
)


(define (triangle-of-stars n)
    (cond
      (( = n 0 )
       (display "\n")
       )
      (( > n 0 )
       (triangle-of-stars (- n 1))
       ;(display n)
       (row-of-stars n)
       (display "\n")

       )
      )

)
```

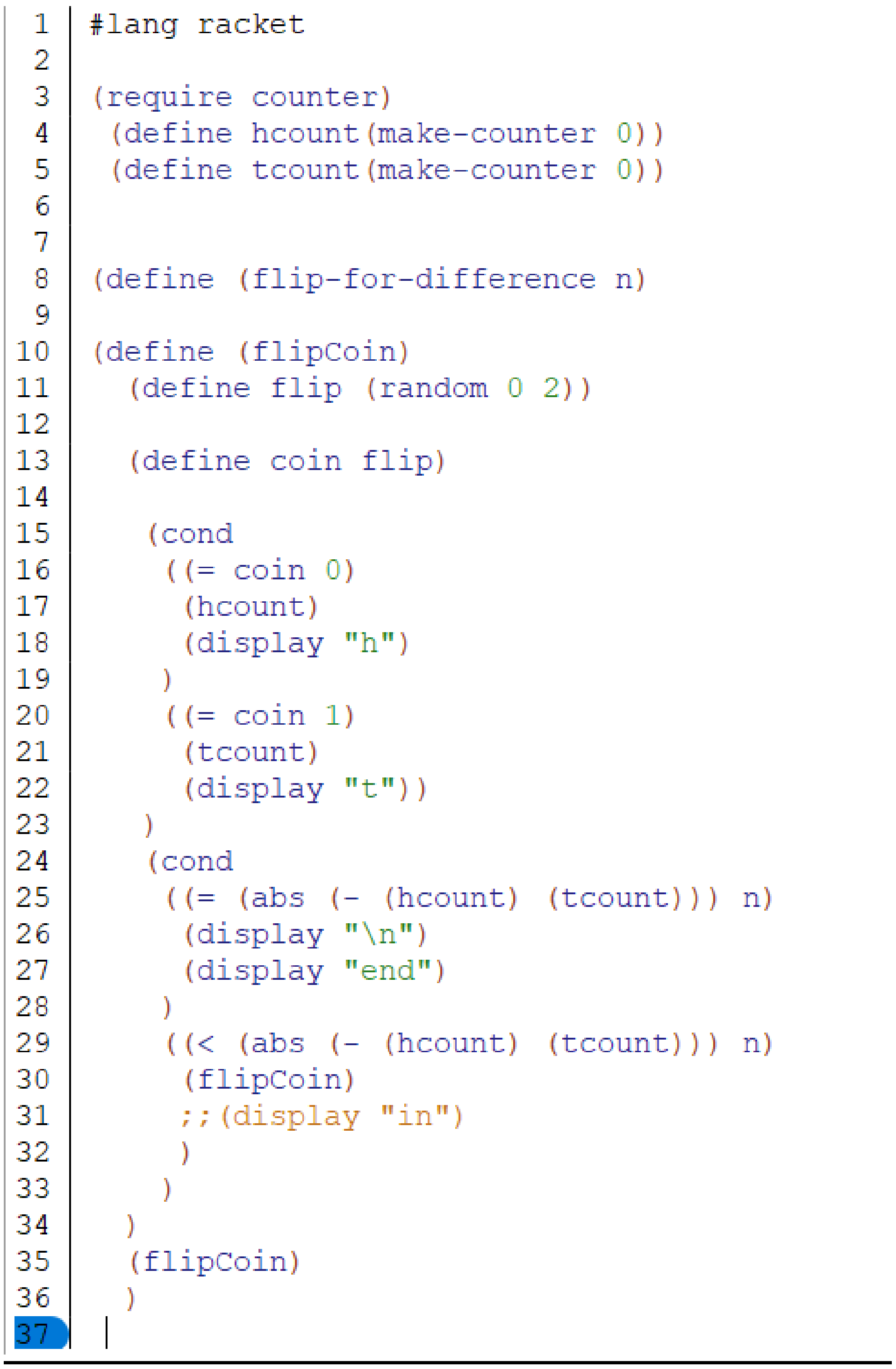
Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (triangle-of-stars 5)
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
> (triangle-of-stars 0)
> (triangle-of-stars 15)
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
*  *  *  *  *  *
*  *  *  *  *  *  *
*  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *
>
```

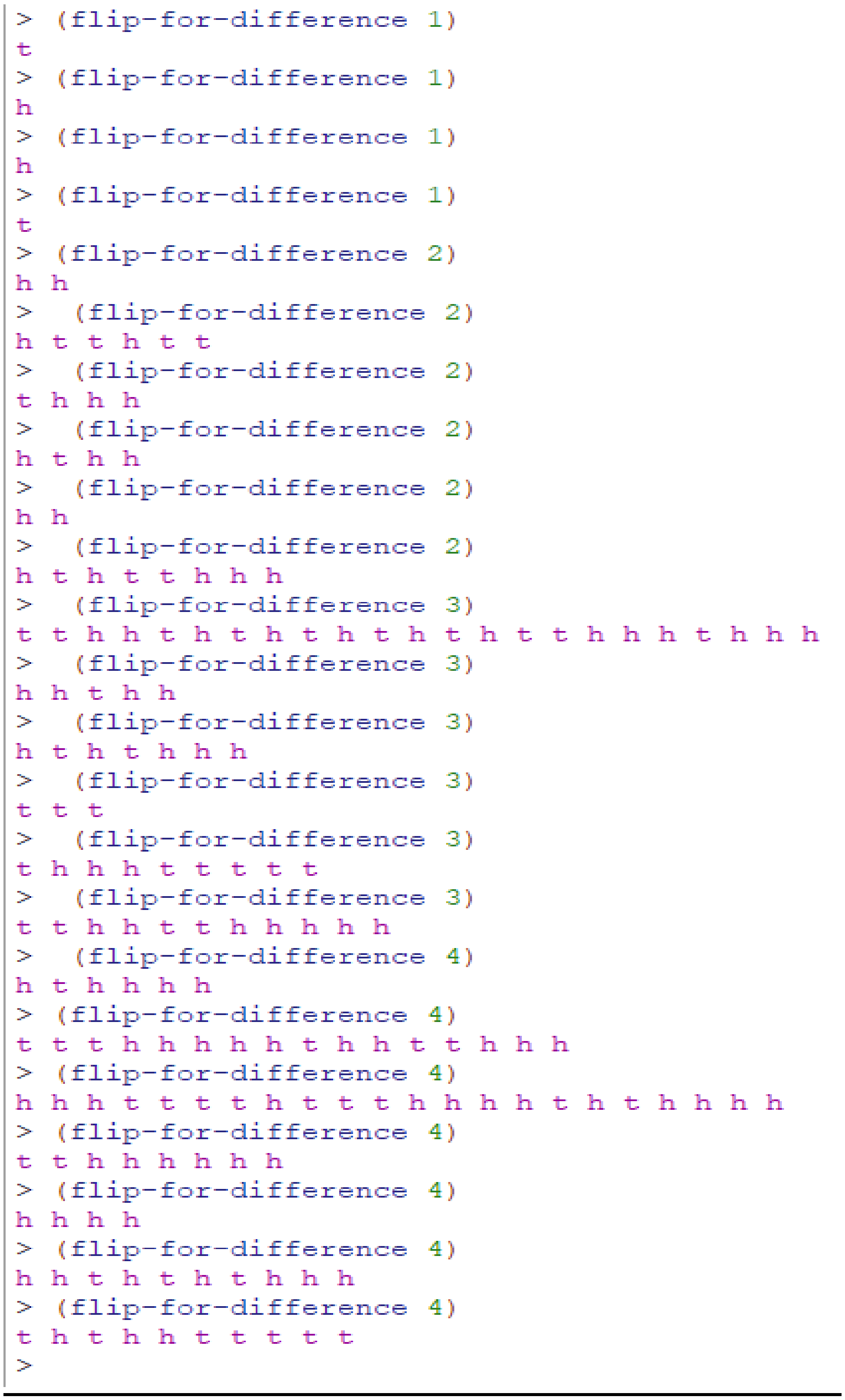
Task 3: Flipping a Coin

Code



```racket
#lang racket

(require counter)
 (define hcount(make-counter 0))
 (define tcount(make-counter 0))


(define (flip-for-difference n)

(define (flipCoin)
  (define flip (random 0 2))

  (define coin flip)

   (cond
    ((= coin 0)
     (hcount)
     (display "h")
    )
    ((= coin 1)
     (tcount)
     (display "t"))
   )
   (cond
    ((= (abs (- (hcount) (tcount))) n)
     (display "\n")
     (display "end")
    )
    ((< (abs (- (hcount) (tcount))) n)
     (flipCoin)
     ;;(display "in")
     )
    )
  )
  (flipCoin)
  )
```

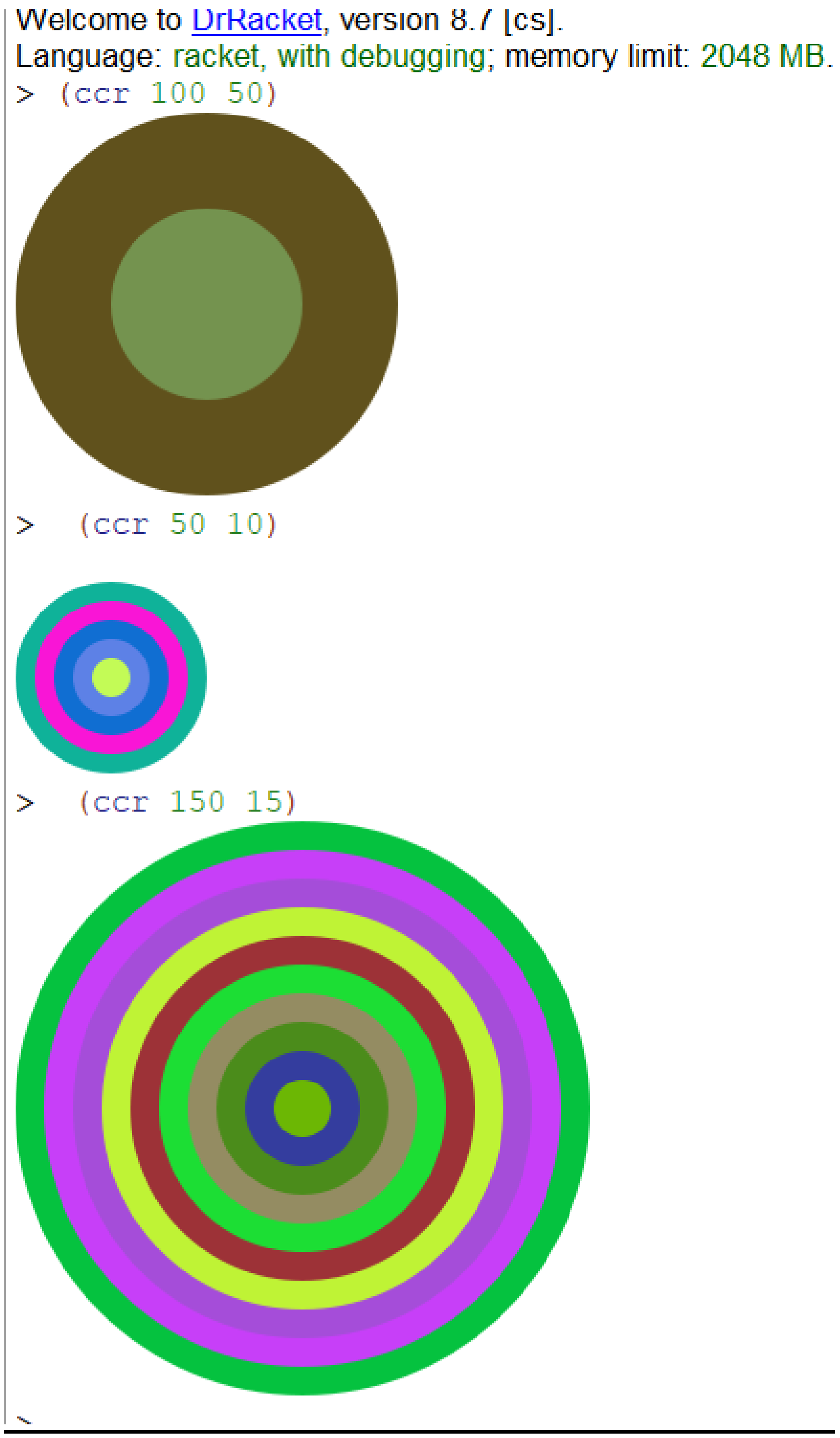
Demo



```
> (flip-for-difference 1)
t
> (flip-for-difference 1)
h
> (flip-for-difference 1)
h
> (flip-for-difference 1)
t
> (flip-for-difference 2)
h h
>  (flip-for-difference 2)
h t t h t t
>  (flip-for-difference 2)
t h h h
>  (flip-for-difference 2)
h t h h
>  (flip-for-difference 2)
h h
>  (flip-for-difference 2)
h t h t t h h h
>  (flip-for-difference 3)
t t h h t h t h t h t h t h t t h h h t h h h
>  (flip-for-difference 3)
h h t h h
>  (flip-for-difference 3)
h t h t h h h
>  (flip-for-difference 3)
t t t
>  (flip-for-difference 3)
t h h h t t t t t
>  (flip-for-difference 3)
t t h h t t h h h h h
>  (flip-for-difference 4)
h t h h h h
> (flip-for-difference 4)
t t t h h h h h t h h t t h h h
> (flip-for-difference 4)
h h h t t t t h t t t h h h h t h t h h h h
> (flip-for-difference 4)
t t h h h h h h
> (flip-for-difference 4)
h h h h
> (flip-for-difference 4)
h h t h t h t h h h
> (flip-for-difference 4)
t h t h h t t t t t
>
```

Task 4: Laying Down Colorful Concentric Disks

CCR Demo



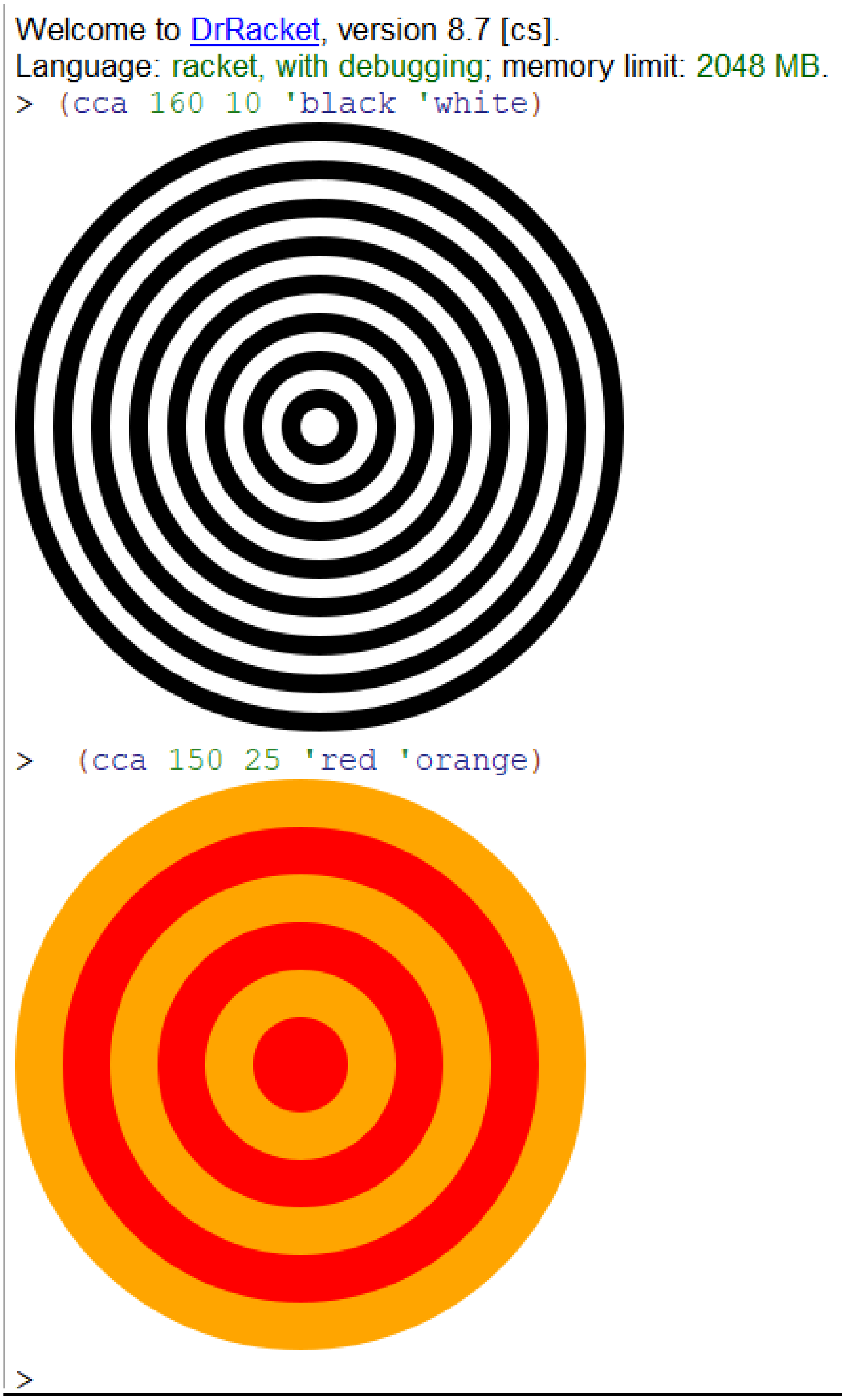
```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (ccr 100 50)
```

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

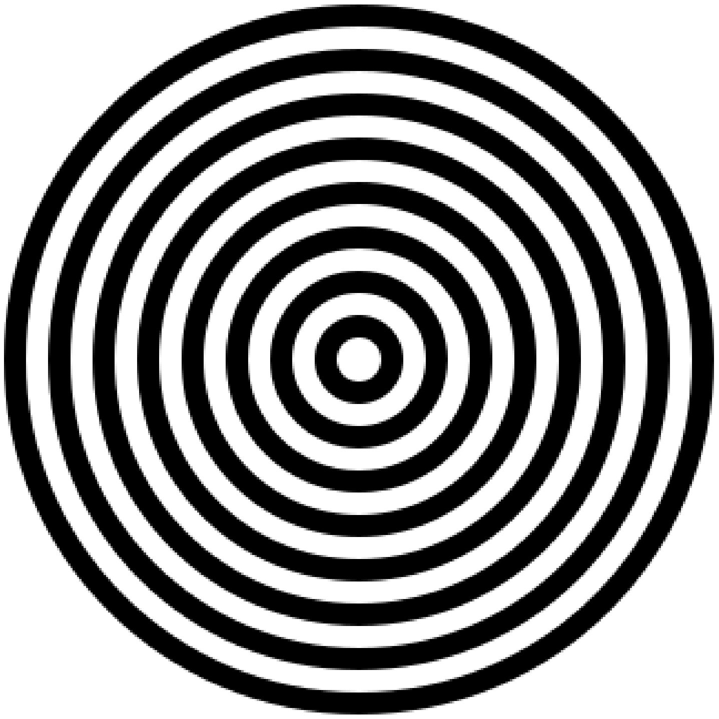
```
>  (ccr 150 15)
```

```
>
```

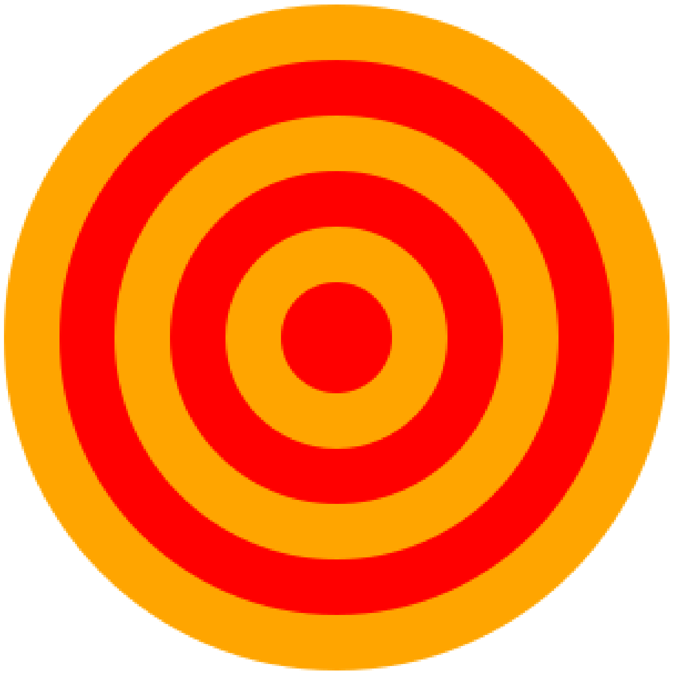
CCA Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (cca 160 10 'black 'white)
```



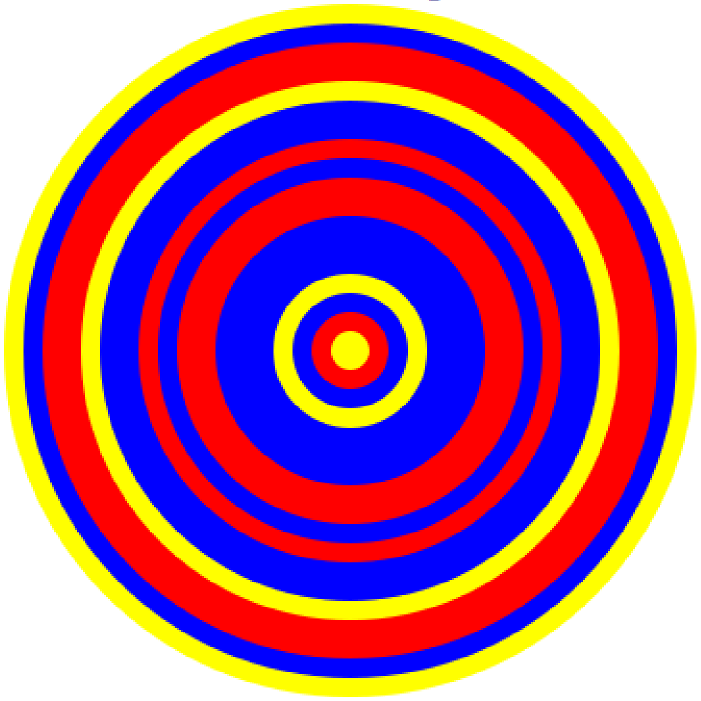
```
>  (cca 150 25 'red 'orange)
```



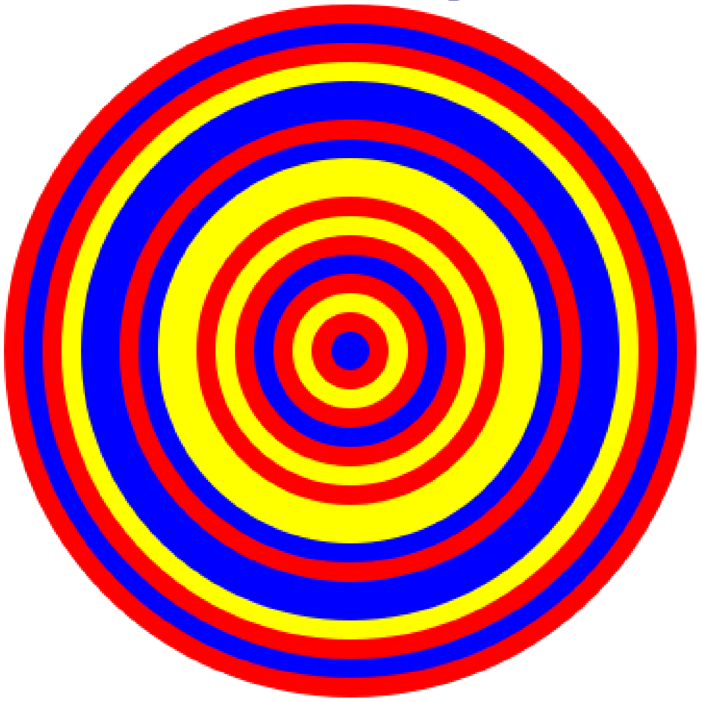
```
>
```

CCS Demo 1

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
>  (ccs 180 10 '(blue yellow red))
```



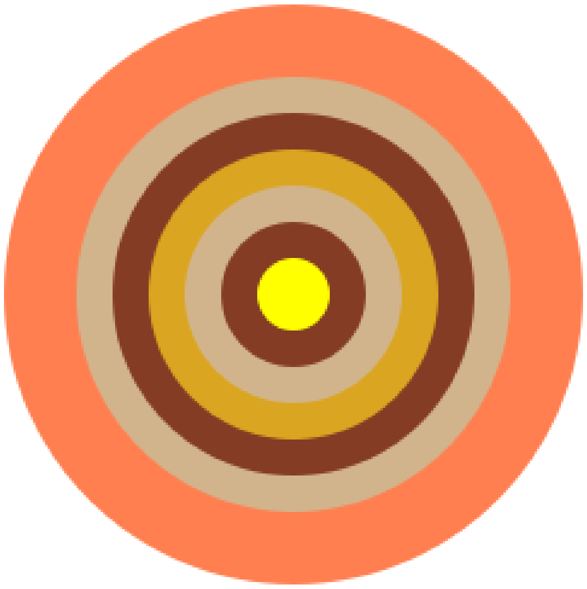
```
>   (ccs 180 10 '(blue yellow red))
```



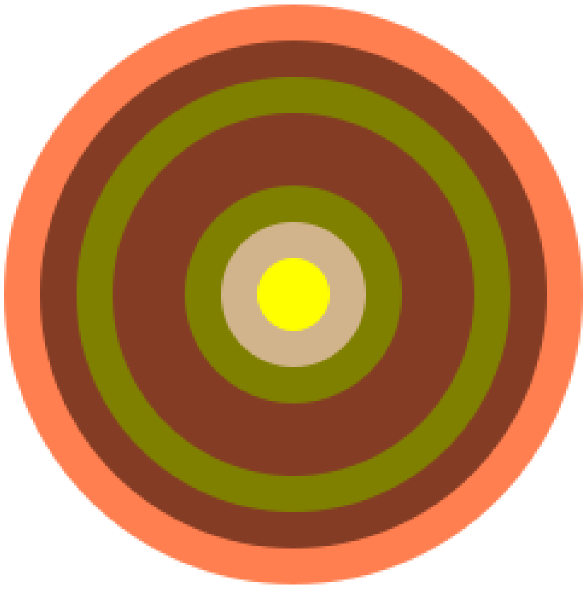
```
>
```

CCS Demo 2

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
>  (ccs 120 15 '(brown coral goldenrod yellow olive tan))
```

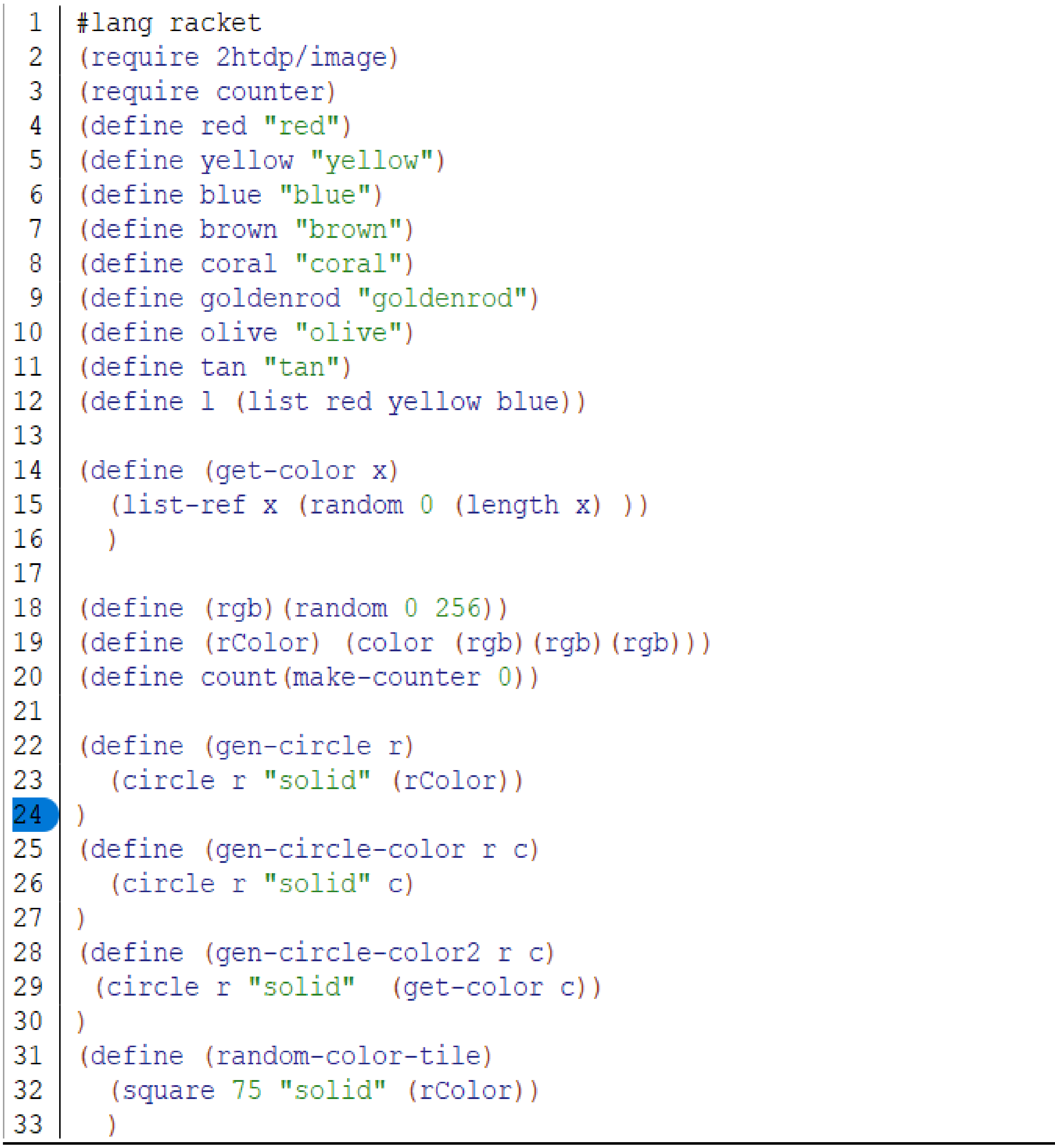


```
> (ccs 120 15 '(brown coral goldenrod yellow olive tan))
```

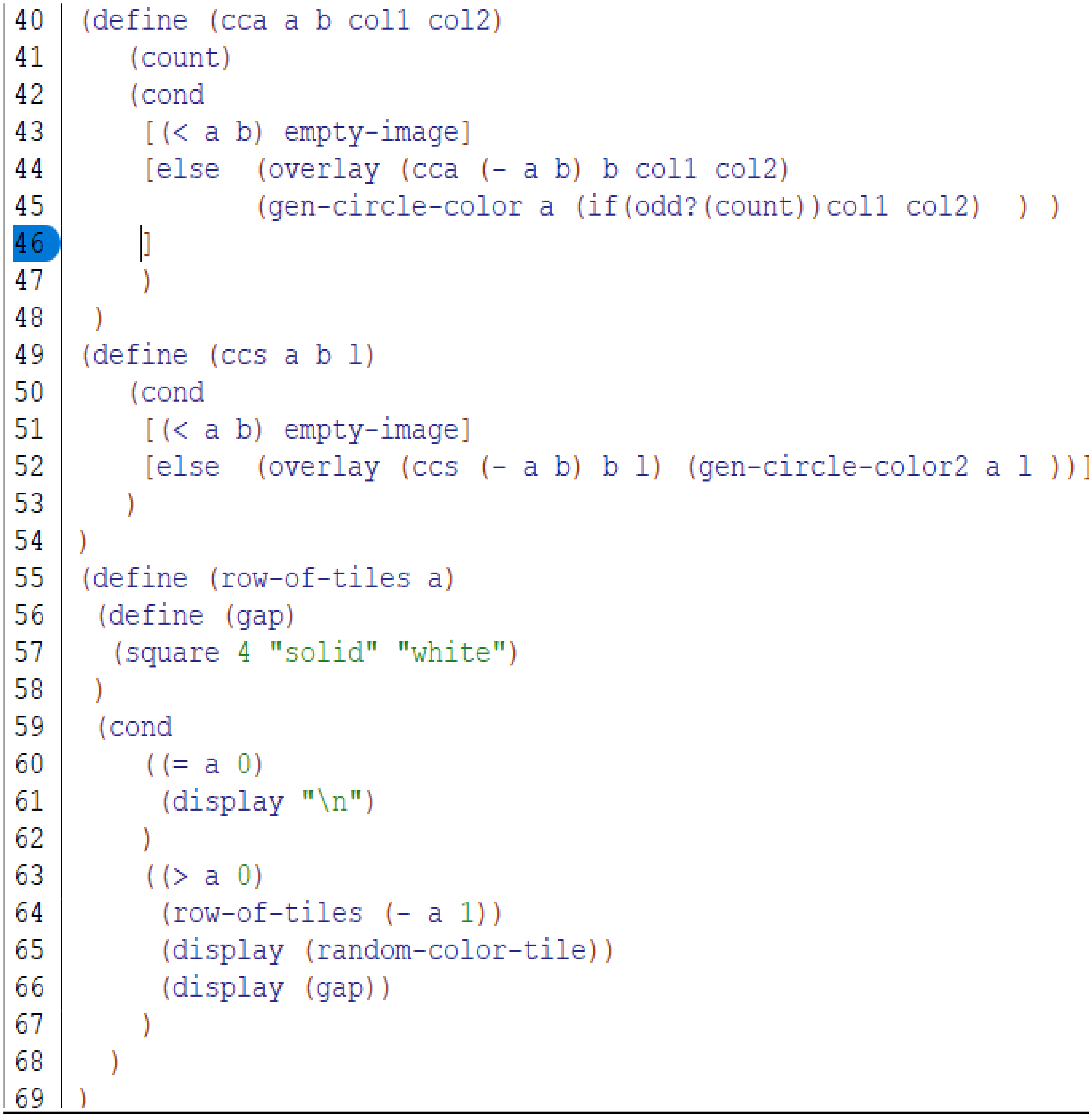


```
>
```

Code



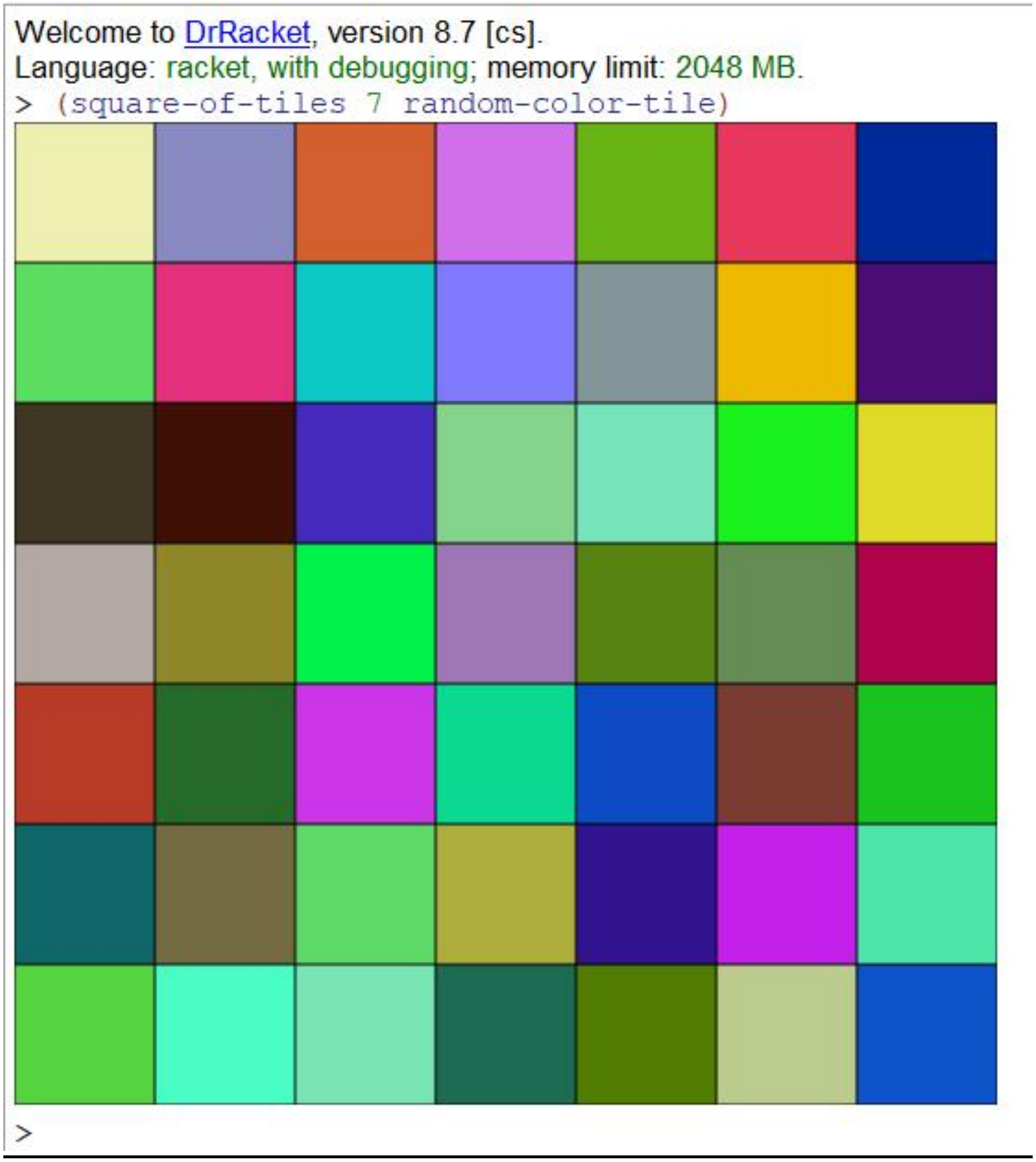
```racket
#lang racket
(require 2htdp/image)
(require counter)
(define red "red")
(define yellow "yellow")
(define blue "blue")
(define brown "brown")
(define coral "coral")
(define goldenrod "goldenrod")
(define olive "olive")
(define tan "tan")
(define l (list red yellow blue))

(define (get-color x)
  (list-ref x (random 0 (length x) ))
  )

(define (rgb)(random 0 256))
(define (rColor) (color (rgb)(rgb)(rgb)))
(define count(make-counter 0))

(define (gen-circle r)
  (circle r "solid" (rColor))
)
(define (gen-circle-color r c)
  (circle r "solid" c)
)
(define (gen-circle-color2 r c)
 (circle r "solid"  (get-color c))
)
(define (random-color-tile)
  (square 75 "solid" (rColor))
  )
```



```
(define (cca a b col1 col2)
   (count)
   (cond
    [(< a b) empty-image]
    [else  (overlay (cca (- a b) b col1 col2)
           (gen-circle-color a (if(odd?(count))col1 col2)  ) )
    ]
    )
 )
(define (ccs a b l)
   (cond
    [(< a b) empty-image]
    [else  (overlay (ccs (- a b) b l) (gen-circle-color2 a l ))]
   )
)
(define (row-of-tiles a)
 (define (gap)
  (square 4 "solid" "white")
 )
 (cond
    ((= a 0)
     (display "\n")
    )
    ((> a 0)
     (row-of-tiles (- a 1))
     (display (random-color-tile))
     (display (gap))
    )
  )
)
```

Task 5: Variations on Hirst Dots

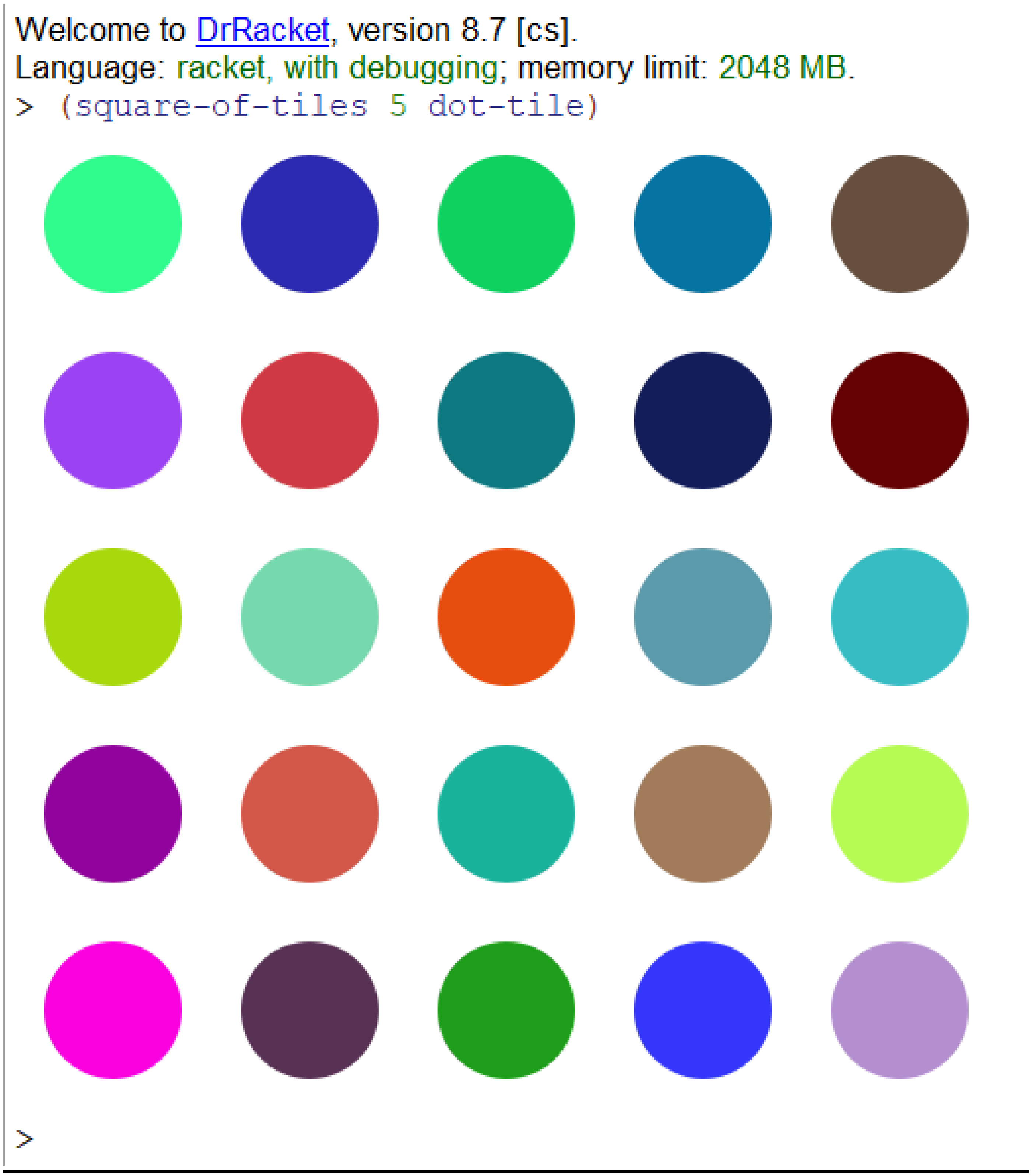
Random Colored Tile Demo



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (square-of-tiles 7 random-color-tile)
```

```
>
```

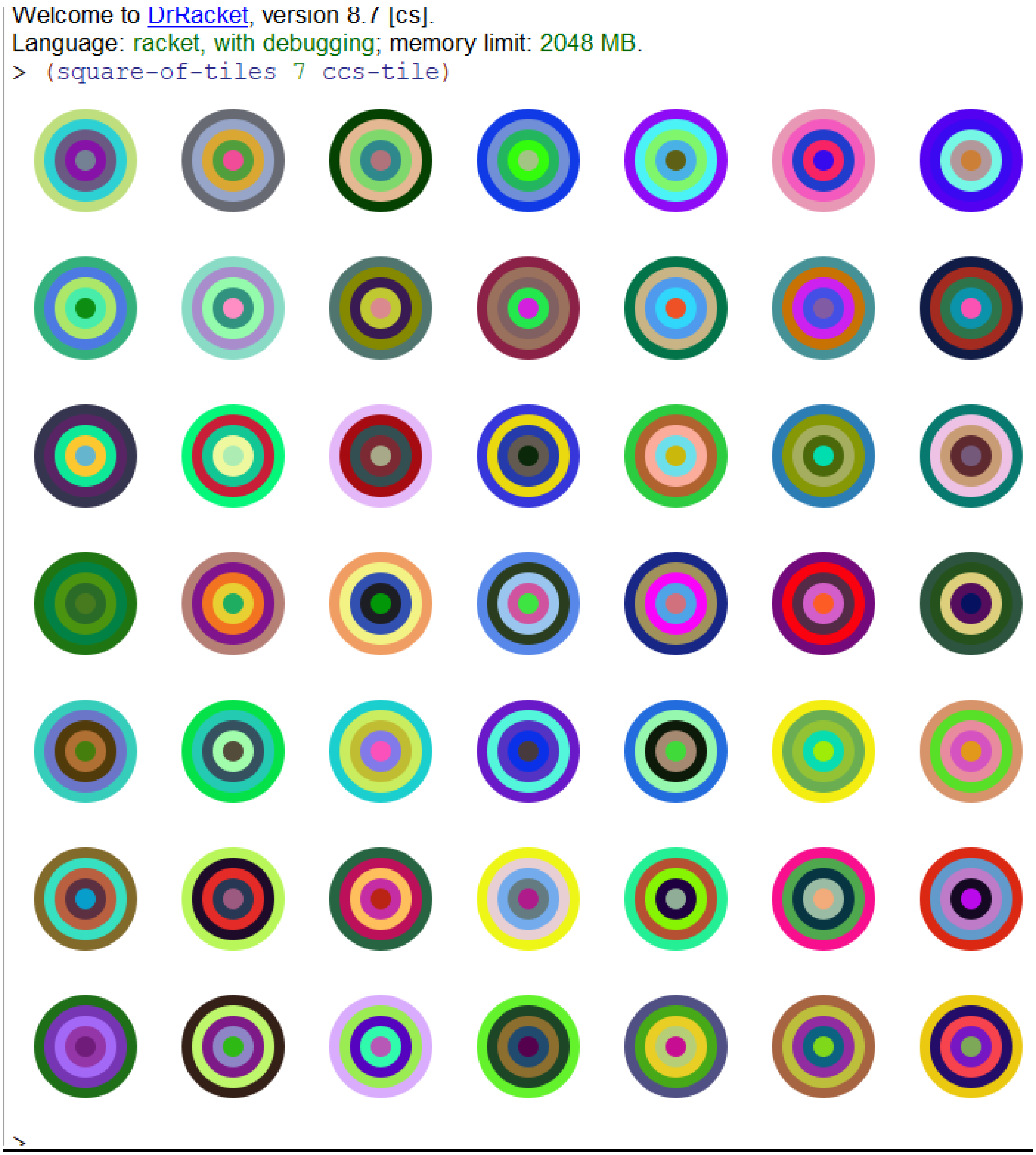
Hirst Dots



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (square-of-tiles 5 dot-tile)
```

```
>
```

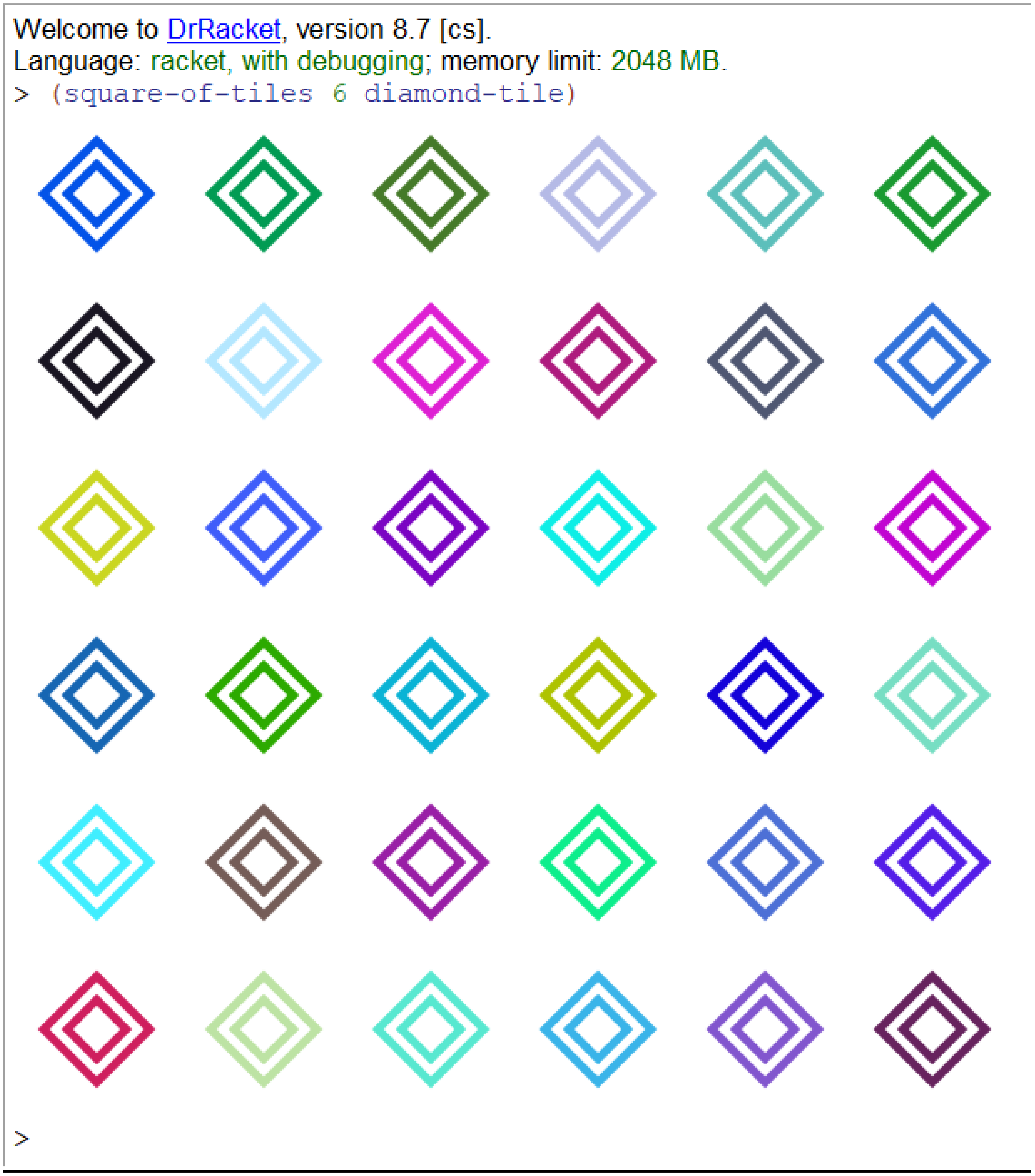
CCS Dots



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (square-of-tiles 7 ccs-tile)
```

```
>
```

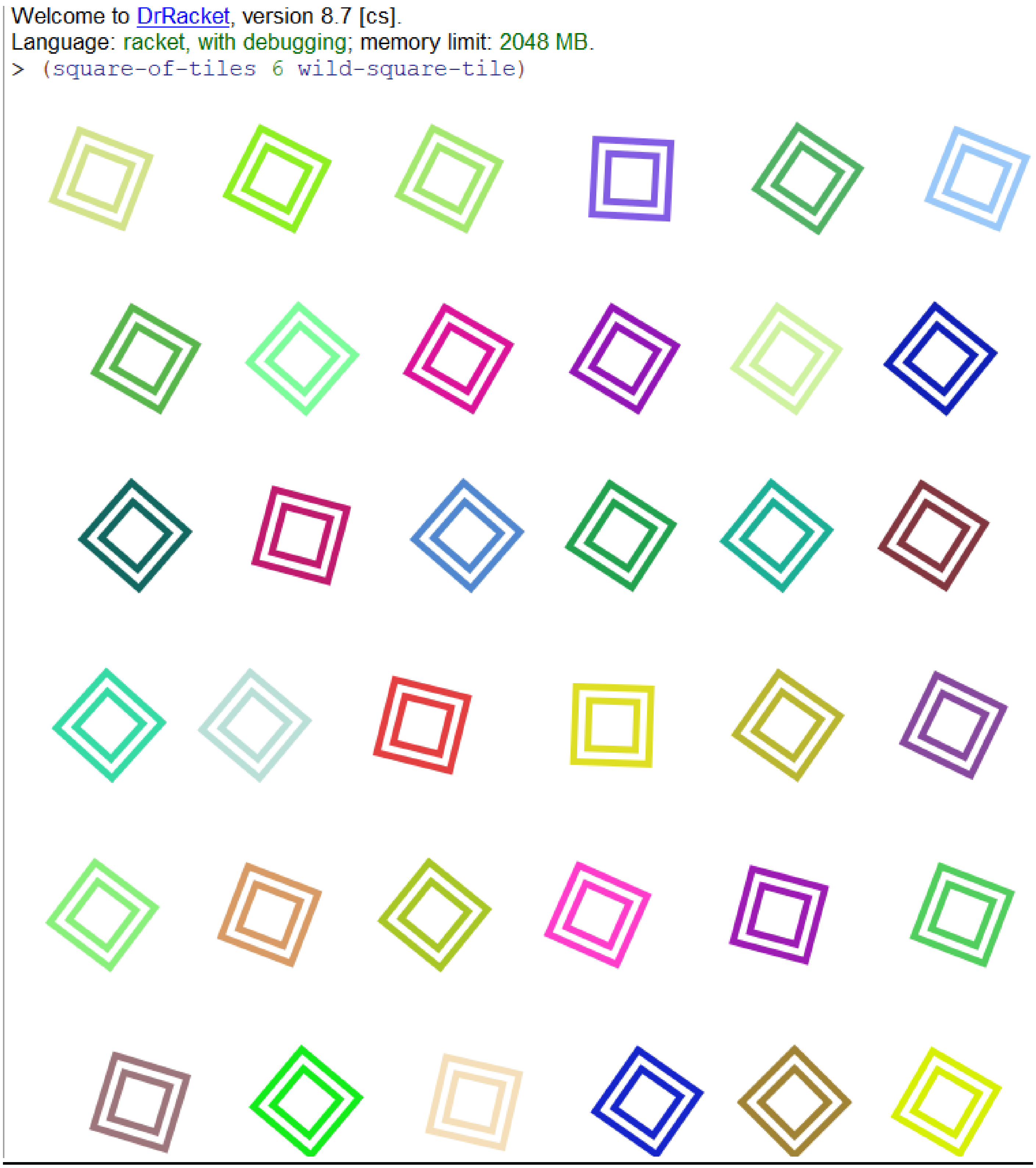
Nested Diamonds



```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (square-of-tiles 6 diamond-tile)
```

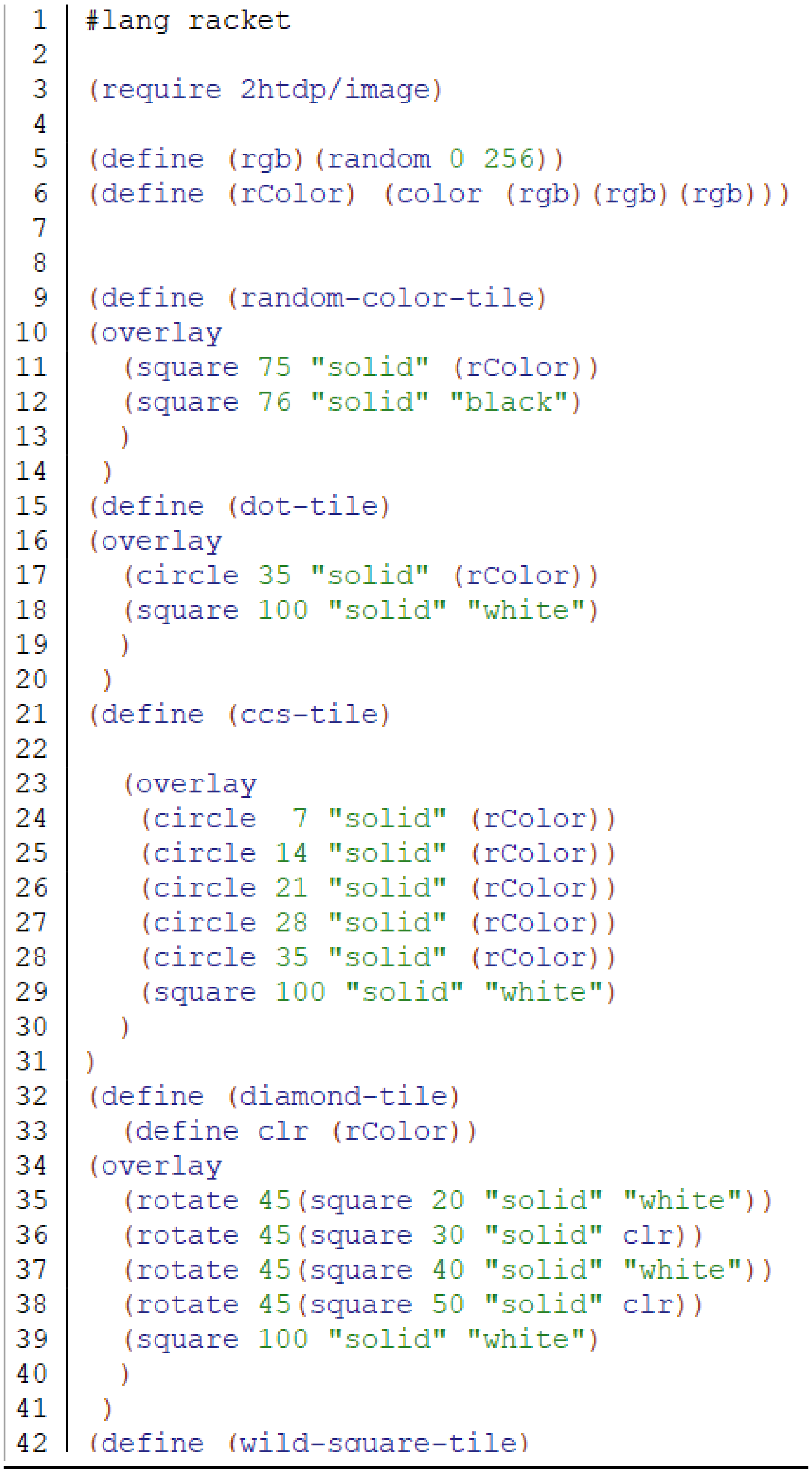
```
>
```

Unruly Squares

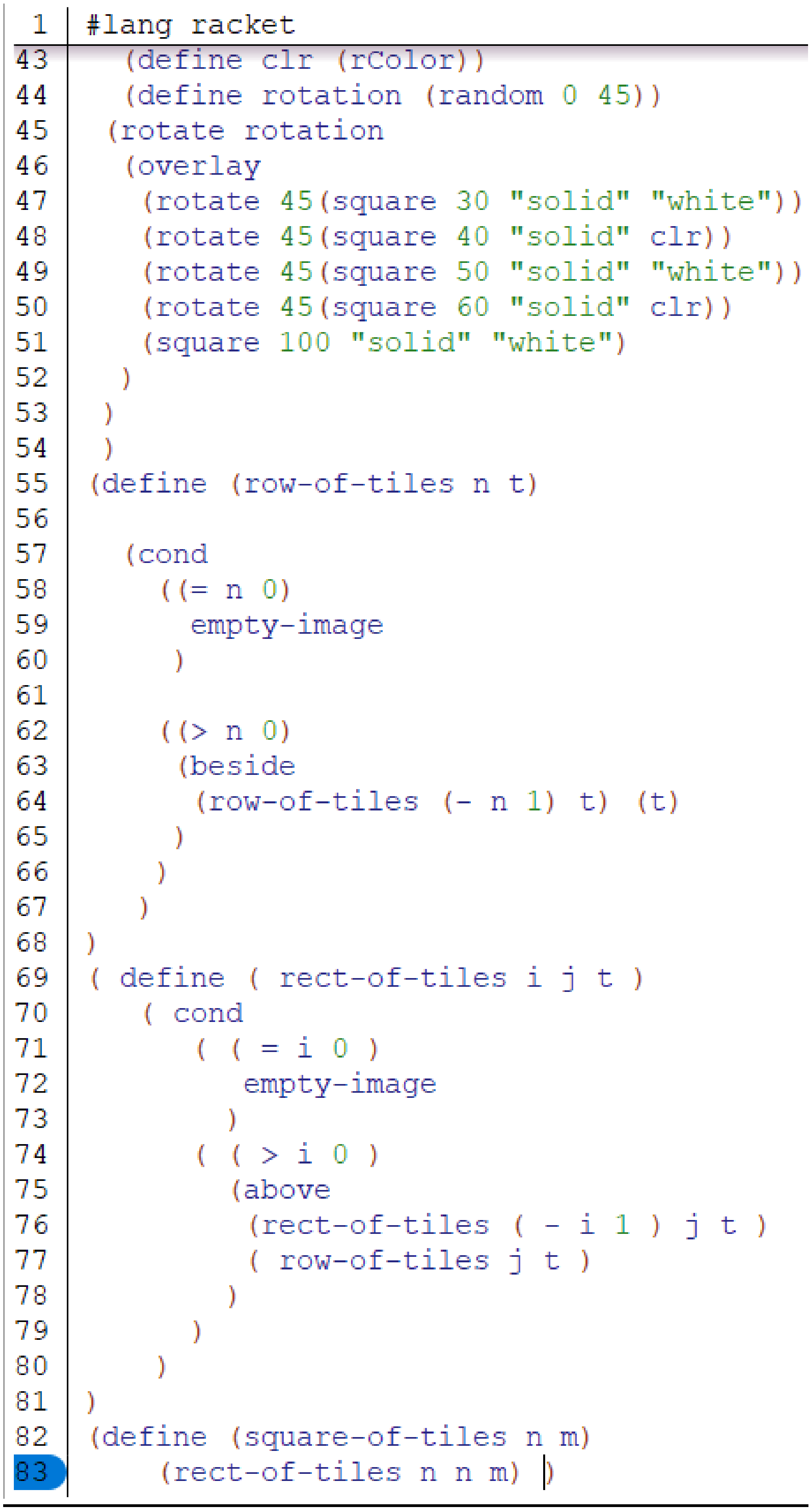


```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 2048 MB.
> (square-of-tiles 6 wild-square-tile)
```

Code



```racket
#lang racket

(require 2htdp/image)

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


(define (random-color-tile)
(overlay
  (square 75 "solid" (rColor))
  (square 76 "solid" "black")
  )
 )
(define (dot-tile)
(overlay
  (circle 35 "solid" (rColor))
  (square 100 "solid" "white")
  )
 )
(define (ccs-tile)

  (overlay
   (circle  7 "solid" (rColor))
   (circle 14 "solid" (rColor))
   (circle 21 "solid" (rColor))
   (circle 28 "solid" (rColor))
   (circle 35 "solid" (rColor))
   (square 100 "solid" "white")
  )
)
(define (diamond-tile)
  (define clr (rColor))
(overlay
  (rotate 45(square 20 "solid" "white"))
  (rotate 45(square 30 "solid" clr))
  (rotate 45(square 40 "solid" "white"))
  (rotate 45(square 50 "solid" clr))
  (square 100 "solid" "white")
  )
 )
(define (wild-square-tile)
```



```racket
#lang racket
  (define clr (rColor))
  (define rotation (random 0 45))
 (rotate rotation
  (overlay
   (rotate 45(square 30 "solid" "white"))
   (rotate 45(square 40 "solid" clr))
   (rotate 45(square 50 "solid" "white"))
   (rotate 45(square 60 "solid" clr))
   (square 100 "solid" "white")
  )
 )
 )
(define (row-of-tiles n t)

  (cond
    ((= n 0)
      empty-image
     )

    ((> n 0)
     (beside
      (row-of-tiles (- n 1) t) (t)
     )
    )
   )
)
( define ( rect-of-tiles i j t )
   ( cond
      ( ( = i 0 )
         empty-image
        )
      ( ( > i 0 )
        (above
         (rect-of-tiles ( - i 1 ) j t )
         ( row-of-tiles j t )
        )
      )
    )
)
(define (square-of-tiles n m)
    (rect-of-tiles n n m) )
```