Matilda Knapp **2/24/2022**
CSC344 **Racket**

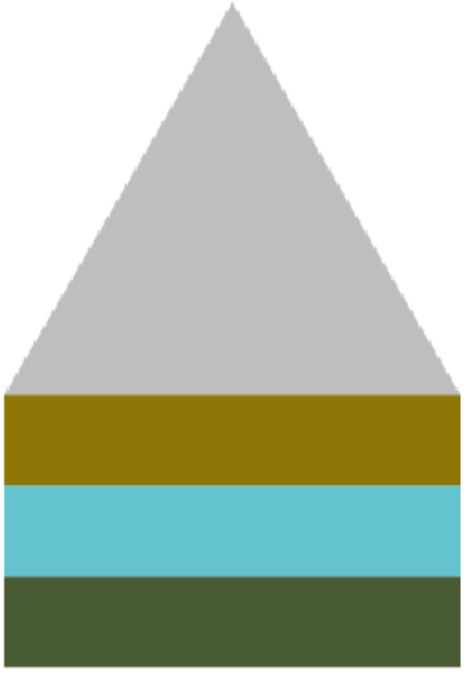
Racket Programming Assignment #2: Racket Functions and Recursion

Learning Abstract

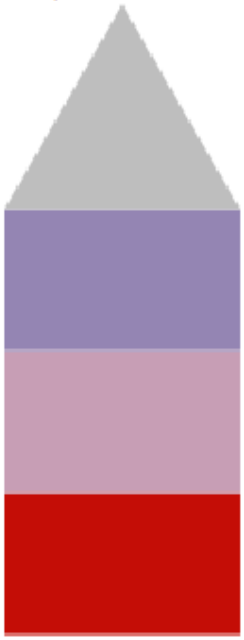
This assignment features the code and demos for several programs in Racket. The first program is one which creates a house tract of six houses with a variety of three colors. This program also has a function to produce a single home. The second program is a program that rolls a 6 sided die. This program also has a few functions for certain roll expectations, such as to keep rolling till a one is outputted. After this the third program works with functions which compute the square, cube, triangular, and sigma of a number. The Triangular and Sigma codes were the ones I was responsible for; there is also a sequence function. Fourth we have a program that produces hirst dots, and fifth we have a program that produces a two colored stella of a circle. These are followed by the sixth program which produces dominos. I worked on creating the recursive functions for the dominos with 4, 5, and 6 pips. Lastly, the seventh program is a program of my own creation. This program creates multiple pyramids which turn on a 45 degree angle each time a new one is created. Each tile in each pyramid has a somewhat randomized color value, and the height of the pyramids can be set when the function is called.

The House Demo (Task One)

```
> ( house 200 40 (random-color) (random-color) (random-color) )
```

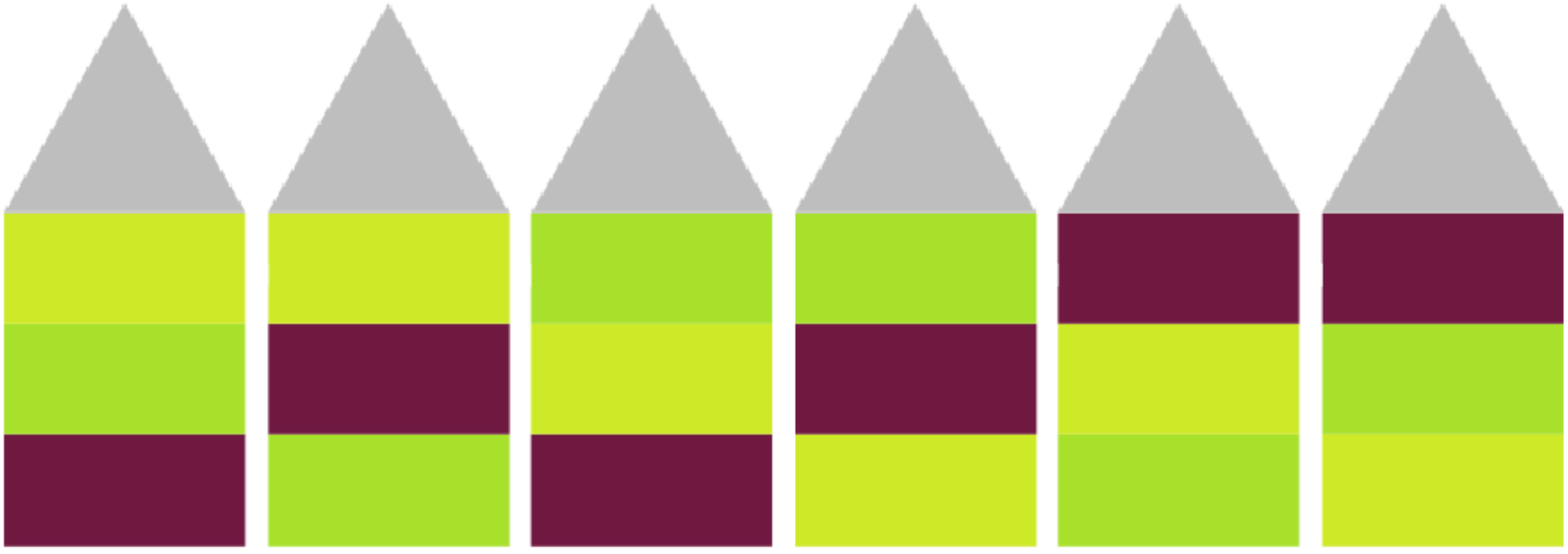


```
> ( house 100 60 (random-color) (random-color) (random-color) )
```

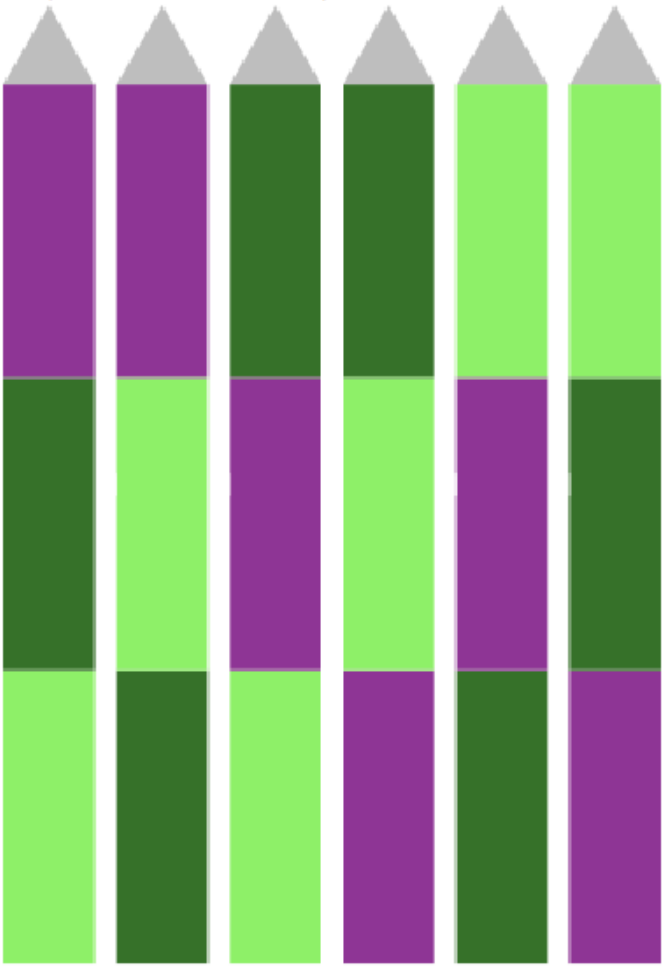


The Tract Demo (Task One)

```
> ( tract 700 150 )
```



```
> ( tract 300 400 )
```



Colorful Permutations of Tract Houses Code (Task One)

```
#lang racket
;Adding shape function
(require 2htdp/image )
;Creating Random Color Function
( define ( rgb ) ( random 256 )
   )
( define ( random-color ) ( color ( rgb ) ( rgb ) ( rgb ) )
   )
;Making Block Function
( define (floor hori verti color-ran) (rectangle hori verti "solid" color-ran )
   )
```

```
;Making Triangle Function
( define (roof hori) (triangle hori "solid" "grey" )
   )
;Making White Space
( define (space) (square 10 "solid" "white" )
   )
;Making House One Function
( define (house width height rainbowL rainbowLL rainbowLLL)
   ( define width-side(* width 1))
   ( define height-side(* height 1))
   ( define bottom (floor width-side height-side rainbowL))
   ( define middle (floor width-side height-side rainbowLL))
   ( define top (floor width-side height-side rainbowLLL))
   ( define the-roof (roof width-side))
   ( define the-house (above the-roof top middle bottom))
   the-house
   )
;Making House Tract
( define (tract road sky)
   (define random-a (random-color))
   (define random-b (random-color))
   (define random-c (random-color))
   ( define measure (- road 50))
   ( define the-sky (/ sky 3))
   ( define the-sidewalk (/ measure 6))
   ( define the-space (space))
   ( define house-o (house the-sidewalk the-sky random-a random-b random-c))
   ( define house-t (house the-sidewalk the-sky random-b random-a random-c))
   ( define house-th (house the-sidewalk the-sky random-a random-c random-b))
   ( define house-f (house the-sidewalk the-sky random-c random-a random-b))
   ( define house-fi (house the-sidewalk the-sky random-b random-c random-a))
   ( define house-s (house the-sidewalk the-sky random-c random-b random-a))
   ( define house-tract (beside house-o the-space house-t the-space house-th
                               the-space house-f the-space
                               house-fi the-space house-s))
   house-tract
   )
```

The Dice Demo (Task Two)

```
> ( roll-die )
2
> ( roll-die )
2
> ( roll-die )
6
> ( roll-die )
2
> ( roll-die )
3
>  ( roll-for-1)
6 5 2 5 3 1
>  ( roll-for-1)
5 4 1
>  ( roll-for-1)
5 4 2 5 3 4 5 3 6 3 2 3 6 1
>  ( roll-for-1)
2 6 5 2 2 1
>  ( roll-for-1)
4 4 6 3 5 2 3 6 5 5 4 4 5 4 5 5 2 6 4 2 3 5 1
>  ( roll-for-11 )
2 6 1 3 3 6 2 1 6 1 2 1 3 6 5 1 1
>  ( roll-for-11 )
4 3 6 1 4 3 5 1 1
>  ( roll-for-11 )
6 4 2 3 6 4 5 6 5 5 4 1 2 4 5 4 4 3 3 2 3 1 3 1 4 4 1 6 6 3 2 4
5 4 1 2 5 2 4 5 3 6 2 6 1 3 4 3 1 3 1 4 3 4 4 4 4 3 5 3 6 4 2 2
3 1 1
>  ( roll-for-11 )
4 1 6 4 2 2 1 2 4 4 2 3 5 2 2 5 6 2 5 5 1 5 6 5 5 3 6 6 5 2 5 4
6 2 1 1
>  ( roll-for-11 )
6 2 1 1
```

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

The Dice Code (Task Two)

```
#lang racket
( define (roll-die)
```

```
   (define outcome (random 6 ) )
   (cond
     ( ( = outcome 0 ) 1 )
     ( ( = outcome 1 ) 2 )
     ( ( = outcome 2 ) 3 )
     ( ( = outcome 3 ) 4 )
     ( ( = outcome 4 ) 5 )
     ( ( = outcome 5 ) 6 )
     )
   )
( 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 )
        )
      )
   )
( 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
      ( ( not ( even? outcome ) )
        ( roll-for-odd )
        )
      )
   )
( define ( roll-for-odd-even )
   ( roll-for-odd )
   ( define outcome ( roll-die ) )
   ( display outcome ) ( display " " )
   ( cond
      ( ( not ( even? 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 )
        )
      )
   )
( define ( roll-two-dice-for-a-lucky-pair )
   ( define outcome-1 ( roll-die ) )
   ( define outcome-2 ( roll-die ) )
   ( define add-outcome ( + outcome-1 outcome-2 ) )
   ( display "(" ) ( display outcome-1 ) ( display " " ) ( display outcome-2 ) (
display ")" )
   ( cond
      ( (= outcome-1 outcome-2 ) )
      ( (= add-outcome 7 ) )
      ( (= add-outcome 11 ) )
      (else ( roll-two-dice-for-a-lucky-pair ) ) )
   ( display " " )
   )
```

Preliminary Demo (Task Three)

```
> ( 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 Demo (Task Three)

```
> ( 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 Demo (Task Three)

```
>  ( 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
```

Preliminary, Triangular, Sigma, Code Mix (Task Three)

```
#lang racket
;This is the 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 " " )
        )
      )
   )
;This is the Triangular Code
( define ( triangular num )
   ( cond
      ( (= num 1 )
        1
        )
```

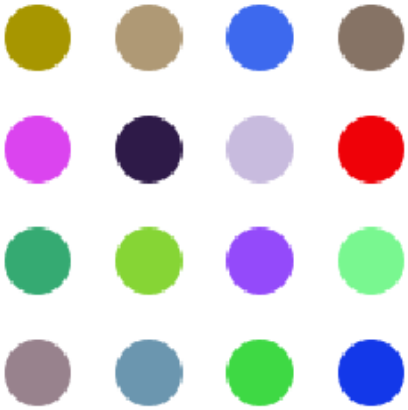
```
      ( else
        (+ num (triangular (- num 1 ) ) )
        )
      )
   )

;This is a method I used to get the Sigma Code
( define ( doublesigma num sub )
   ( cond
      ( (= num 1)
        1
        )
      ( (= (modulo sub num) 0)
        (+ num ( doublesigma (- num 1) sub ))
        )
      ( else
        (+ 0 ( doublesigma (- num 1) sub ))
        )
      )
   )
;This is the Sigma Code
( define ( sigma n )
   ( doublesigma n n )
   )
```

Hirst Dots Demo (Task Four)

```
> ( hirst-dots 10 )
```

```
> ( hirst-dots 4 )
```



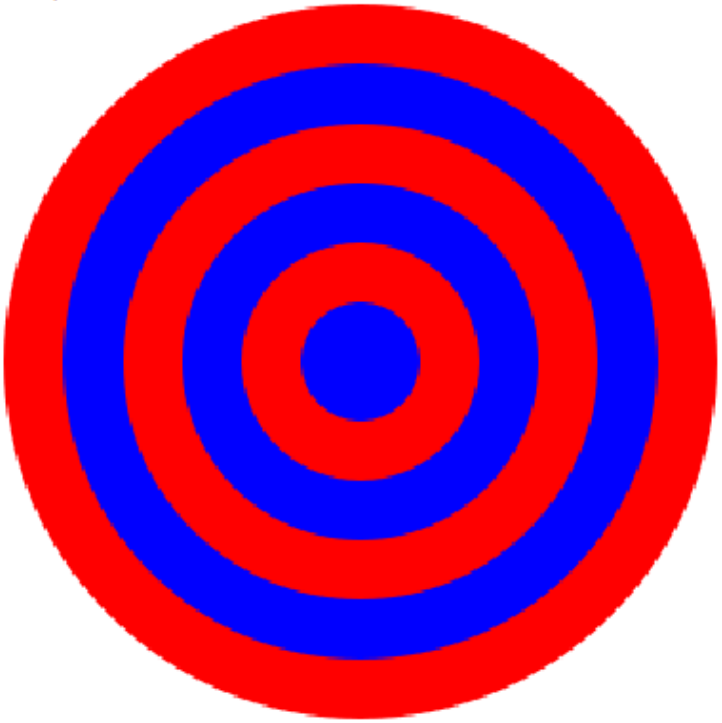
Hirst Dots Code (Task Four)

```
#lang racket
; -----------------------------------------------------------------------
; This Program Creates Hirst Dots.
; Each Dot Is A Random Color.
; -----------------------------------------------------------------------
; (Image Library)
(require 2htdp/image)
; Generate a row of Hirst Dots.
( define ( row-of-dots n)
   ( define ( tile )
      ( overlay
        ( square 50 "solid" ( white-color ) )
        ( circle 15 "solid" ( random-color ) )
        )
      )
   ( cond
      ((= n 0)
       empty-image
       )
      ((> n 0)
       ( beside ( row-of-dots ( - n 1 )) ( tile ) )
       )
      )
   )
; -----------------------------------------------------------------------
; Generate a rectangle of Hirst Dots.
( define ( rectangle-of-dots r c )
   ( cond
      ((= r 0)
       empty-image
       )
      ((> r 0)
       ( above
         ( rectangle-of-dots ( - r 1 ) c) ( row-of-dots c ) )
       )
      )
```

```
   )
; ---------------------------------------------------------------------
; Generate a square of Hirst Dots.
( define ( hirst-dots n )
   ( rectangle-of-dots n n )
   )
; ---------------------------------------------------------------------
; Random Color Function.
( define ( random-color )
   ( color
     ( rgb ) ( rgb ) ( rgb )
     )
   )
( define ( white-color )
   ( color 0 0 0 0 )
   )
( define ( rgb ) ( random 256 ) )
```

Stella Circle Demo (Task Five)

```
> ( nested-circles 300 6 "red" "blue" )
```



```
> ( nested-circles 200 20 "purple" "white" )
```

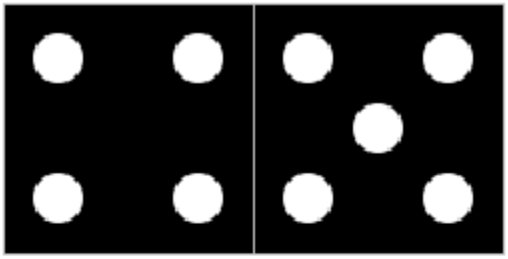
Stella Circle Code (Task Five)

```
#lang racket
( require 2htdp/image )
( define ( nested-circles side count color1 color2 )
   ( define delta ( / (/ side 2) count ) )
   ( paint-nested-circles 1 count delta color1 color2 )
   )
( define ( paint-nested-circles from to delta color1 color2 )
   ( define side-length ( * from delta ) )
   ( cond
      ( ( = from to )
        ( if ( even? from )
             ( circle side-length "solid" color1 )
             ( circle side-length "solid" color2 )
             )
        )
      ( ( < from to )
        ( if ( even? from )
             ( overlay
               ( circle side-length "solid" color1 )
               ( paint-nested-circles ( + from 1 ) to delta color1 color2 )
               )
             ( overlay
```

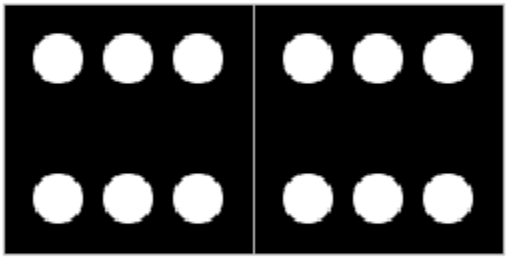
```
                ( circle side-length "solid" color2 )
                ( paint-nested-circles ( + from 1 ) to delta color1 color2 )
                )
              )
          )
        )
    )
( define ( random-color ) ( color ( rgb ) ( rgb ) ( rgb ) ) )
( define ( rgb ) ( random 256 ) )
```

Final Domino Demo (Task Six)

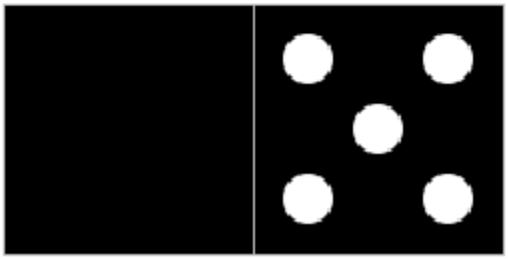
```
> ( domino 4 5 )
```



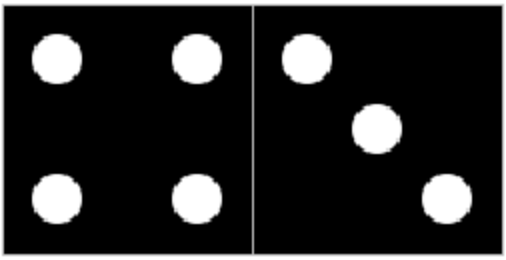
```
> ( domino 6 6 )
```



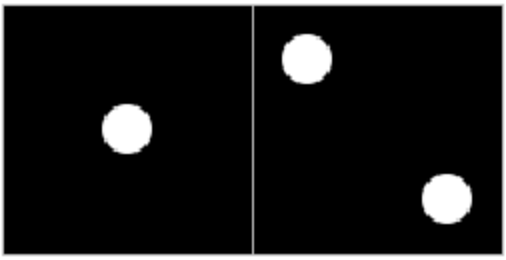
```
> ( domino 0 5 )
```



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



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



Finished Domino Code (Task Six)

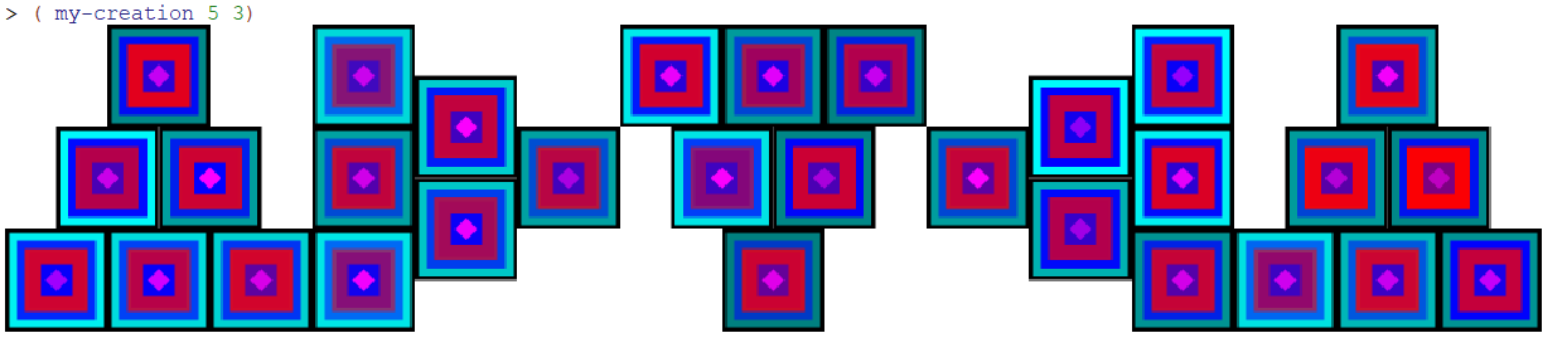
```
#lang racket
;-------------------------------------------------------------------------------
; Racket Image Machine
;
( require 2htdp/image )
;-------------------------------------------------------------------------------
; Dimensions
;
( define side-of-tile 100 )
( define diameter-of-pip ( * side-of-tile 0.2 ) )
( define radius-of-pip ( / diameter-of-pip 2 ) )
;-------------------------------------------------------------------------------
; Numbers for offsetting pips
;
( define d ( * diameter-of-pip 1.4 ) )
( define nd ( * -1 d ) )
;-------------------------------------------------------------------------------
; Tile and the pip generator
;
( define blank-tile ( square side-of-tile "solid" "black" ) )
( define ( pip ) ( circle radius-of-pip "solid" "white" ) )
```

```
;-------------------------------------------------------------------------
; Dominos
;
( 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 )
                    )
   )
;-------------------------------------------------------------------------
; Adds Outline
;
( 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 ) )
;-------------------------------------------------------------------------
; Function to generate a domino
;
( define ( domino a b )
   ( beside ( tile a ) ( tile b ) )
   )
( define ( tile x )
```

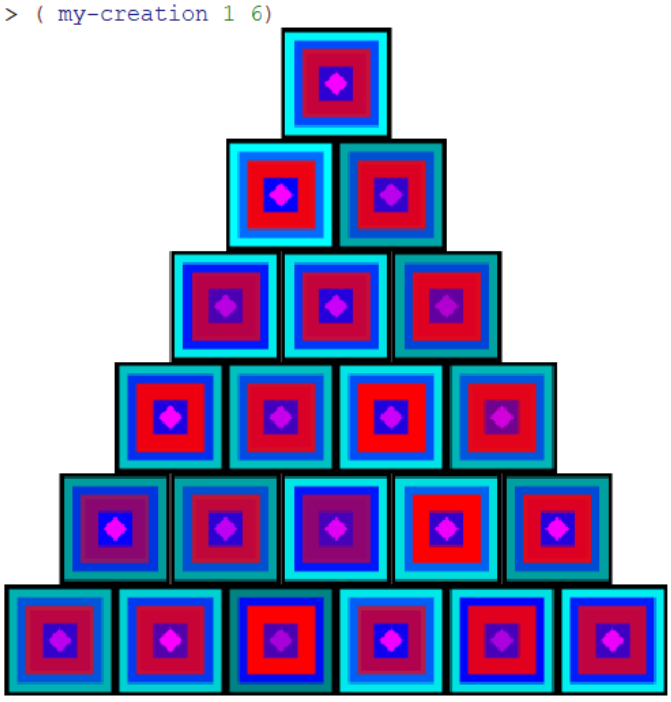
```
( cond
   ( ( = x 0 ) tile0 )
   ( ( = x 1 ) tile1 )
   ( ( = x 2 ) tile2 )
   ( ( = x 3 ) tile3 )
   ( ( = x 4 ) tile4 )
   ( ( = x 5 ) tile5 )
   ( ( = x 6 ) tile6 )
   )
)
```

My Creation Demo (Task Seven)

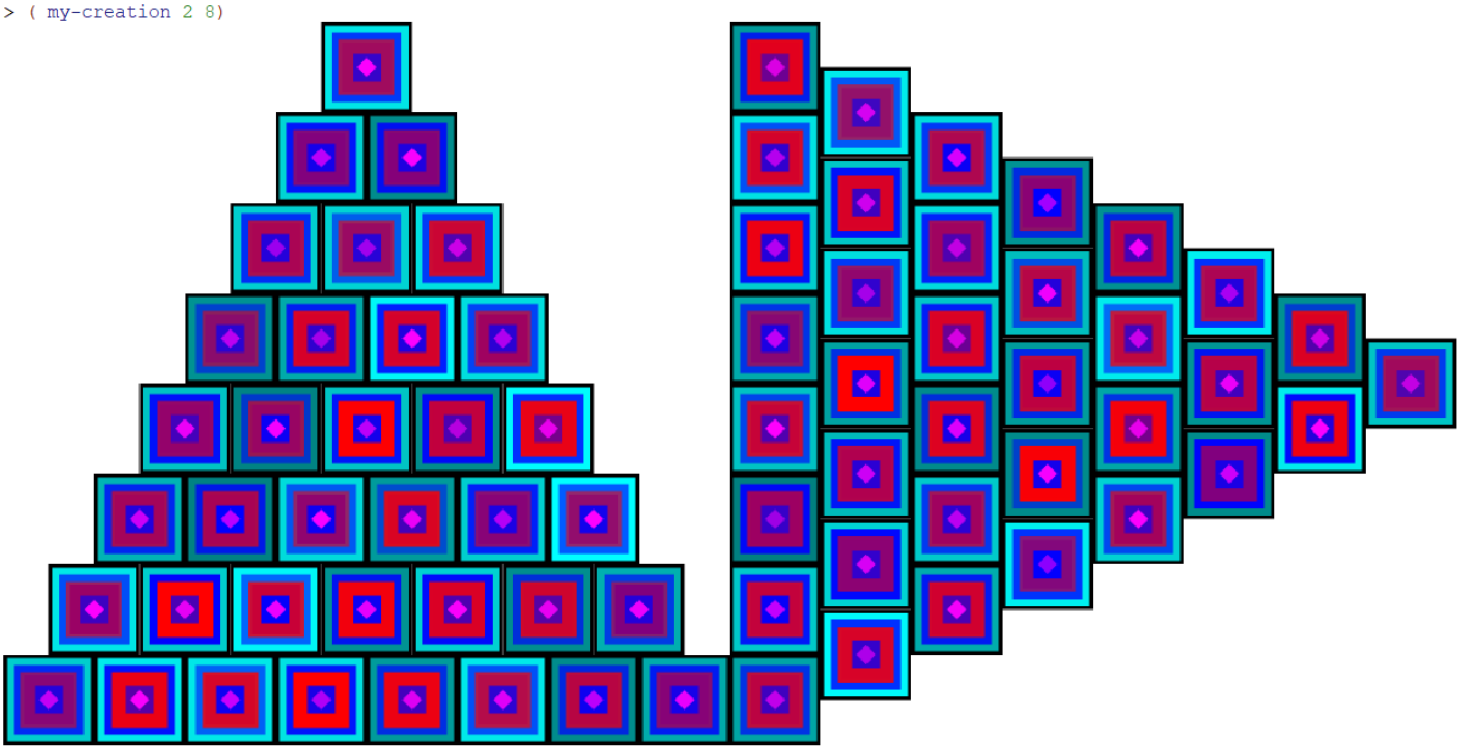
```
> ( my-creation 5 3)
```



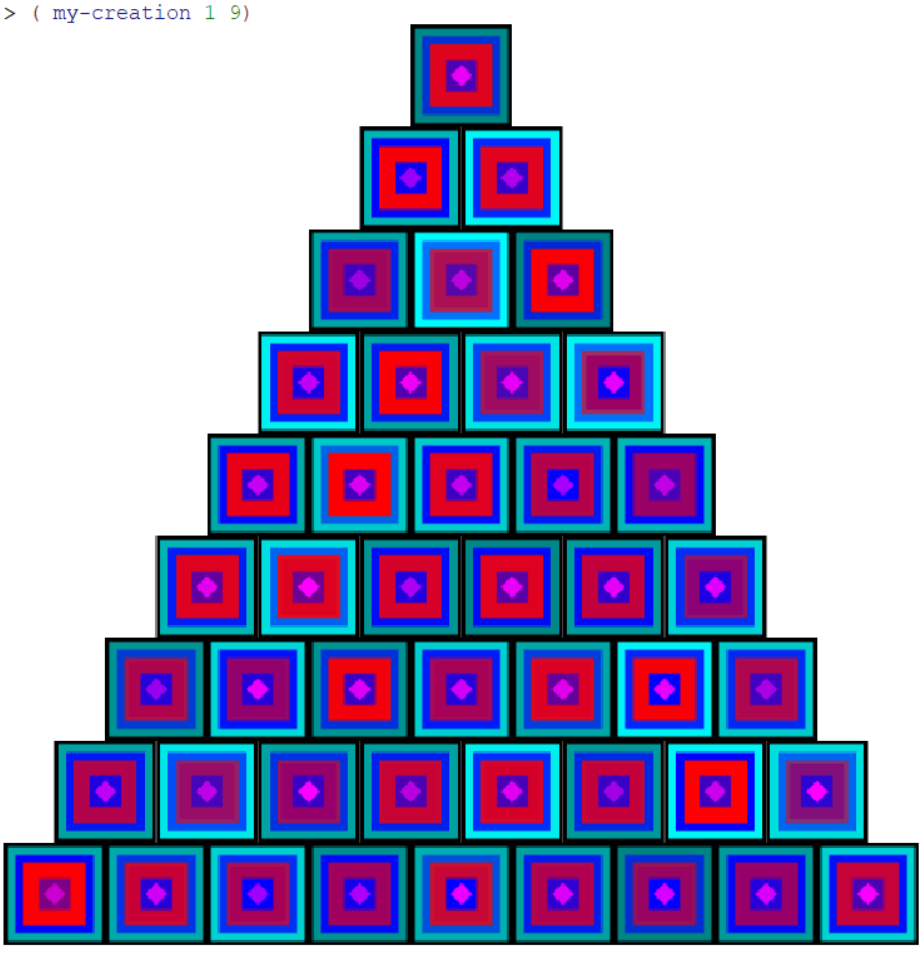
```
> ( my-creation 1 6)
```



```
> ( my-creation 2 8)
```



```
> ( my-creation 1 9)
```



My Creation Code (Task Seven)

```
#lang racket
(require 2htdp/image)
; Generate a row of colorful bricks
( define ( row-of-tiles n)
   ( define ( tile )
      ( overlay
        (rotate 45( square 10 "solid" ( random-magenta-color ) ))
        ( square 20 "solid" ( random-blue-color ) )
        ( square 40 "solid" ( random-red-color ) )
        ( square 35 "solid" ( random-blue-color ) )
        ( square 50 "solid" ( random-blue-color ) )
        ( square 60 "solid" ( random-cyan-color ) )
        ( square 65 "solid" "black" )
        )
      )
   ( cond
      ((= n 0)
       empty-image
       )
      ((> n 0)
       ( beside ( row-of-tiles ( - n 1 )) ( tile ) )
       )
      )
   )
; ---------------------------------------------------------------------
; pyramid
( define ( triangle r )
   ( cond
      ((= r 0)
       empty-image
       )
      ((> r 0)
       ( above
         ( triangle ( - r 1 ) ) ( row-of-tiles r ) )
       )
      )
```

```
   )
( define ( mycreation amount h rotz)
   ( cond
      ((= amount 0)
       empty-image
       )
      ((> amount 0)
       ( beside
         ( mycreation ( - amount 1 )h (- rotz 1)) (rotate (* rotz 270)( triangle
h )) )
       )
      )
   )
( define ( my-creation amount h)
   ( mycreation amount h (- amount 1))
   )


( define ( random-blue-color ) ( color 0 0 255 ( dark-rgb-value)))
( define ( random-red-color ) ( color 255 0 0 ( dark-rgb-value )))
( define ( random-magenta-color ) ( color 255 0 255 ( dark-rgb-value ) ) )
( define ( random-cyan-color ) ( color 0 255 255 ( dark-rgb-value ) ) )
( define ( dark-rgb-value ) ( + ( random 128 ) 128 ) )
```

Ending Note

I hope you like my last program. I recommend messing around with it, I find it fun to see how the colors change a bit in each type a pyramid is recursively called.