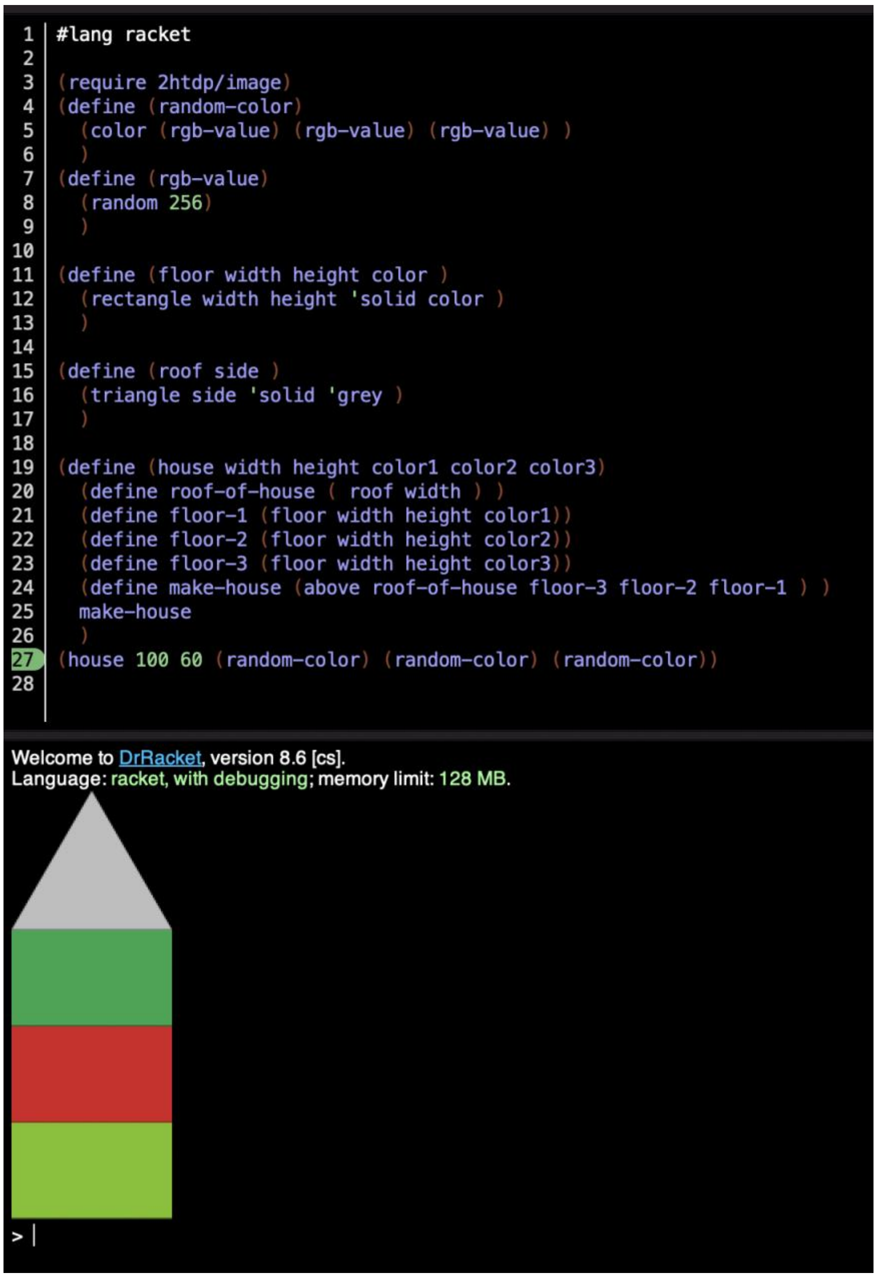
Racket Programming Assignment#2: Racket Functions and Recursion (by Aaroha Sapkota)

Learning Abstract

This assignment features programs that generate images in the context of the 2htdp/image library, most of which are recursive in nature.

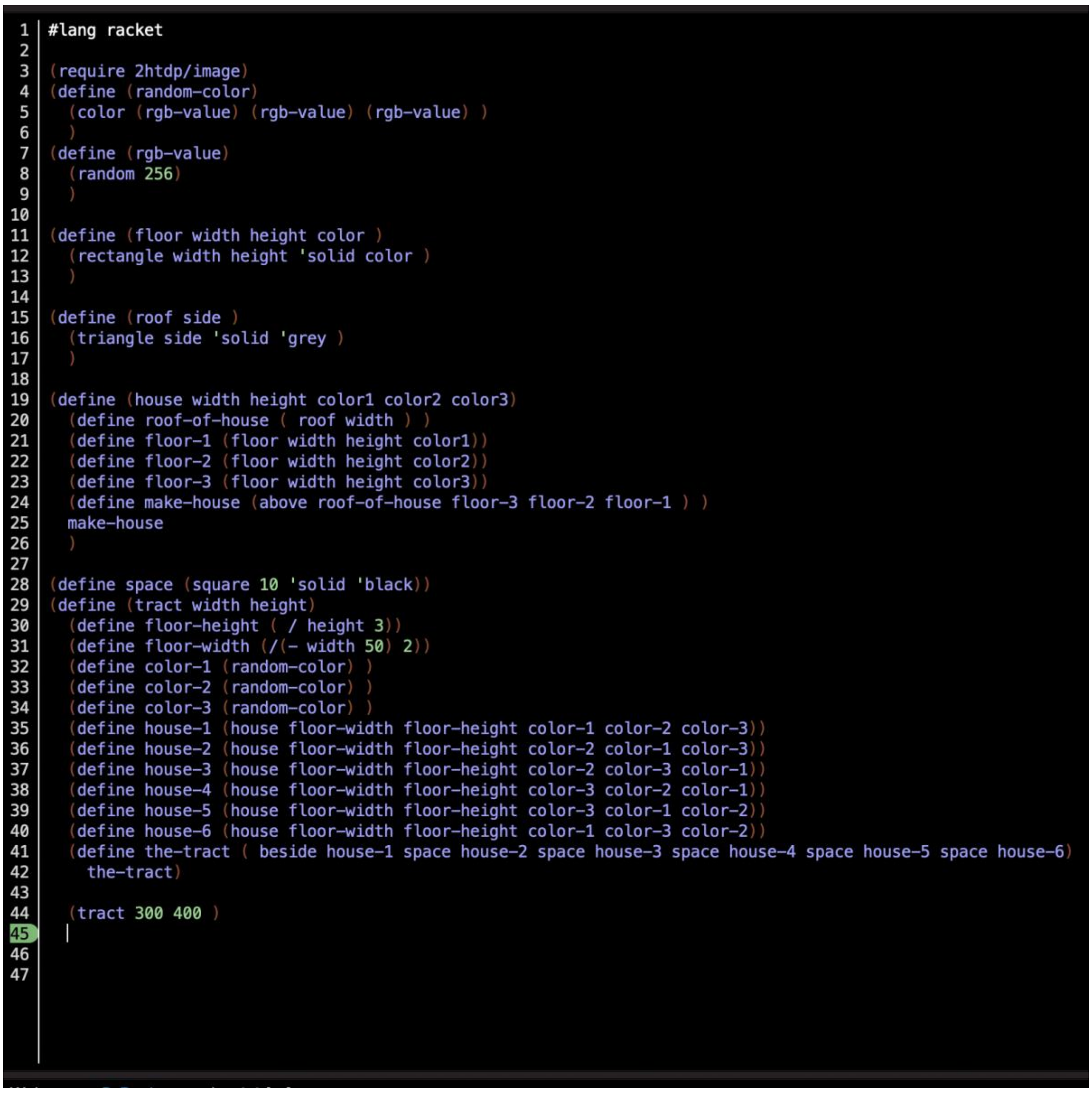
Task 1 : Colorful Permutations of Tract Houses



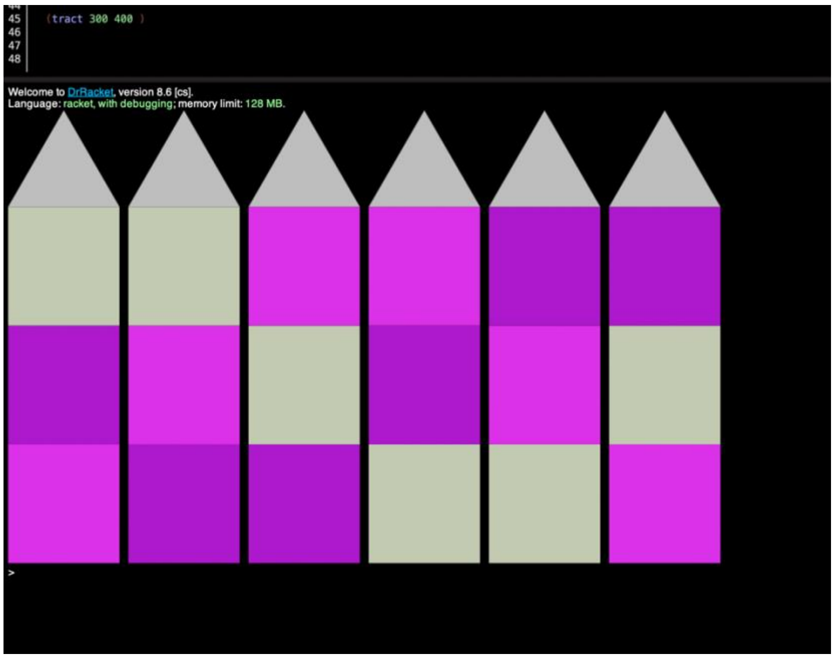
```
#lang racket

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

(define (floor width height color )
  (rectangle width height 'solid color )
  )

(define (roof side )
  (triangle side 'solid 'grey )
  )

(define (house width height color1 color2 color3)
  (define roof-of-house ( roof width ) )
  (define floor-1 (floor width height color1))
  (define floor-2 (floor width height color2))
  (define floor-3 (floor width height color3))
  (define make-house (above roof-of-house floor-3 floor-2 floor-1 ) )
  make-house
  )
(house 100 60 (random-color) (random-color) (random-color))
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
```

```
> |
```



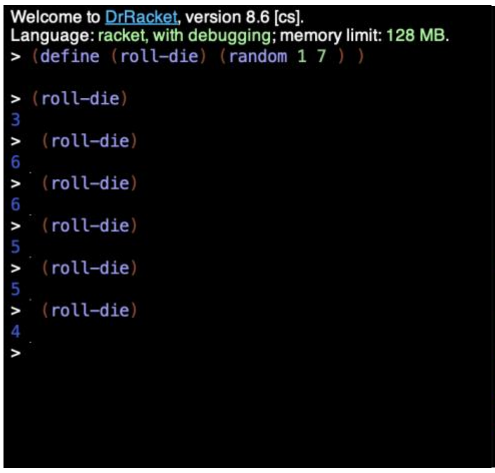
```racket
#lang racket

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

(define (floor width height color )
  (rectangle width height 'solid color )
  )

(define (roof side )
  (triangle side 'solid 'grey )
  )

(define (house width height color1 color2 color3)
  (define roof-of-house ( roof width ) )
  (define floor-1 (floor width height color1))
  (define floor-2 (floor width height color2))
  (define floor-3 (floor width height color3))
  (define make-house (above roof-of-house floor-3 floor-2 floor-1 ) )
  make-house
  )

(define space (square 10 'solid 'black))
(define (tract width height)
  (define floor-height ( / height 3))
  (define floor-width (/(- width 50) 2))
  (define color-1 (random-color) )
  (define color-2 (random-color) )
  (define color-3 (random-color) )
  (define house-1 (house floor-width floor-height color-1 color-2 color-3))
  (define house-2 (house floor-width floor-height color-2 color-1 color-3))
  (define house-3 (house floor-width floor-height color-2 color-3 color-1))
  (define house-4 (house floor-width floor-height color-3 color-2 color-1))
  (define house-5 (house floor-width floor-height color-3 color-1 color-2))
  (define house-6 (house floor-width floor-height color-1 color-3 color-2))
  (define the-tract ( beside house-1 space house-2 space house-3 space house-4 space house-5 space house-6)
    the-tract)

  (tract 300 400 )
```



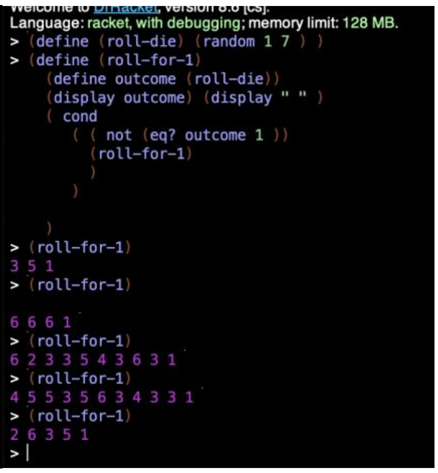
```
  (tract 300 400 )
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
>
```

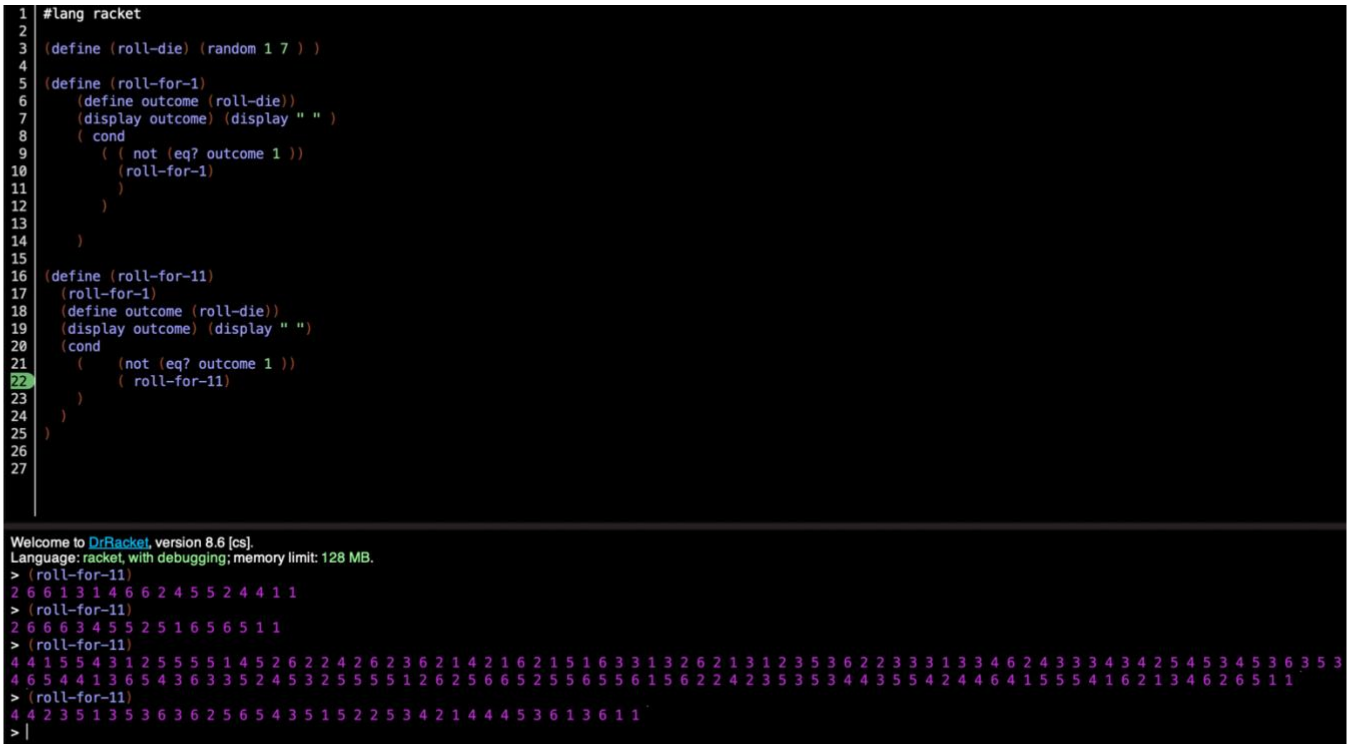
Task 2: Dice



```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define (roll-die) (random 1 7 ) )

> (roll-die)
3
>  (roll-die)
6
>  (roll-die)
6
>  (roll-die)
5
>  (roll-die)
5
>  (roll-die)
4
>
```

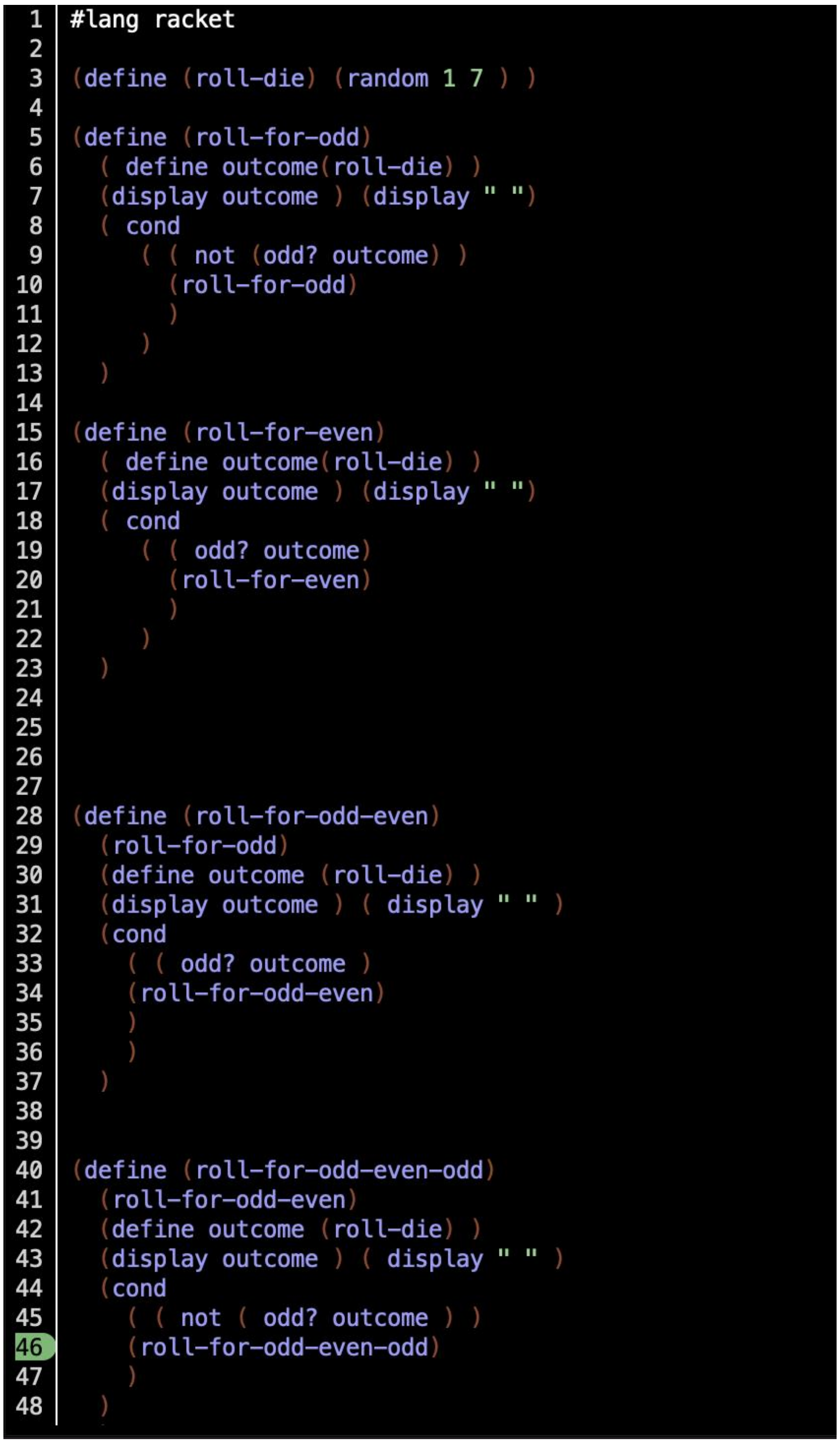


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (define (roll-die) (random 1 7 ) )
> (define (roll-for-1)
    (define outcome (roll-die))
    (display outcome) (display " " )
    ( cond
       ( ( not (eq? outcome 1 ))
         (roll-for-1)
         )
       )


    )
> (roll-for-1)
3 5 1
> (roll-for-1)

6 6 6 1
> (roll-for-1)
6 2 3 3 5 4 3 6 3 1
> (roll-for-1)
4 5 5 3 5 6 3 4 3 3 1
> (roll-for-1)
2 6 3 5 1
> |
```

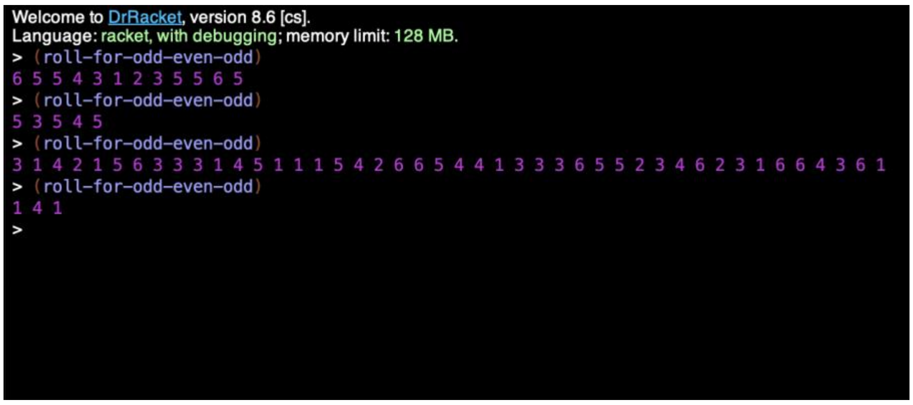


```
#lang racket

(define (roll-die) (random 1 7 ) )

(define (roll-for-1)
    (define outcome (roll-die))
    (display outcome) (display " " )
    ( cond
       ( ( not (eq? outcome 1 ))
         (roll-for-1)
         )
       )


    )

(define (roll-for-11)
  (roll-for-1)
  (define outcome (roll-die))
  (display outcome) (display " ")
  (cond
    (     (not (eq? outcome 1 ))
          ( roll-for-11)
    )
  )
)


```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (roll-for-11)
2 6 6 1 3 1 4 6 6 2 4 5 5 2 4 4 1 1
> (roll-for-11)
2 6 6 6 3 4 5 5 2 5 1 6 5 6 5 1 1
> (roll-for-11)
4 4 1 5 5 4 3 1 2 5 5 5 5 1 4 5 2 6 2 2 4 2 6 2 3 6 2 1 4 2 1 6 2 1 5 1 6 3 3 1 3 2 6 2 1 3 1 2 3 5 3 6 2 2 3 3 3 1 3 3 4 6 2 4 3 3 3 4 3 4 2 5 4 5 3 4 5 3 6 3 5 3
4 6 5 4 4 1 3 6 5 4 3 6 3 3 5 2 4 5 3 2 5 5 5 5 1 2 6 2 5 6 6 5 2 5 5 6 5 5 6 1 5 6 2 2 4 2 3 5 3 5 3 4 4 3 5 5 4 2 4 4 6 4 1 5 5 5 4 1 6 2 1 3 4 6 2 6 5 1 1
> (roll-for-11)
4 4 2 3 5 1 3 5 3 6 3 6 2 5 6 5 4 3 5 1 5 2 2 5 3 4 2 1 4 4 4 5 3 6 1 3 6 1 1
> |
```

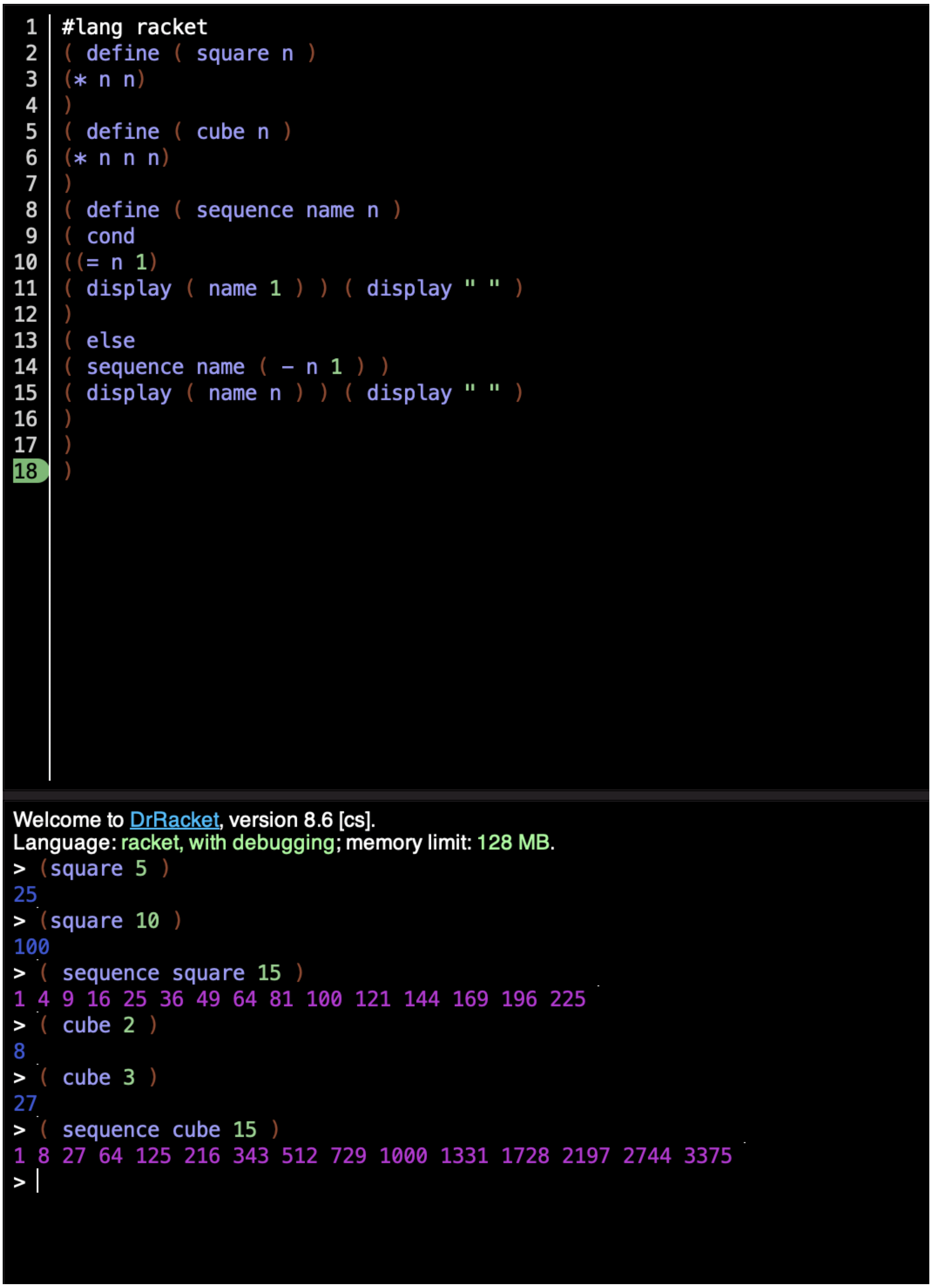


```racket
#lang racket

(define (roll-die) (random 1 7 ) )

(define (roll-for-odd)
  ( define outcome(roll-die) )
  (display outcome ) (display " ")
  ( cond
     ( ( not (odd? outcome) )
       (roll-for-odd)
       )
     )
  )

(define (roll-for-even)
  ( define outcome(roll-die) )
  (display outcome ) (display " ")
  ( cond
     ( ( odd? outcome)
       (roll-for-even)
       )
     )
  )



(define (roll-for-odd-even)
  (roll-for-odd)
  (define outcome (roll-die) )
  (display outcome ) ( display " " )
  (cond
    ( ( odd? outcome )
    (roll-for-odd-even)
    )
    )
  )


(define (roll-for-odd-even-odd)
  (roll-for-odd-even)
  (define outcome (roll-die) )
  (display outcome ) ( display " " )
  (cond
    ( ( not ( odd? outcome ) )
    (roll-for-odd-even-odd)
    )
  )
```



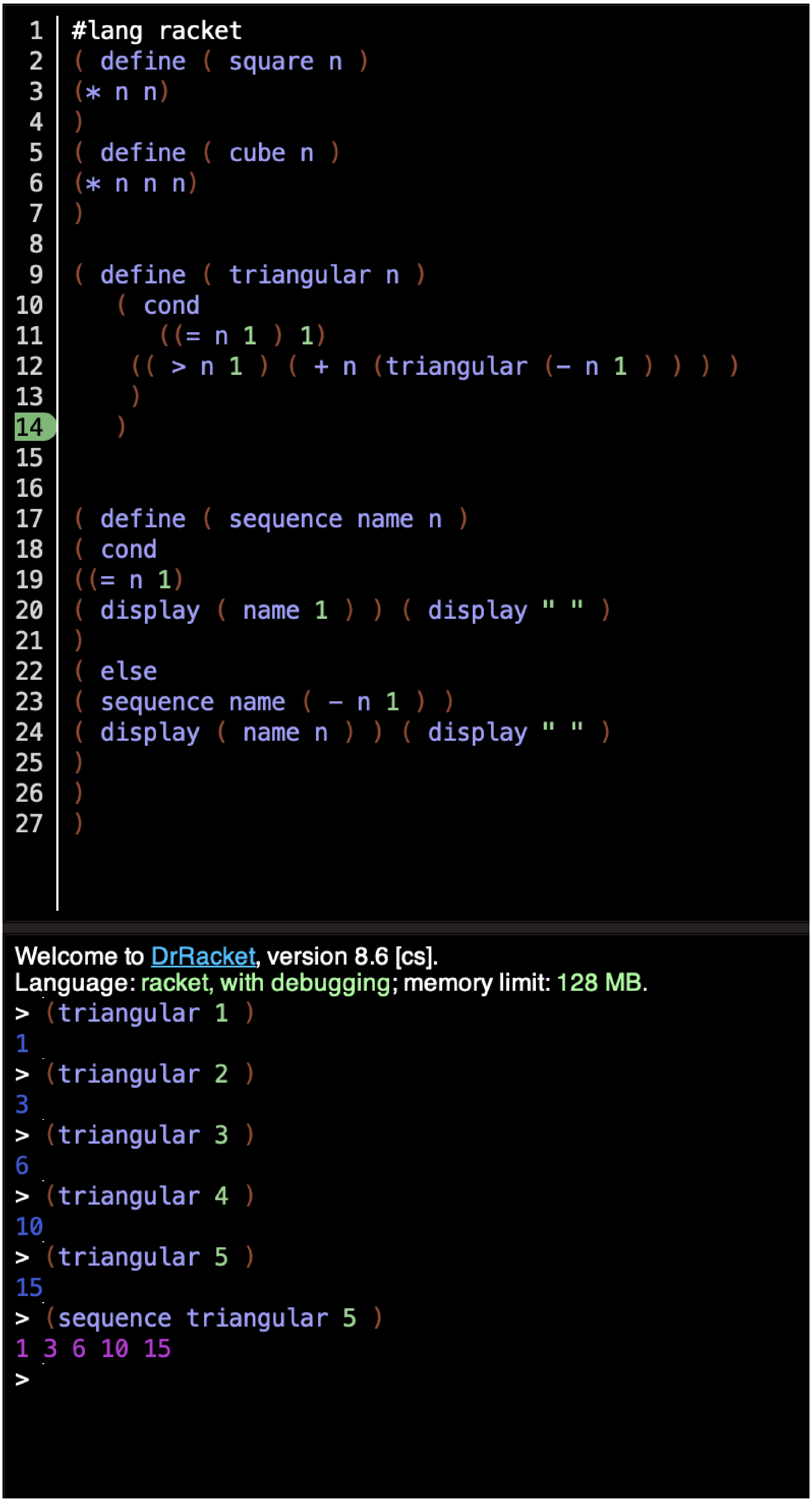
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (roll-for-odd-even-odd)
6 5 5 4 3 1 2 3 5 5 6 5
> (roll-for-odd-even-odd)
5 3 5 4 5
> (roll-for-odd-even-odd)
3 1 4 2 1 5 6 3 3 3 1 4 5 1 1 1 5 4 2 6 6 5 4 4 1 3 3 3 6 5 5 2 3 4 6 2 3 1 6 6 4 3 6 1
> (roll-for-odd-even-odd)
1 4 1
>
```

Task 3: Number Sequences



```
#lang racket
( define ( square n )
(* n n)
)
( define ( cube n )
(* n n n)
)
( define ( sequence name n )
( cond
((= n 1)
( display ( name 1 ) ) ( display " " )
)
( else
( sequence name ( - n 1 ) )
( display ( name n ) ) ( display " " )
)
)
)
```

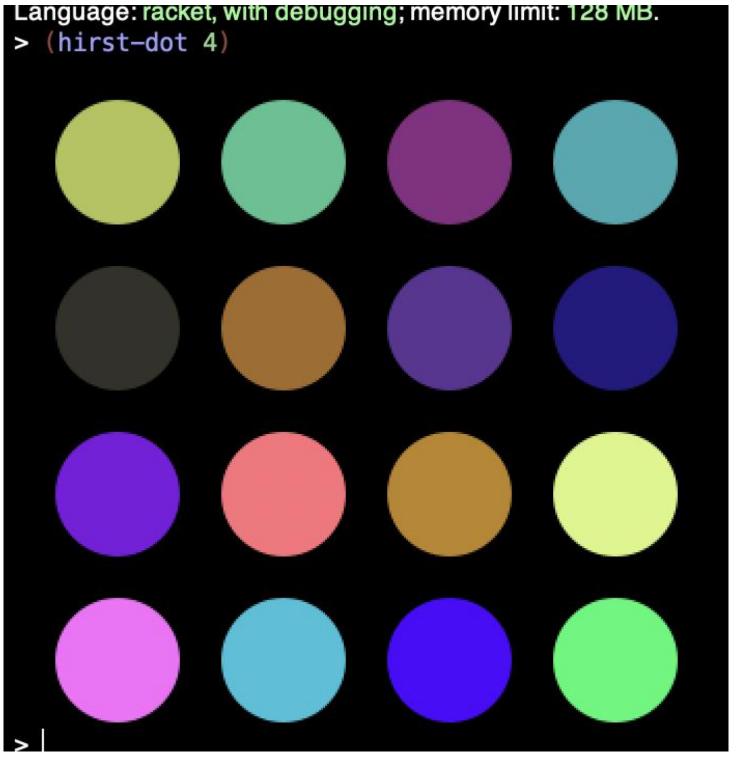
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (square 5 )
25
> (square 10 )
100
> ( sequence square 15 )
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225
> ( cube 2 )
8
> ( cube 3 )
27
> ( sequence cube 15 )
1 8 27 64 125 216 343 512 729 1000 1331 1728 2197 2744 3375
> |
```



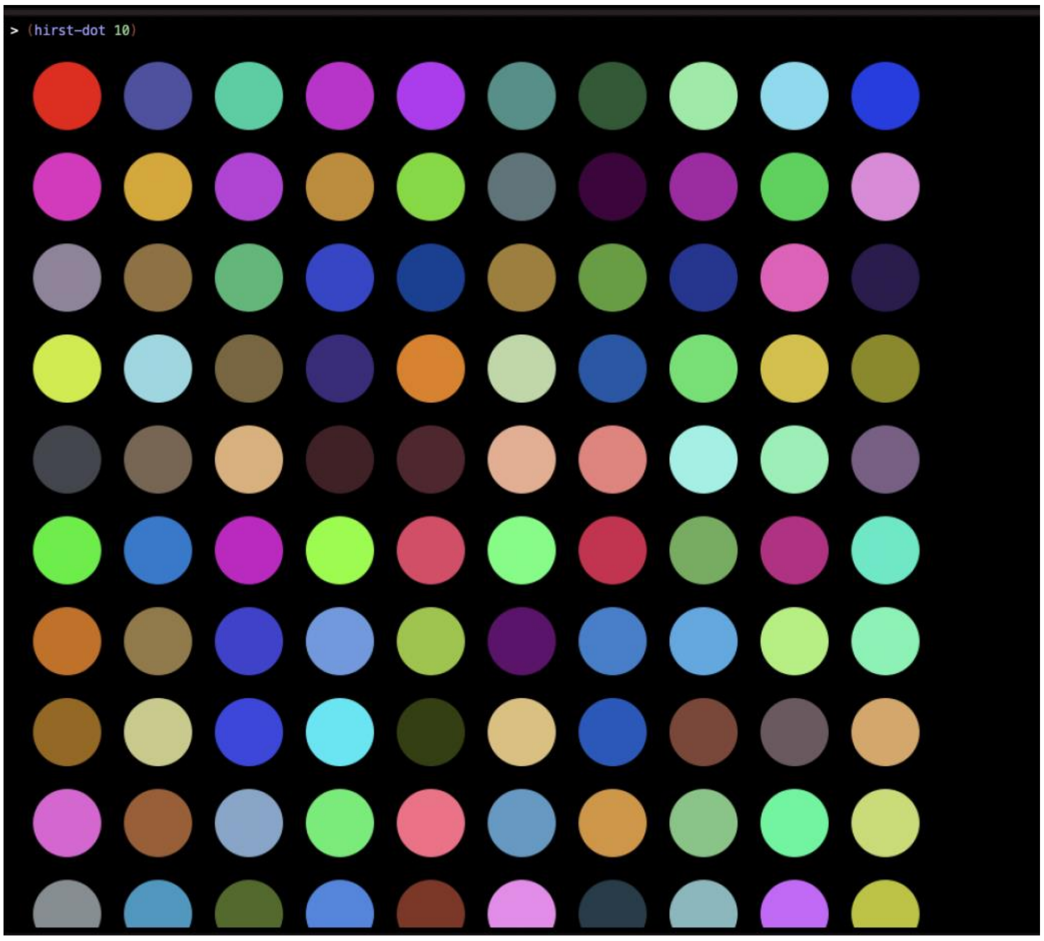
```
#lang racket
( define ( square n )
(* n n)
)
( define ( cube n )
(* n n n)
)

( define ( triangular n )
   ( cond
      ((= n 1 ) 1)
    (( > n 1 ) ( + n (triangular (- n 1 ) ) ) )
    )
   )


( define ( sequence name n )
( cond
((= n 1)
( display ( name 1 ) ) ( display " " )
)
( else
( sequence name ( - n 1 ) )
( display ( name n ) ) ( display " " )
)
)
)
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (triangular 1 )
1
> (triangular 2 )
3
> (triangular 3 )
6
> (triangular 4 )
10
> (triangular 5 )
15
> (sequence triangular 5 )
1 3 6 10 15
>
```

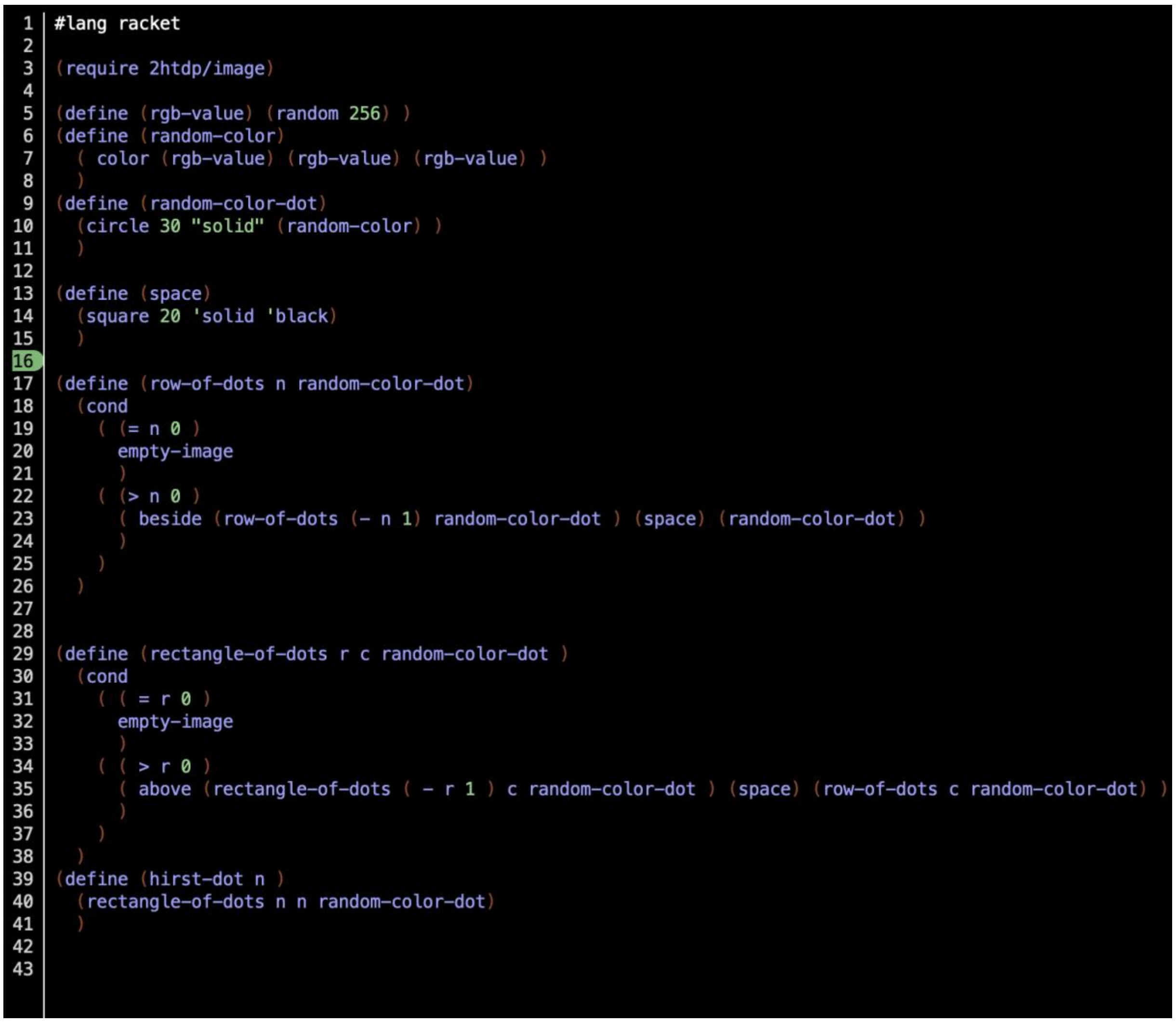
Task4: Hirst Dots



```
Language: racket, with debugging; memory limit: 128 MB.
> (hirst-dot 4)
>
```

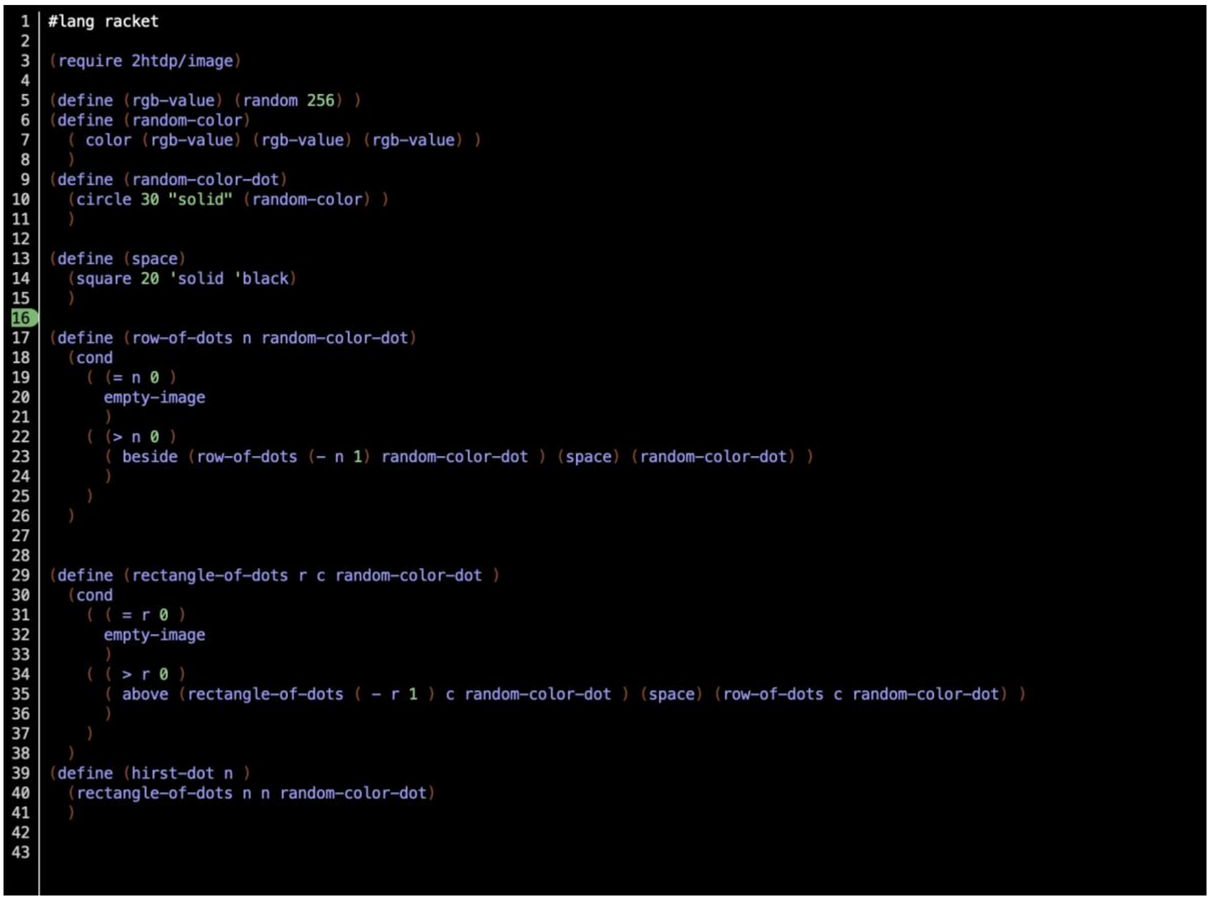


```
> (hirst-dot 10)
```



```racket
#lang racket

(require 2htdp/image)

(define (rgb-value) (random 256) )
(define (random-color)
  ( color (rgb-value) (rgb-value) (rgb-value) )
  )
(define (random-color-dot)
  (circle 30 "solid" (random-color) )
  )

(define (space)
  (square 20 'solid 'black)
  )

(define (row-of-dots n random-color-dot)
  (cond
    ( (= n 0 )
      empty-image
      )
    ( (> n 0 )
      ( beside (row-of-dots (- n 1) random-color-dot ) (space) (random-color-dot) )
      )
    )
  )


(define (rectangle-of-dots r c random-color-dot )
  (cond
    ( ( = r 0 )
      empty-image
      )
    ( ( > r 0 )
      ( above (rectangle-of-dots ( - r 1 ) c random-color-dot ) (space) (row-of-dots c random-color-dot) )
      )
    )
  )
(define (hirst-dot n )
  (rectangle-of-dots n n random-color-dot)
  )
```

Task5: Channeling Frank Stella



```racket
#lang racket

(require 2htdp/image)

(define (rgb-value) (random 256) )
(define (random-color)
  ( color (rgb-value) (rgb-value) (rgb-value) )
  )
(define (random-color-dot)
  (circle 30 "solid" (random-color) )
  )

(define (space)
  (square 20 'solid 'black)
  )

(define (row-of-dots n random-color-dot)
  (cond
    ( (= n 0 )
      empty-image
      )
    ( (> n 0 )
      ( beside (row-of-dots (- n 1) random-color-dot ) (space) (random-color-dot) )
      )
    )
  )


(define (rectangle-of-dots r c random-color-dot )
  (cond
    ( ( = r 0 )
      empty-image
      )
    ( ( > r 0 )
      ( above (rectangle-of-dots ( - r 1 ) c random-color-dot ) (space) (row-of-dots c random-color-dot) )
      )
    )
  )
(define (hirst-dot n )
  (rectangle-of-dots n n random-color-dot)
  )


```