Racket Programming Assignment #2: Racket Functions and Recursion

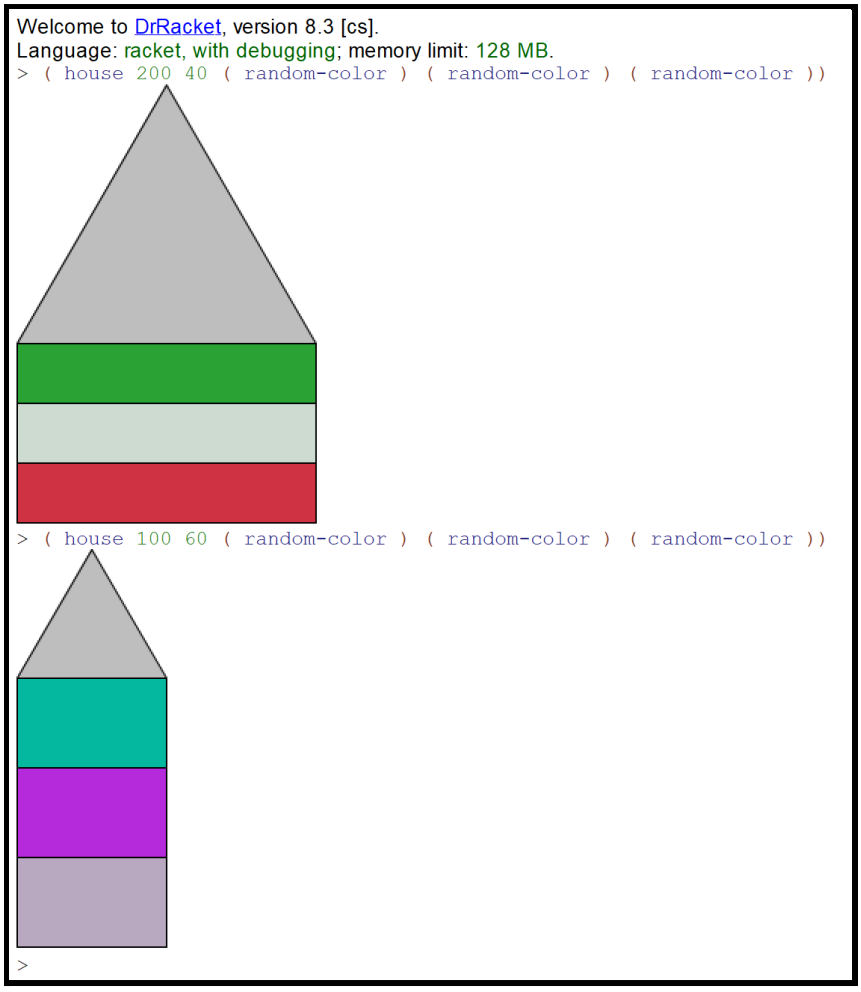
Learning Abstract

This programming assignment focuses on the 2htdp/image library as well as recursive programming in Racket. Every task except the first is completed using recursive methods. This solution is a work in progress and I will continue to update once I finish the final task.

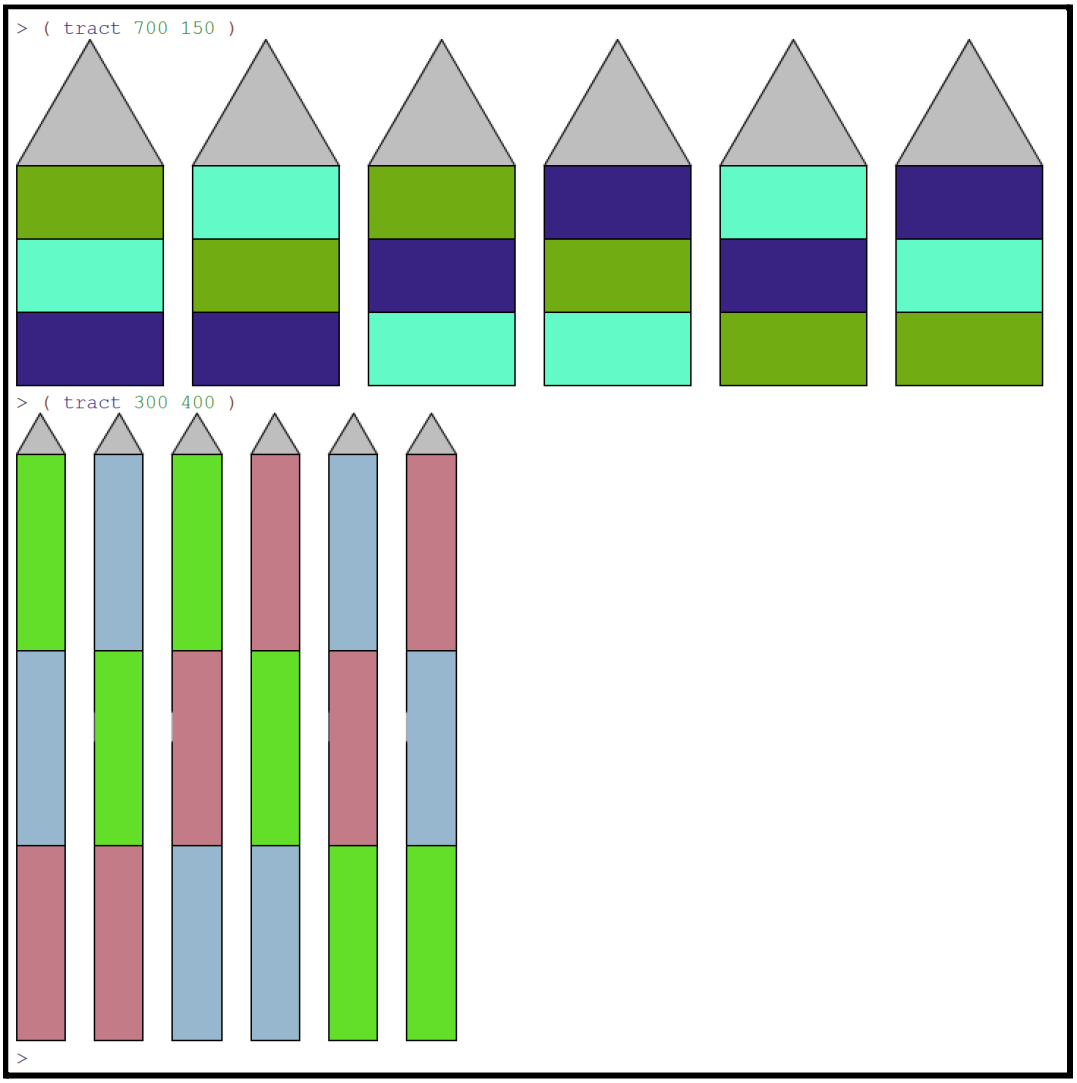
Task 1: Colorful Permutations of Tract Houses

This code creates single houses and sets of 6 houses all in a row each with three floors and a roof. The house function takes two integer parameters for the width and height of the floor, and three color parameters for the floors in ascending order. Tract only takes two arguments: the width of the tract and height of the combined floors.

House Demo:



Tract Demo:



House and Track Code:

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

;function for creating a random color from three random rbg values
( define ( random-color ) ( color (rgb-val) (rgb-val) (rgb-val))
 )

;function for getting random number 0-255
( define ( rgb-val )
   ( random 256 )
 )
```

```
;function for building and printing one house
( define ( house floor-width floor-height color1 color2 color3 )

   ;define the floors
   ( define first-floor
      (overlay
         ( rectangle floor-width floor-height "outline" "black" )
         ( rectangle floor-width floor-height "solid" color1 )
       )
    )
   ( define second-floor
      (overlay
        ( rectangle floor-width floor-height "outline" "black" )
        ( rectangle floor-width floor-height "solid" color2 )
       )
    )
   ( define third-floor
      (overlay
        ( rectangle floor-width floor-height "outline" "black" )
        ( rectangle floor-width floor-height "solid" color3 )
       )
    )

   ;define the roof
   ( define roof
      (overlay
        ( triangle floor-width "outline" "black")
        ( triangle floor-width "solid" "gray" )
       )
     )

   ;stack all of it
   ( define the-house ( above roof third-floor second-floor first-floor))

   ;display the-house
   the-house
 )

; - function to create a row of 6 houses containing all permutations of
;       the three random colors with a space of 10 pixels
; - this function accepts two parameters the floor-width and floor-height
( define (tract track-width total-floor-height)

   ;create our three colors to work with
   ( define color1 (random-color))
   ( define color2 (random-color))
   ( define color3 (random-color))

   ;math to feed correct values to (house)
```

```
   ( define floor-width ( / ( - track-width 100) 6))
   ( define floor-height (/ total-floor-height 3.0))

   ;create a spacer object to create blocks
   ( define (spacer)
      ( square 20 "solid" "white" )
    )

   ;create our train of houses
   ( define the-tract
      (beside
         (house floor-width floor-height color1 color2 color3)
         (spacer)
         (house floor-width floor-height color1 color3 color2)
         (spacer)
         (house floor-width floor-height color2 color1 color3)
         (spacer)
         (house floor-width floor-height color2 color3 color1)
         (spacer)
         (house floor-width floor-height color3 color1 color2)
         (spacer)
         (house floor-width floor-height color3 color2 color1)
       )
    )
   ;display the-tract
   the-tract
)
```

Task 2: Dice

The Dice code creates and rolls assorted imaginary dice and contains functions to roll for a 1 or two 1's in a row, and similarly can stop after a pattern of odds and evens as well as finding pairs similar to a game of craps

Dice Demo:

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( roll-die )
3
> ( roll-die )
2
> ( roll-die )
6
> ( roll-die )
5
> ( roll-die )
2
> ( roll-for-1 )
2 3 1
> ( roll-for-1 )
6 1
> ( roll-for-1 )
3 1
> ( roll-for-1 )
4 3 5 5 6 3 3 4 3 5 1
> ( roll-for-1 )
5 3 1
```

```
> ( roll-for-11 )
4 1 3 5 4 5 4 5 2 3 2 3 4 3 3 6 3 5 1 4 2 6 5 3 2 3 6 2 3 6 2 6 2 6 3 1 5 6 1 5 ↵
2 3 2 5 6 6 5 4 4 2 6 2 6 1 4 5 5 4 5 4 4 2 4 4 5 5 2 5 4 1 1
> ( roll-for-11 )
1 3 6 5 4 5 1 1
> ( roll-for-11 )
2 5 2 1 2 1 6 2 4 2 5 1 1
> ( roll-for-11 )
1 4 4 6 1 5 6 1 5 2 4 2 5 2 2 2 6 3 5 3 5 1 4 2 6 5 3 4 4 5 1 1
> ( roll-for-11 )
6 5 3 2 3 3 5 2 4 5 3 1 3 3 2 6 6 4 4 5 4 6 1 5 5 6 4 3 5 6 2 6 3 2 6 4 5 4 3 4 ↵
4 6 2 5 6 1 3 4 4 3 4 6 5 4 3 3 6 4 4 5 4 3 1 4 6 1 5 2 1 3 2 5 2 5 5 5 6 2 6 6 ↵
3 2 4 2 3 4 1 2 6 2 5 1 6 2 5 5 3 4 3 5 4 2 3 3 6 6 6 2 3 2 2 6 5 4 4 3 5 5 6 5 ↵
2 5 2 4 1 2 3 6 3 3 2 2 3 1 2 2 1 3 3 5 3 3 2 6 4 1 2 3 2 2 6 4 1 5 1 2 3 4 4 6 ↵
6 4 5 4 6 5 5 6 5 3 6 4 1 2 4 2 1 1
> ( roll-for-odd-even-odd )
5 5 1 4 2 4 2 2 3 5 6 2 1 2 2 1 2 2 3 1 4 1
> ( roll-for-odd-even-odd )
4 2 6 3 1 1 1 1 3 4 4 2 4 1 2 2 6 4 3 1 5 5 4 5
> ( roll-for-odd-even-odd )
5 1 5 4 1
> ( roll-for-odd-even-odd )
6 1 3 4 3
> ( roll-for-odd-even-odd )
5 1 3 2 3
```

```
> ( roll-for-odd-even-odd )
5 5 1 4 2 4 2 2 3 5 6 2 1 2 2 1 2 2 3 1 4 1
> ( roll-for-odd-even-odd )
4 2 6 3 1 1 1 1 3 4 4 2 4 1 2 2 6 4 3 1 5 5 4 5
> ( roll-for-odd-even-odd )
5 1 5 4 1
> ( roll-for-odd-even-odd )
6 1 3 4 3
> ( roll-for-odd-even-odd )
5 1 3 2 3
> ( roll-two-dice-for-a-lucky-pair )
(3 4) (4 6) (6 2) (6 4) (3 1) (2 5) (2 6) (2 4) (1 6) (6 1) (6 5)
> ( roll-two-dice-for-a-lucky-pair )
(5 2) (6 1) (5 5)
> ( roll-two-dice-for-a-lucky-pair )
(1 5) (1 6) (2 4) (2 3) (2 3) (6 3) (4 6) (4 3) (5 1) (2 5) (5 5)
> ( roll-two-dice-for-a-lucky-pair )
(6 3) (5 4) (6 5)
> ( roll-two-dice-for-a-lucky-pair )
(5 4) (6 1) (4 4)
> ( roll-two-dice-for-a-lucky-pair )
(2 6) (2 2)
> ( roll-two-dice-for-a-lucky-pair )
(1 3) (4 1) (4 1) (5 5)
> ( roll-two-dice-for-a-lucky-pair )
(5 4) (6 4) (5 3) (1 6) (2 6) (3 1) (4 5) (4 3) (3 4) (1 5) (3 6) (5 5)
> ( roll-two-dice-for-a-lucky-pair )
(6 4) (5 4) (6 3) (1 6) (4 5) (1 6) (6 3) (2 2)
> ( roll-two-dice-for-a-lucky-pair )
(5 1) (2 2)
>
```

Dice Code:

```
—------------------------
#lang racket

;roll-die function rolls an imaginary die and returns
;  a value 1-6
( define ( roll-die )
   ( + ( random 6 ) 1 )
  )

;roll-for-1 function rolls a die until a 1 is rolled
;  all rolls are printed
( define ( roll-for-1 )
   ( define current-roll ( roll-die ))
   ( display current-roll)( display " ")
   ( cond ((not(= current-roll 1 ))
       (roll-for-1)
       )
    )
 )

;roll-for-11 functions rolls a die for two consecutive
;  1's and prints outcomes along the way
( define ( roll-for-11 )
   ( roll-for-1 )
   ( define current-roll ( roll-die ))
   ( display current-roll)( display " ")
   ( cond ((not ( = current-roll 1 ))
             ( roll-for-11)
           )
     )
 )

;roll-for-odd-even-odd is gonna take a few more functions to build
;  this block will be used to hold these functions
;

;three-count-roller
( define ( three-count-roller prev1-roll prev2-roll )
   ( define current-roll ( roll-die ))
   ( display current-roll )( display " " )
   (cond ((not (and (odd? current-roll) (even? prev2-roll ) (odd? prev1-roll)))
          (three-count-roller prev2-roll current-roll)
         )
     )
)

;roll-for-odd-even will roll until an odd then even number is rolled
( define ( roll-for-odd-even-odd )
```

```
   ( define first-roll ( roll-die ))
   ( display first-roll ) (display " ")

   ( define second-roll ( roll-die ))
   ( display second-roll ) (display " ")

   ( three-count-roller first-roll second-roll)
 )

;roll-two-dice-for-a-lucky-pair rolls two dice and stops once their
;    sum is 7, 11 or the dice are equal
( define ( roll-two-dice-for-a-lucky-pair )
   ( define roll-one ( roll-die ) )
   ( define roll-two  ( roll-die) )

   ( display "(") ( display roll-one ) ( display " ")
    ( display roll-two) ( display ") ")

   ( define roll-sum ( + roll-one roll-two) )

   (cond ((not (or ( = roll-sum 11) ( = roll-sum 11 ) ( = roll-one roll-two)))
          (roll-two-dice-for-a-lucky-pair )
         )
     )
 )
```

Task 3: Number Sequences

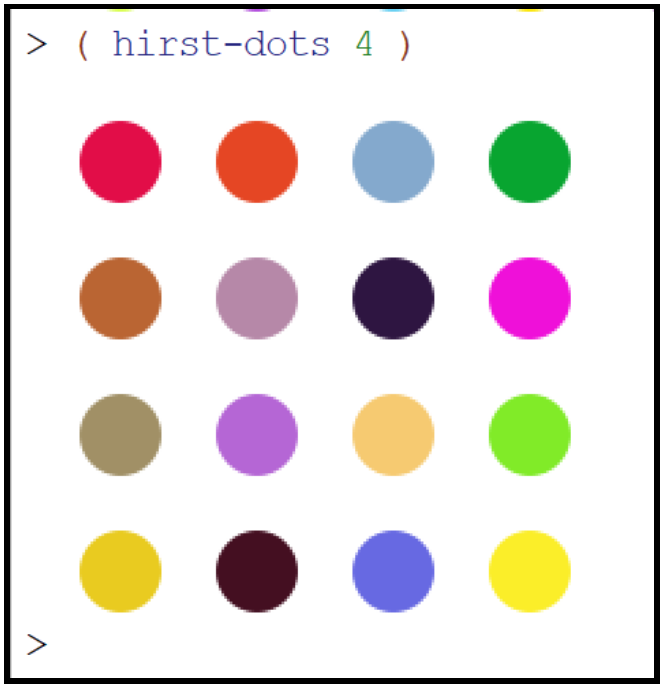
Preliminary Demo:

```
Welcome to DrRacket, version 8.3 [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
>
```

Task 4: Hirst Dots

This set of programs creates a square grid of randomly colored circles with radius 15 and a spacing of 20 pixels above and below each dot. These programs are written using recursion.

Hirst Dots Demo:



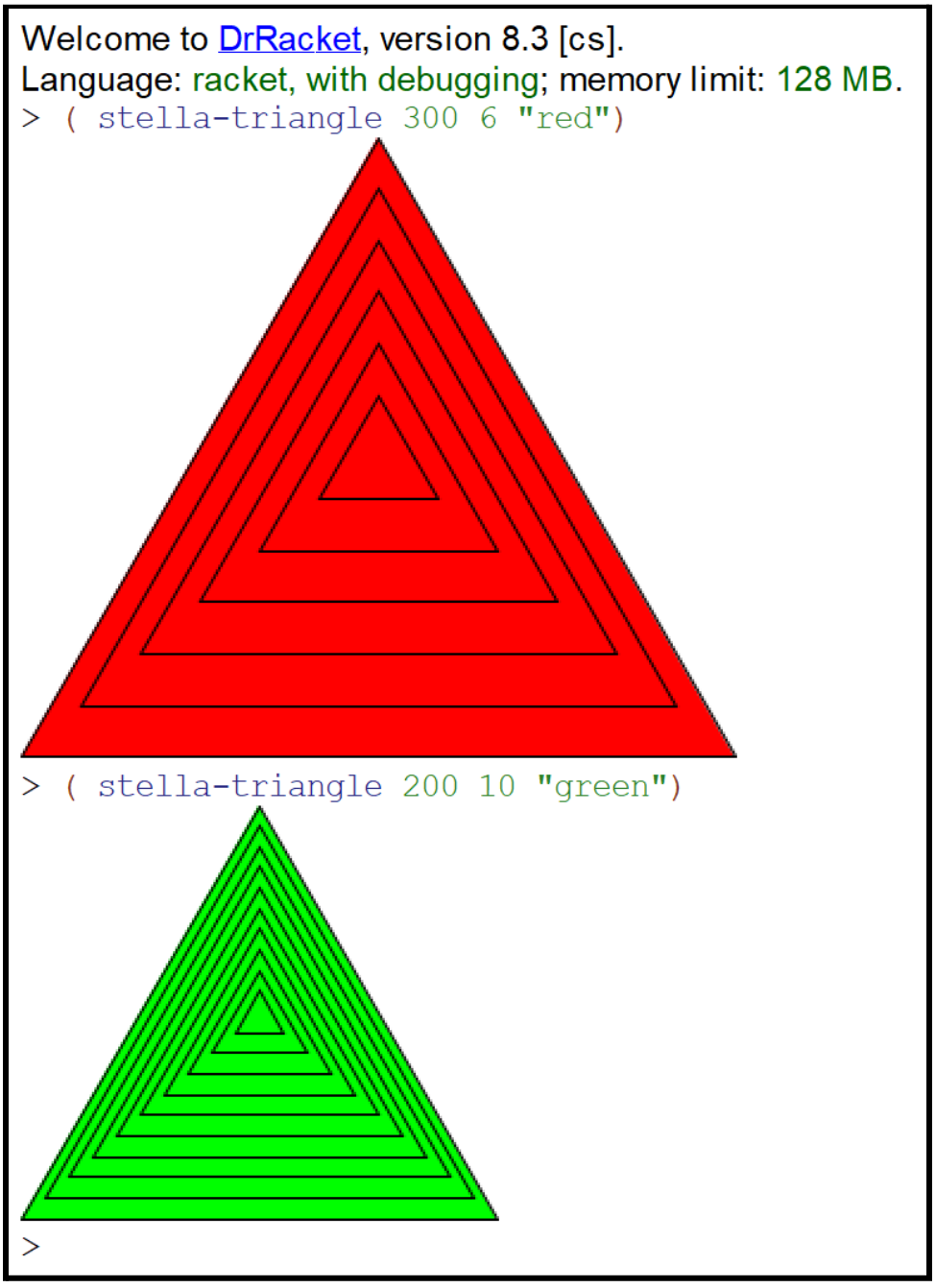

```
#lang racket
(require 2htdp/image)

;function for creating a random color from three random rbg values
( define ( random-color ) ( color (rgb-val) (rgb-val) (rgb-val))
 )

;function for getting random number 0-255
( define ( rgb-val )
   ( random 256 )
 )

; function to create on randomly colored circle
( define ( random-colored-circle )
   ( circle 15 "solid" ( random-color ) )
   )

; function to create spacer item
( define ( spacer )
   ( square 20 "solid" "white" )
   )

;function to create row of n circles
( define ( row-of-circles n random-colored-circle )
   ( cond (( = n 0 )
           empty-image
       )  (( > n 0 )
           ( beside ( row-of-circles ( - n 1 ) random-colored-circle ) (spacer)
```

```
                    ( random-colored-circle ) )
           )
       )
   )

;function to create rectangle of r rows and c columns
( define ( rectangle-of-circles r c random-colored-circle )
   ( cond (( = r 0 )
            empty-image
       )  (( > r 0 )
            ( above ( rectangle-of-circles ( - r 1 ) c  random-colored-circle )
                              (spacer)( row-of-circles c random-colored-circle )
)
           )
      )
   )

;function to create a square of m by m size
( define ( hirst-dots n )
   ( rectangle-of-circles n n random-colored-circle )
   )
```

Task 5: Channeling Frank Stella

Program that creates monochromatic stella triangles. Takes parameters `max-side`, `count`, and a color. Creates the shapes using recursion.

Channeling Frank Stella Demo:



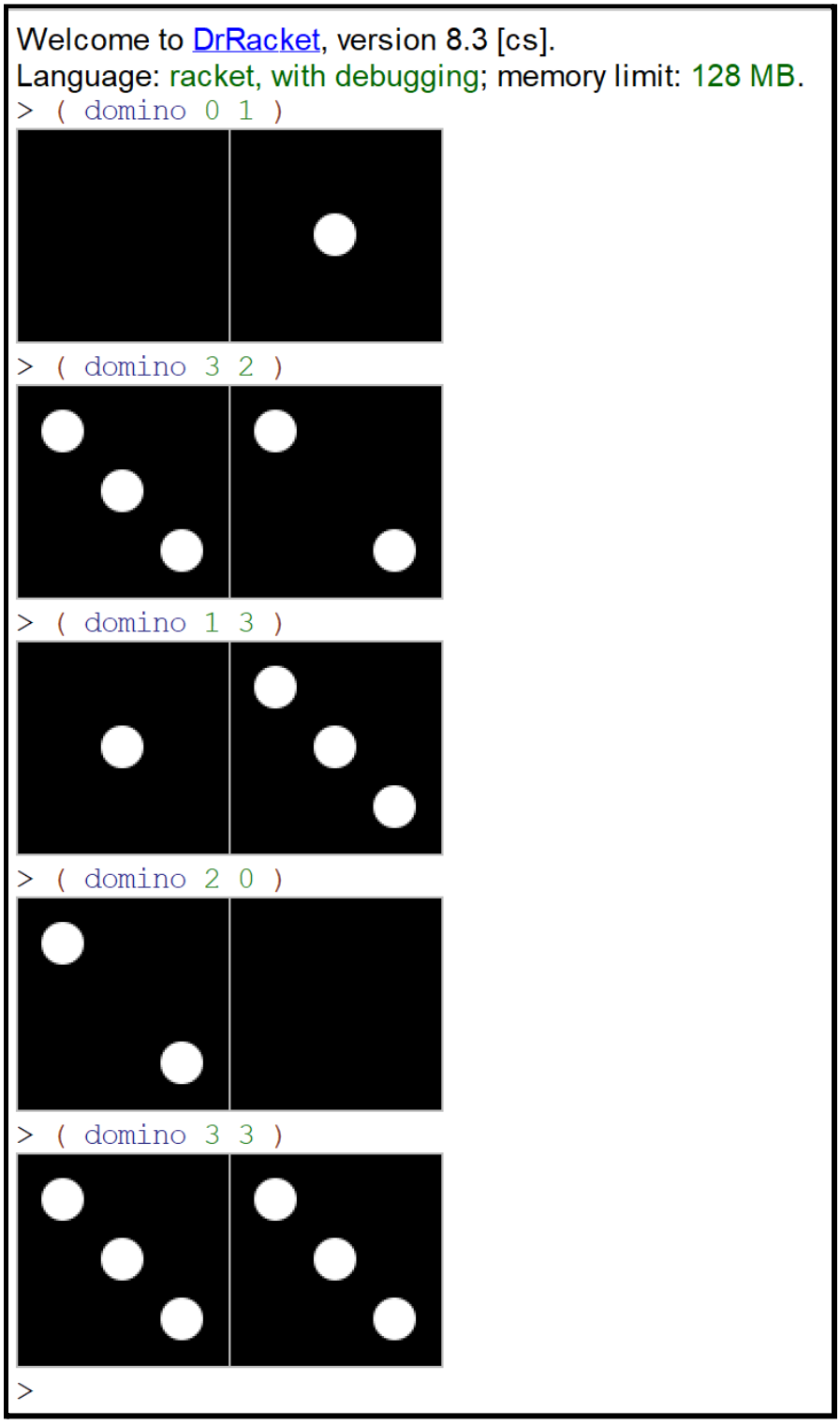
Channeling Frank Stella Demo:

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

;recursively calls to paint a triangle and then decrease side
;length by appropriate amount to then calls itself again
( define ( paint-triangle counter total unit color)
   ( define side-length ( * counter unit ) )
   ( cond (( = counter total )
           ( framed-triangle side-length color )
           )
          (( < counter total )
           ( overlay
             ( framed-triangle side-length color )
             ( paint-triangle ( + counter 1 ) total unit color )
            )
          )
    )
 )

;creates one triangle with side length side-length
( define ( framed-triangle side-length color )
   ( overlay
     ( triangle side-length "outline" "black" )
     ( triangle side-length "solid" color )
    )
  )

;used to create the stella shape kicked off recursion
( define ( stella-triangle max-side count color )
   ( define unit-side ( / max-side count ))
   ( paint-triangle 1 count unit-side color )
)
```

Task 6: Dominos

Program to create and print dominos that builds off of given code to finish all needed functions.

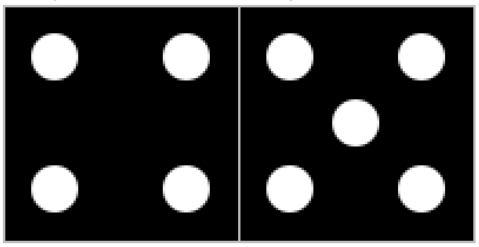
Demo for 0, 1, 2, and 3 pip dominos:



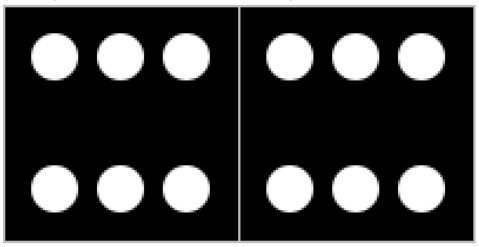
Final domino rendering demo:

Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.

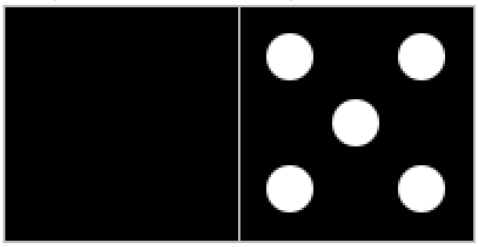
```
> ( domino 4 5 )
```



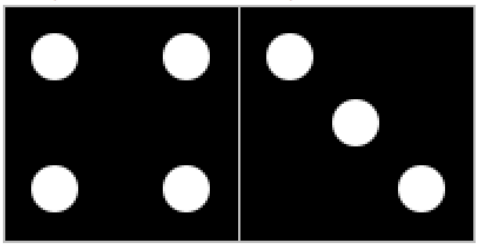
```
> ( domino 6 6 )
```



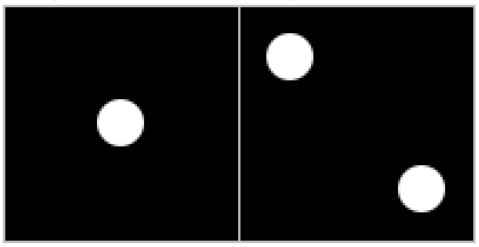
```
> ( domino 0 5 )
```



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



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



```
>
```

Domino 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 nd (overlay/offset ( pip ) nd d
basic-tile2 ) ) )

( define basic-tile5 ( overlay ( pip ) basic-tile4 ) )

( define basic-tile6 ( overlay/offset ( pip ) 0 nd ( overlay/offset ( pip ) 0 d
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 )s
      ( ( = x 1 ) tile1 )
      ( ( = x 2 ) tile2 )
      ( ( = x 3 ) tile3 )
      ( ( = x 4 ) tile4 )
      ( ( = x 5 ) tile5 )
      ( ( = x 6 ) tile6 )
    )
  )
```

Task 7: Creation

I had several ideas, but none were working as well as I was hoping. My end goal is to recursively create a clock face utilizing the dominos created in Task 6. I will update this when I make it happen.