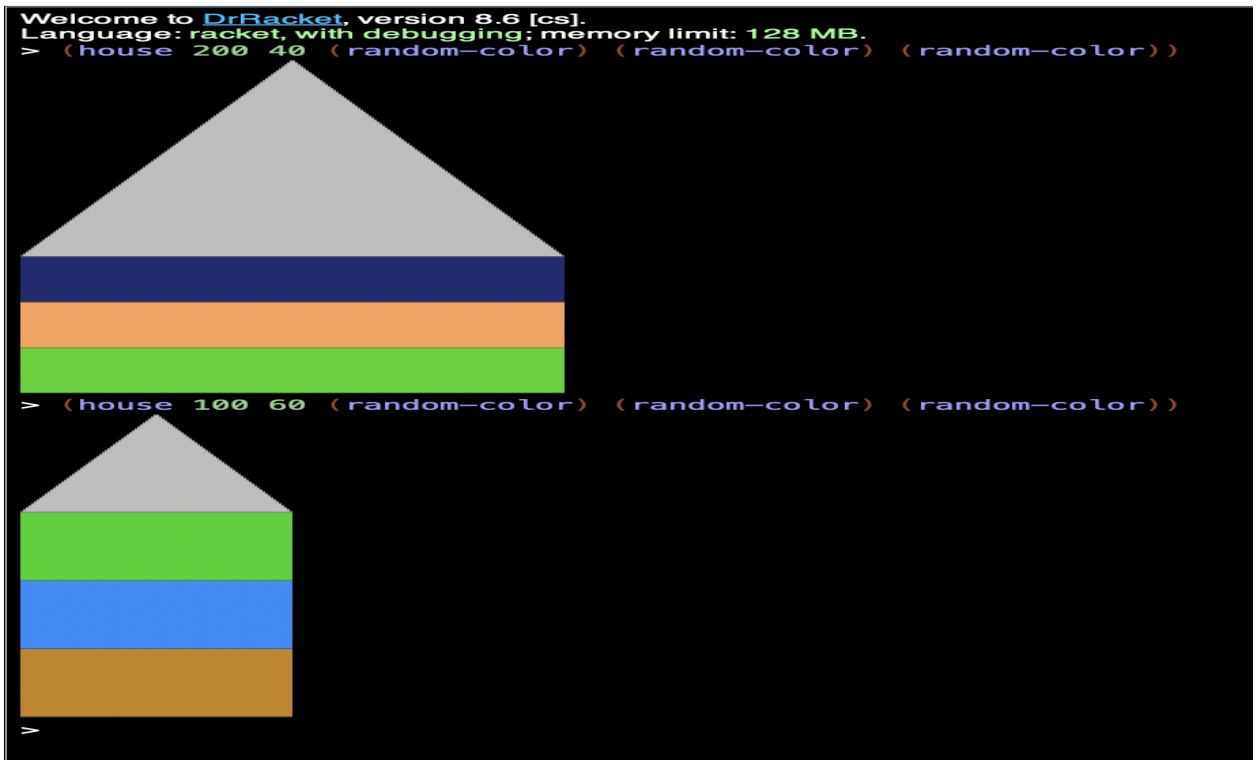
Racket Programming Assignment #2: Racket Functions and Recursions

Abstract:

Racket images and recursions. Write a program to display images of colorful tract houses that are grouped by the concept of permutation. Write some programs to do things with virtual dice. Write a couple of programs pertaining to classical number sequences. Write a program to display a grid of Hirst dots. Write a program to display images which channel Frank Stella. Complete a program to display a set of dominos. Programmably create an image that uses bits of functionality from the 2htdp/image library which was featured in class.

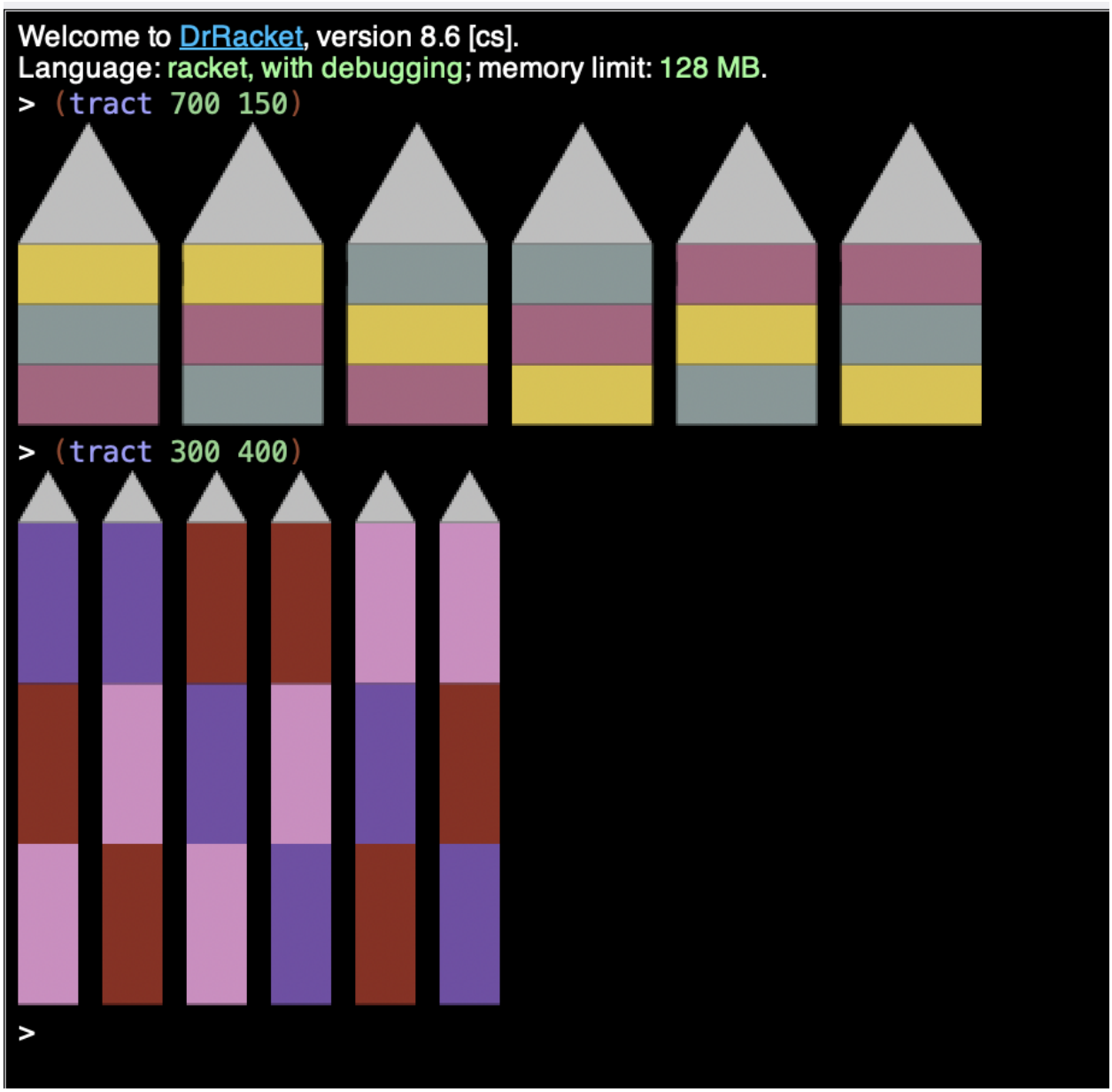
Colorful Permutation of Tract Houses:

The House Demo



Colorful Permutation of Tract Houses:

The Tract Demo



Colorful Permutation of Tract Houses:

The House and Tract Code

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

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

; House code-------------------------------------------------------------------------------
( define (house width height color1 color2 color3)
( define ( first-floor width height color1) ( rectangle width height "solid" color1 ))
( define ( second-floor width height color2) ( rectangle width height "solid" color2 ))
( define ( third-floor width height color3) ( rectangle width height "solid" color3 ))
( define ( roof width) ( triangle width "solid" "grey"))
( define house-stack ( above (roof width) (first-floor width height color1)
                             (second-floor width height color2)
                             (third-floor width height color3))) house-stack)


; Tract Code-------------------------------------------------------------------------------

(define (house-floors width height) ( rectangle width height "solid" (random-color)))
( define (tract width height)
(define width-of-floors (/ width 12))
(define height-of-floors (/ height 6))
(define seperate ( square 10 "solid" "black"))
(define first-floor ( house-floors width-of-floors height-of-floors))
(define second-floor (  house-floors width-of-floors height-of-floors))
(define third-floor (  house-floors width-of-floors height-of-floors))
( define ( roof) ( triangle width-of-floors "solid" "grey"))
(define house1 ( above (roof) first-floor second-floor third-floor))
(define house2 ( above (roof) first-floor third-floor second-floor))
(define house3 ( above (roof) second-floor first-floor third-floor))
(define house4 ( above (roof) second-floor third-floor first-floor))
(define house5 ( above (roof) third-floor first-floor second-floor))
(define house6 ( above (roof) third-floor second-floor first-floor))
(define all-houses ( beside house1 seperate house2 seperate house3 seperate house4 seperate house5 seperate house6)) all-houses)


```

Dice:

Demo

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

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

Dice:

Code

```
#lang racket
;Rolling dice number 1 to 6--------------------------------
(define ( roll-die) (+ (random 6) 1))

;Roll for 1-------------------------------------------
( define ( roll-for-1 ) (define n (roll-die)) (display n)
   (display " ") (cond((> n 1) (roll-for-1))))

;Roll for 11------------------------------------------------
(define (roll-for-11) (roll-for-1)(define n (roll-die))
(display n) (display " " ) (cond((> n 1)(roll-for-11))))

;Roll for odd-even-odd-----------------------------------
(define (roll-for-even) (define n (roll-die)) (display n)
(display " ") (cond (( not (even? n)) (roll-for-even))))

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

(define (roll-for-odd-even-odd) (roll-for-odd) (roll-for-even)
(roll-for-odd))
;Roll two-dice-for-lucky-pair--------------------------------
(define (roll-two-dice-for-lucky-pair)
  (define n1 (roll-die))
  (define n2 (roll-die))
  (display "(")
  (display n1)
  (display " ")
  (display n2)
  (display ")")
  (define n3 (+ n1 n2))
  (define seven (= n3 7))
  (define eleven (= n3 11))
  (define pair (= n1 n2))
  (cond ( (not (or seven eleven pair)) (roll-two-dice-for-lucky-pair))))

```

Number Sequences:

Preliminary, Triangular, and Sigma Demo

```
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
> (triangular 1)
1
> (triangular 2)
3
> (triangular 3)
6
> (triangular 4)
10
> (triangular 5)
15
> (sequence triangular 20)
1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210
> (sigma 1)
1
> (sigma 2)
3
> (sigma 3)
4
> (sigma 4)
7
> (sigma 5)
6
> (sequence sigma 20)
1 3 4 7 6 12 8 15 13 18 12 28 14 24 24 31 18 39 20 42
>
```

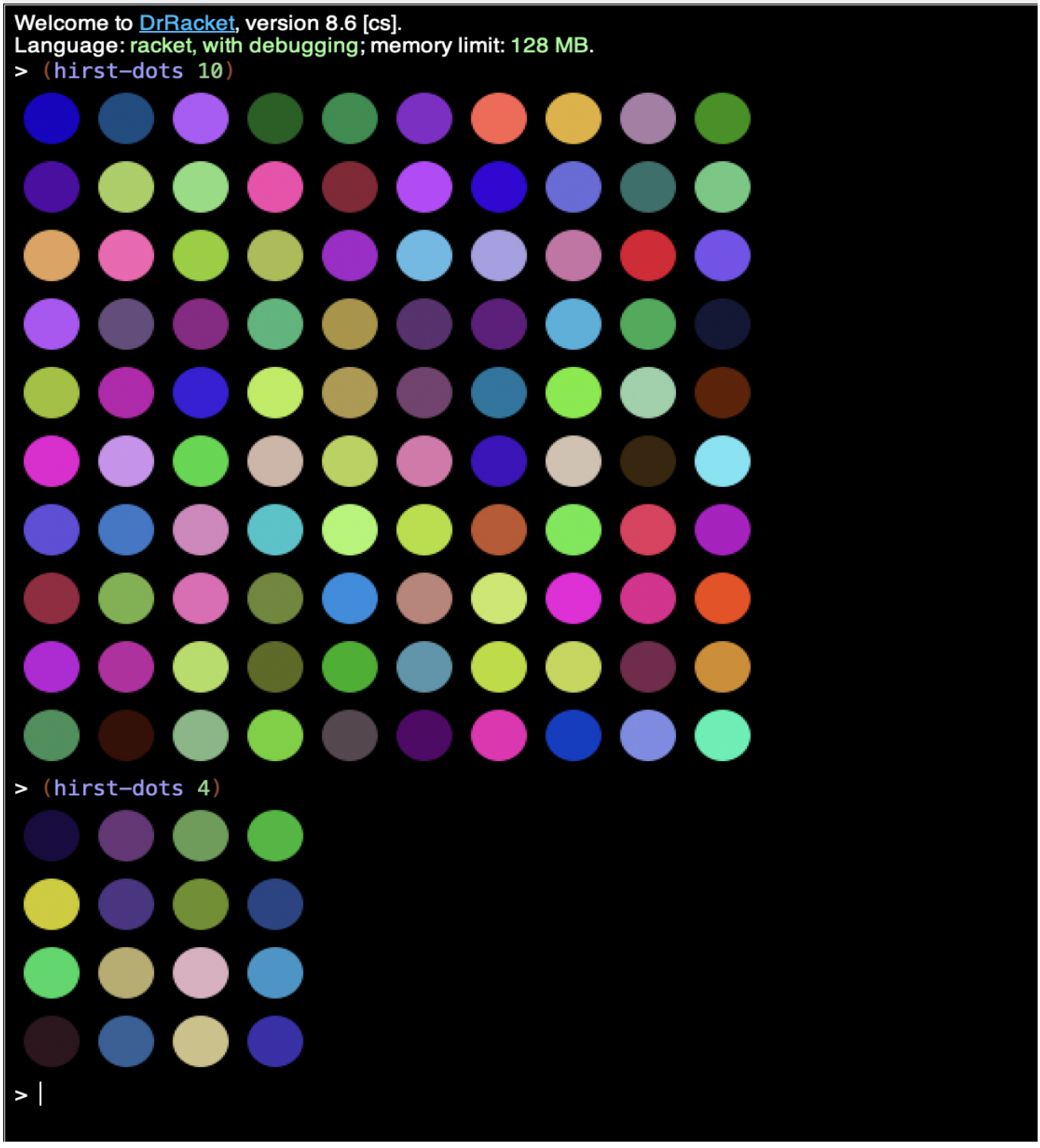
Number Sequences:

Preliminary, Triangular, and Sigma Code

```
#lang racket

;Preliminary Code-------------------------------------
( 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 " " ) )) )
;Triangular Code-----------------------------------------
( define ( triangular n)( cond ( (= n 1) 1)
( (= n 2) 3) ( (> n 2)(+ (triangular (- n 1)) n ))))
;Sigma Code----------------------------------------------
(define (sigma n ) (calculation n (o)))
(define (o) 1)(define (calculation n d)
( cond( (= d n ) d)( ( = ( remainder n d ) 0 )
( + d ( calculation n ( + d 1 ) ) ) ))
( else( calculation n ( + d 1 ))) ))
```

Hirst Dots:

The Demo



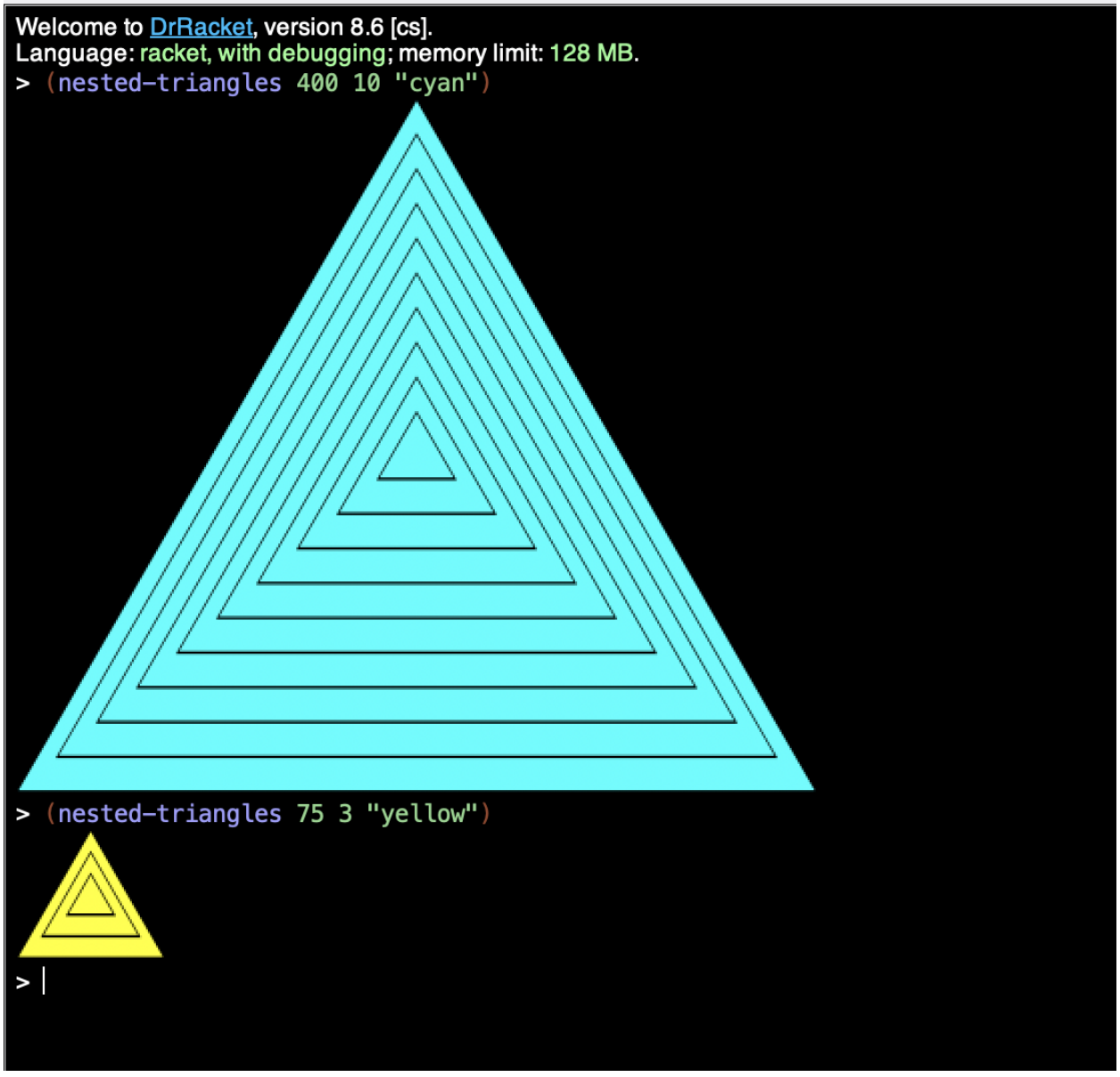
Hirst Dots:

The Code

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

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

( define ( hirst-dots n) ( square-of-tiles n n))
;The pattern-----------------------------------------
( define ( square-of-tiles r c ) ( cond
((= r 0) empty-image)
(( > r 0)( above
( square-of-tiles ( - r 1 ) c ) ( row-of-tiles c) ) )) )
;The tile-------------------------------
(define (tile) (overlay (circle 15 "solid" (random-color))
(square 40 "solid" "black" )))
;The Row---------------------------------
( define ( row-of-tiles n) ( cond
(( = n 0) empty-image)
(( > n 0)
( beside ( row-of-tiles ( - n 1 ) ) (tile) ) )
) )
```

Channeling Frank Stella:

The Demo



Channeling Frank Stella:

The Code

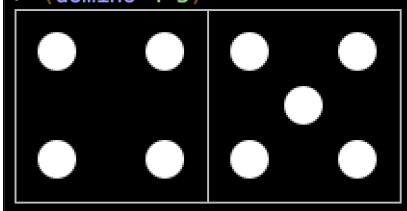
```
#lang racket
( require 2htdp/image )
( define ( nested-triangles side count color ) ( define unit ( / side count ) )
( paint-nested-stars 1 count unit color )
)
( define ( paint-nested-stars from to unit color)
( define side-length ( * from unit ) ) ( cond
( ( = from to )
( framed-star side-length color )
)
( ( < from to )
( overlay
( framed-star side-length color )
( paint-nested-stars ( + from 1 ) to unit color )
) )
) )
( define ( framed-star side-length color ) ( overlay
( triangle ( - side-length 3 ) "solid" color )
( triangle side-length "solid" "black" ) )
)

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

Dominos:

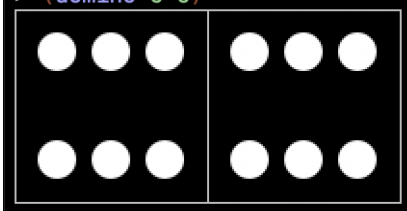
The Demo

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

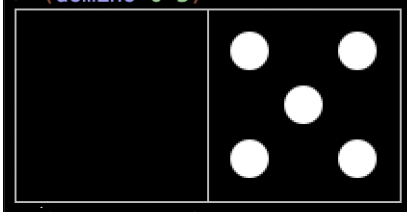
```
> (domino 4 5)
```



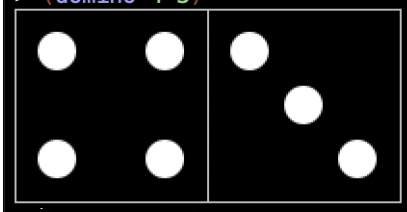
```
> (domino 6 6)
```



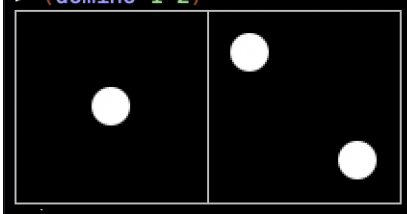
```
> (domino 0 5)
```



```
> (domino 4 3)
```



```
> (domino 1 2)
```



```
>
```

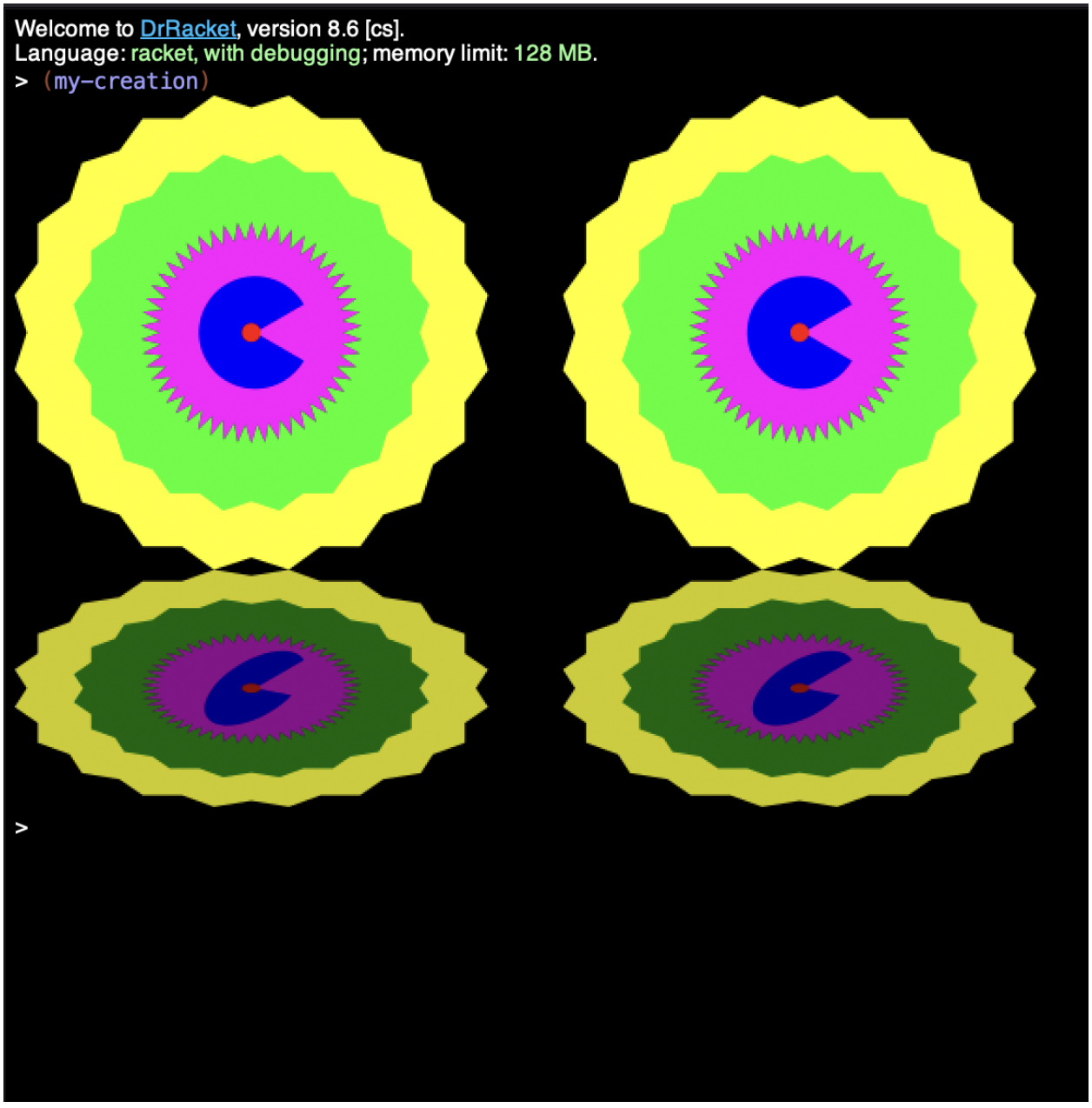
Dominos:

The Code

```
#lang racket
;------------------------------------------------------------------------------ ; Requirements
; - Just the image library from Version 2 of "How to Design Programs"
( require 2htdp/image )
;------------------------------------------------------------------------------
; Problem parameters ;
; - Variables to denote the side of a tile and the dimensions of a pip
( define side-of-tile 100 )
( define diameter-of-pip ( * side-of-tile 0.2 ) )
( define radius-of-pip ( / diameter-of-pip 2 ) )
;------------------------------------------------------------------------------
; Numbers used for offsetting pips from the center of a tile ;
; - d and nd are used as offsets in the overlay/offset function applications
( define d ( * diameter-of-pip 1.4 ) ) ( define nd ( * -1 d ) )
;------------------------------------------------------------------------------
; The blank tile and the pip generator - Bind one variable to a blank tile and another to a pip
( define blank-tile ( square side-of-tile "solid" "black" ) )
( define ( pip ) ( circle radius-of-pip "solid" "white" ) )
;------------------------------------------------------------------------------
; The basic tiles  - Bind one variable to each of the basic tiles
( define basic-tile1 ( overlay ( pip ) blank-tile ) )
( define basic-tile2 ( overlay/offset (pip ) d d ( overlay/offset (pip)  nd nd blank-tile)))
( define basic-tile3( overlay ( pip ) basic-tile2 ) )
( define basic-tile4( overlay/offset (pip ) d d ( overlay/offset (pip)  d nd
(overlay/offset (pip ) nd d ( overlay/offset (pip)  nd nd blank-tile)) )))
( define basic-tile5 ( overlay ( pip ) basic-tile4 blank-tile ) )
( define basic-tile6 ( overlay/offset (pip ) 0 d ( overlay/offset (pip)  0 nd basic-tile4)))
;------------------------------------------------------------------------------
; The framed framed tiles - Bind one variable to each of the six framed tiles
( define frame ( square side-of-tile "outline" "gray" ) )
( define tile0 ( overlay frame blank-tile ) )
( define tile1 ( overlay frame basic-tile1 ) )
( define tile2 ( overlay frame basic-tile2 ) )
( define tile3 ( overlay frame basic-tile3 ) )
( define tile4 ( overlay frame basic-tile4 ) )
( define tile5 ( overlay frame basic-tile5 ) )
( define tile6 ( overlay frame basic-tile6 ) )
;------------------------------------------------------------------------------
; Domino generator  - Funtion to generate a domino
( define ( domino a b )
( beside ( tile a ) ( tile b ) )
)
( define ( tile x ) ( cond

( ( = x 0 ) tile0 )
    ( ( = x 1 ) tile1 )
    ( ( = x 2 ) tile2 )
    ( ( = x 3 ) tile3 )
    ( ( = x 4 ) tile4 )
    ( ( = x 5 ) tile5 )
    ( ( = x 6 ) tile6 )
) )
```

Creation:

The Demo



Creation:

The Code

```
1  #lang racket
2  (require 2htdp/image)
3  (define (my-creation)
4    (define (space) (square 40 "solid" "black"))
5    (above (beside (eye) (space) (eye))
6     (beside(shadow) (space) (shadow))))
7
8  (define (eye) (overlay (circle 5 "solid" "red")
9    (rotate 30 (wedge 30 300 "solid" "blue"))
10   (radial-star 50 50 60 "solid" "magenta")
11   (star-polygon 30 20 3 "solid" "green")
12   (star-polygon 40 20 3 "solid" "yellow") ))
13
14 (define (eye-shadow) (overlay (circle 5 "solid" "dark red")
15   (rotate 30 (wedge 30 300 "solid" "dark blue"))
16   (radial-star 50 50 60 "solid" "dark magenta")
17   (star-polygon 30 20 3 "solid" "dark green")
18   (star-polygon 40 20 3 "solid" "dark yellow") ))
19
20 (define (shadow) (scale/xy 1 1/2 (flip-vertical (eye-shadow) )))
```