Racket Assignment 2

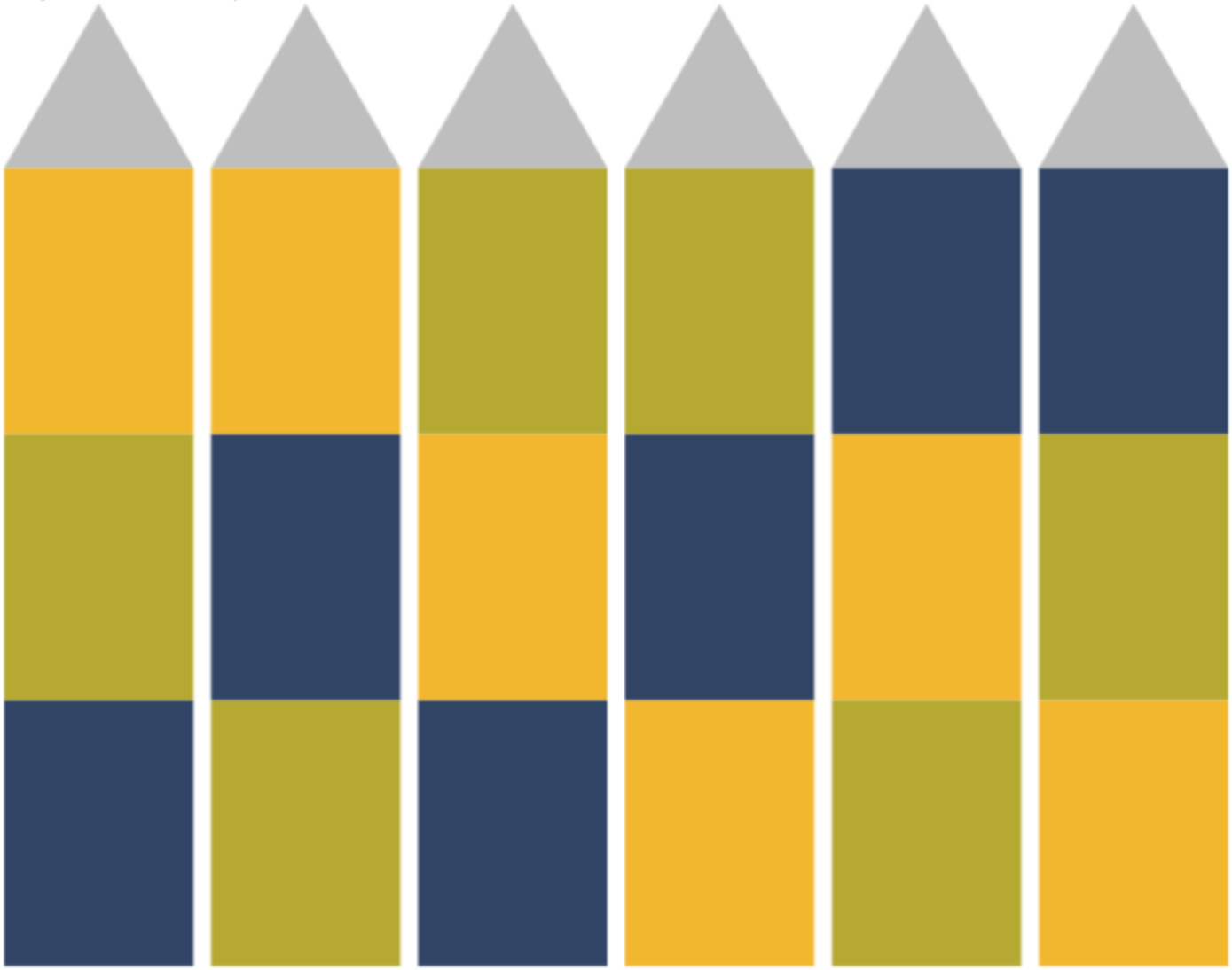
Learning Abstract

In this assignment, we learned how to use more advanced shape functions as well as Recursion.

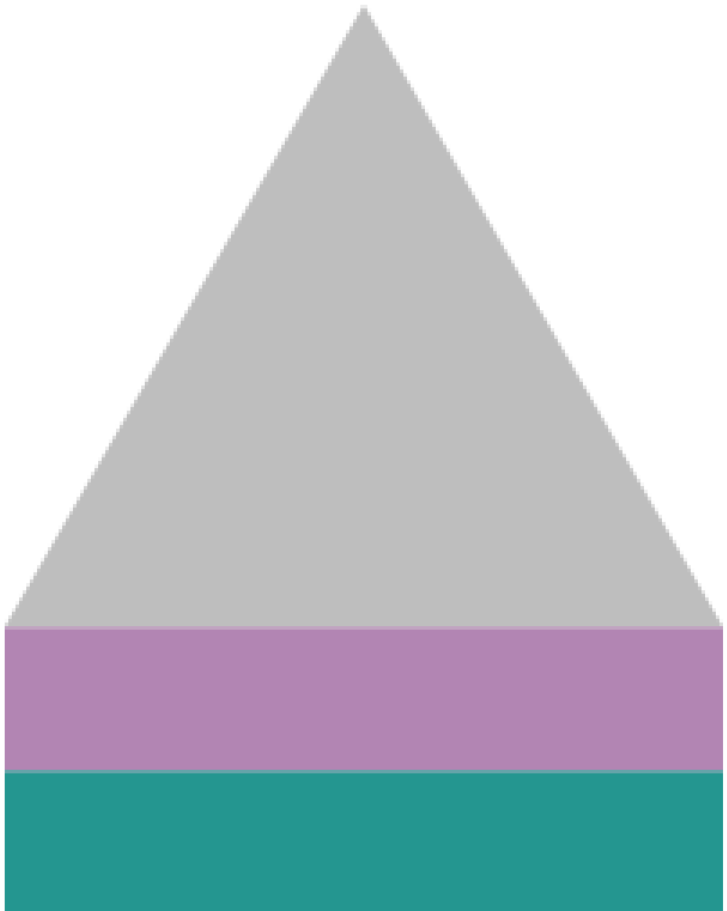
Task 1: Colorful Permutations of Tract Houses

Demo:

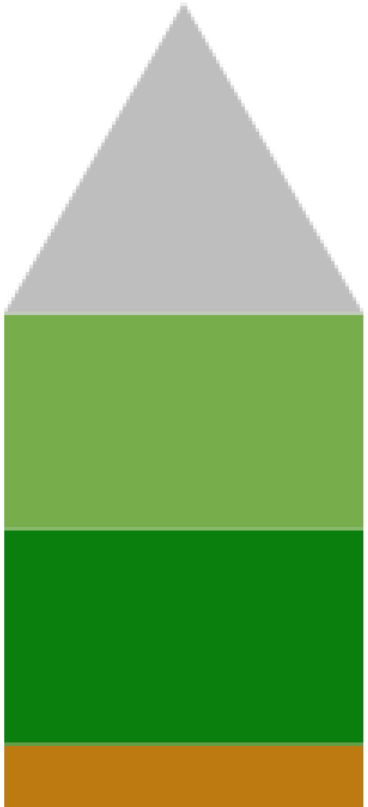
> (tract 700 150)

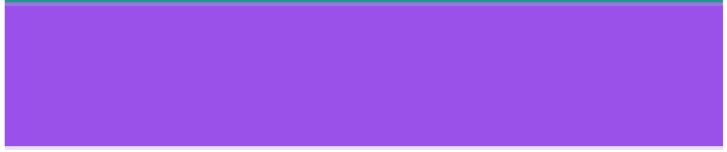


> (house 200 40 (rc) (rc) (rc))



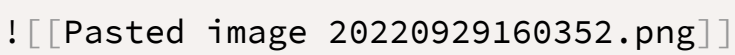
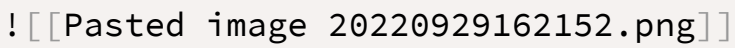
> (house 100 60 (rc) (rc) (rc))






Code:

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


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

(define (rc) (color (rgb-value) (rgb-value) (rgb-value)))

(define (floor x y color) (rectangle x y 'solid color))

(define (base x y color1 color2 color3) (above (floor x y color1)
(floor x y color2) (floor x y color3)))

(define (house x y color1 color2 color3) (above (triangle x 'solid
'grey) (base x y color1 color2 color3)))

(define (invisSquare)
  (square 10 'solid (color 255 0 0 0)))

(define (tract x y)
  (define housex (/ (- x (* 10 6)) 6))
  (define color1 (rc))
  (define color2 (rc))
  (define color3 (rc))
  (beside (house housex y color1 color2 color3) (invisSquare) (house
housex y color1 color3 color2) (invisSquare) (house housex y color2
color1 color3)(invisSquare) (house housex y color2 color3 color1)
(invisSquare) (house housex y color3 color1 color2) (invisSquare)
(house housex y color3 color2 color1))
  )```



### Task 2: Dice
```

Demo:



Code:
```racket
#lang racket/base

(define (roll-die)
  (define roll (random 6))
  (cond
    ((= roll 0)
      (roll-die)
    )
  ( else
    roll)
  ))

(define (roll-for-1)
  (define roll (roll-die))
  (display roll)
  (cond
    ( (= roll 1)
      (display "")
      )
  ( (> roll 1)
    (display " ")
    (roll-for-1)
    )
  )
 )

(define (roll-for-11)
  (roll-for-1)
  (define roll (roll-die))
  (cond
    ( (= roll 1)
      (display " ")
      (display roll)
      )
```

```
    ( (not (= roll 1))
      (display " ")
      (roll-for-11)
      )
    )
  )

(define (roll-for-odd)
  (define roll (roll-die))
  (display roll)
  (cond
    ( (= (remainder roll 2) 1)
      (display " ")
      )

    ( (not (= (remainder roll 2) 1))
      (display " ")
      (roll-for-odd)
      )
    )
  )

(define (roll-for-even)
  (define roll (roll-die))
  (display roll)
  (cond
    ( (= (remainder roll 2) 0)
      (display " ")
      )

    ( (not (= (remainder roll 2) 0))
      (display " ")
      (roll-for-even)
      )
    )
  )

(define (roll-for-odd-even-odd)
  (roll-for-odd)
  (roll-for-even)
  (roll-for-odd)
```

```
  )

(define (roll-two-dice-for-a-lucky-pair)
  (define roll1 (roll-die))
  (define roll2 (roll-die))

  (display "(") (display roll1)(display " ") (display roll2)
(display ")")
  (if (or (= (+ roll1 roll2) 7) (= (+ roll1 roll2) 11) (= roll1
roll2)) (display " ")
      (roll-two-dice-for-a-lucky-pair)))
```

Task 3: Number Sequences

Preliminary Demo:

```
> (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:

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

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

Code:

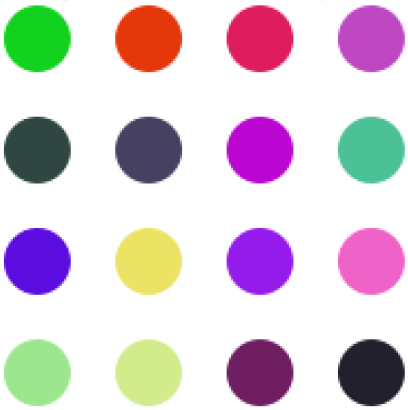
```
#lang racket/base

(define (square n)
  (* n n)
  )

(define (cube n)
  (* n n n )
  )

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

(define (sigmaCalc n a b)
  (cond
    ((= n a)
     (+ a b)
     )
    (else (cond
            ((< a (+ n 1))
             (cond
               ((eq? (modulo n a) 0)
                (sigmaCalc n (+ a 1) (+ b a))
                )
               (else
                (sigmaCalc n (+ a 1) b)
                )
               )
             )
            )
          )
    )
  )
```

```
(define (sigma n) (sigmaCalc n 1 0))

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

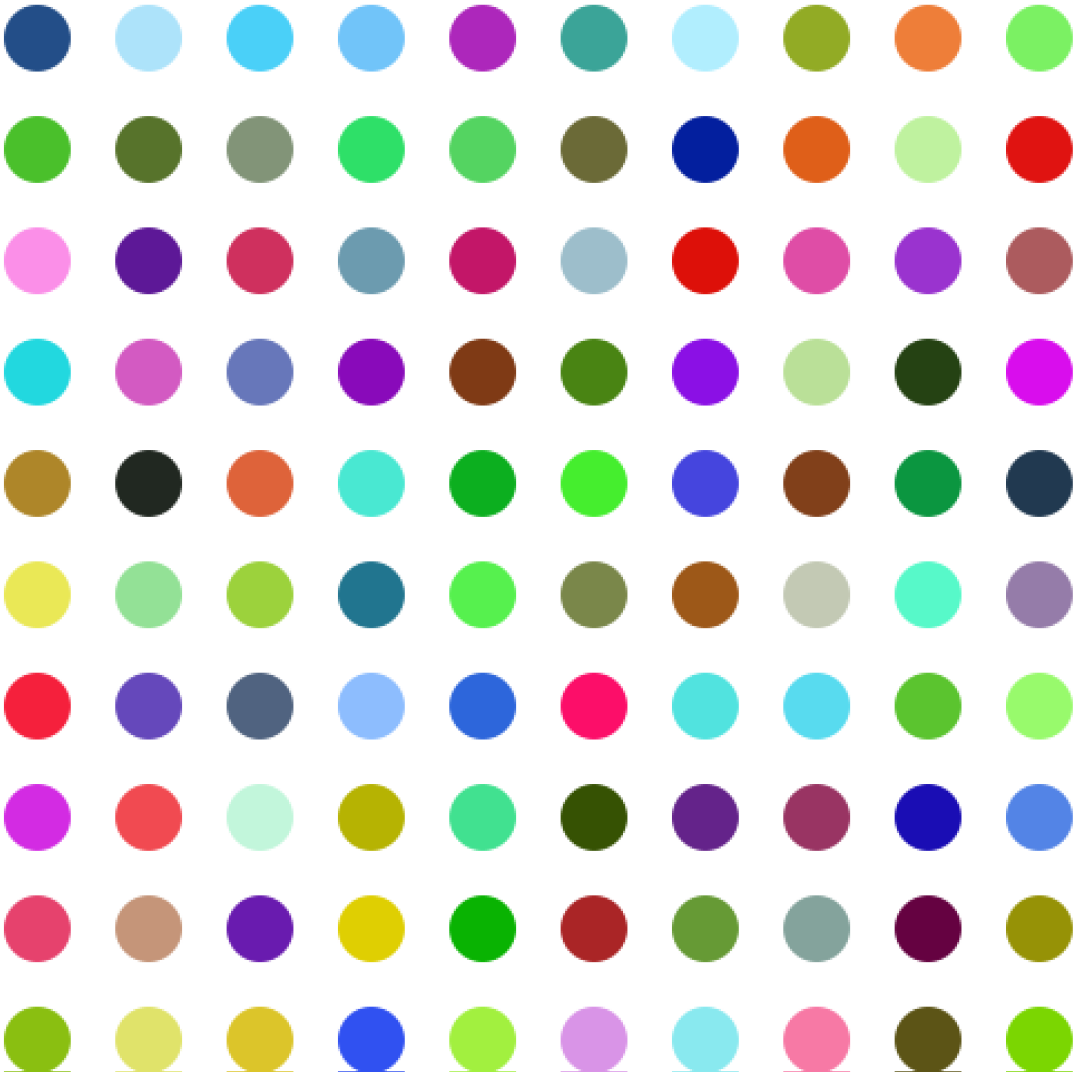
Task 4: Hirst Dots

Demo:

> (hirstDots 4)



> (hirstDots 10)



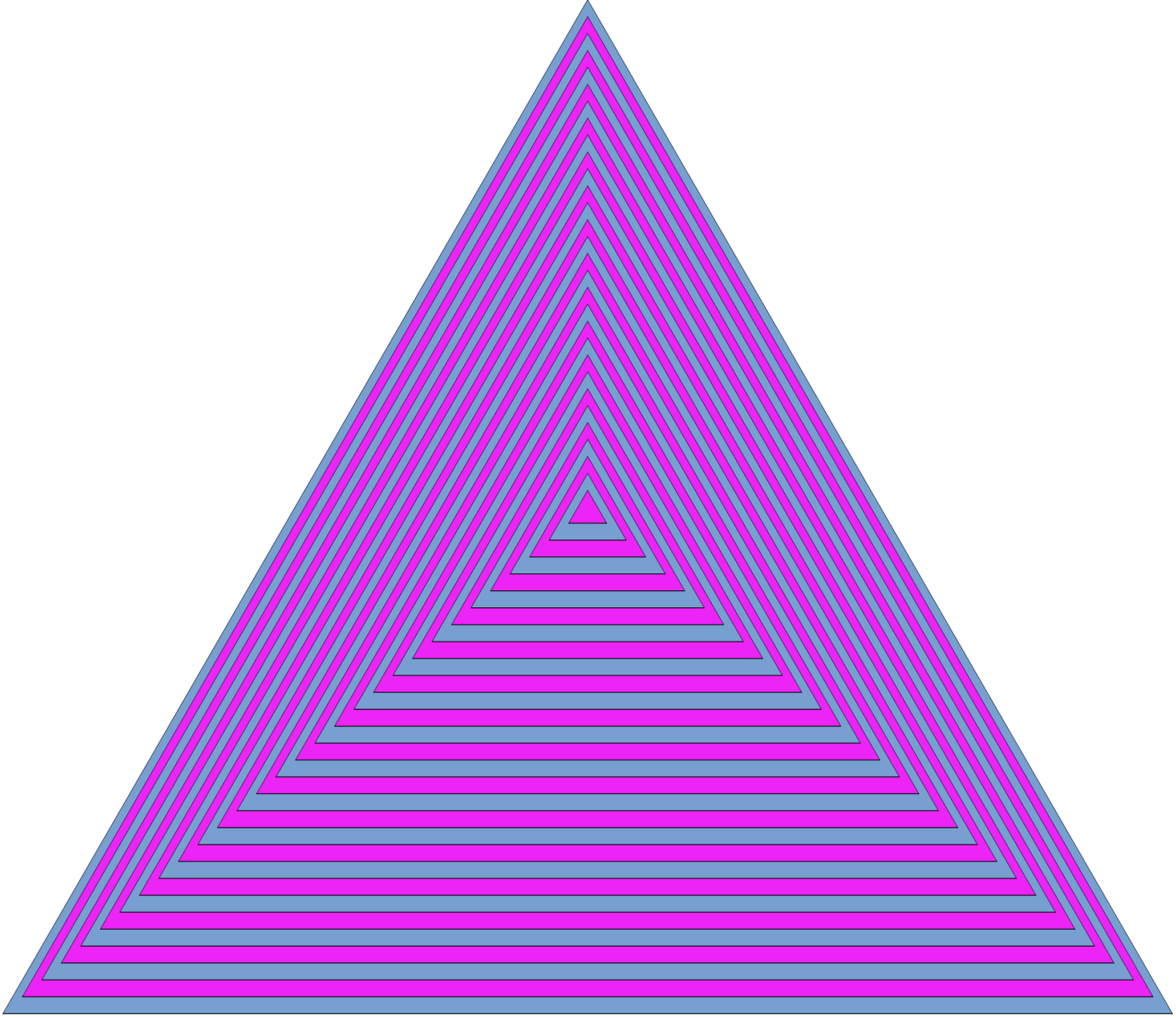
Coding:

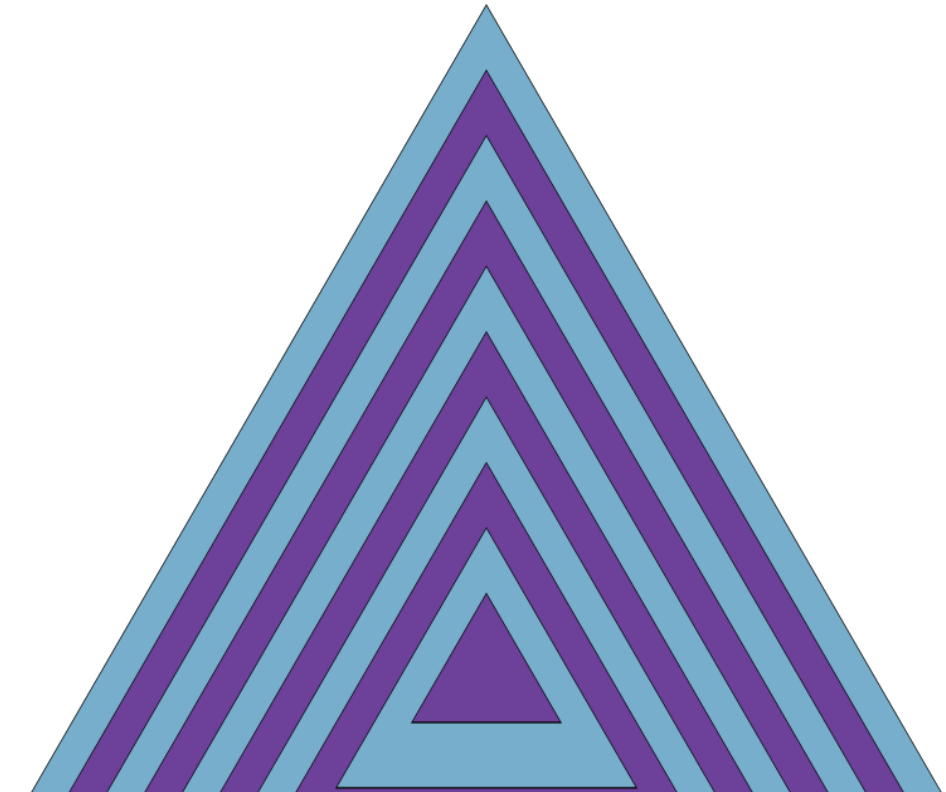
```
#lang racket/base
(require 2htdp/image)

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

(define (rc) (color (rgb-value) (rgb-value) (rgb-value)))

(define (dot) (circle 15 'solid (rc)))

(define (invisSquare)
  (square 20 'solid (color 255 0 0 0)))

(define (dotRow n)
  (cond
    ((= n 1)
```

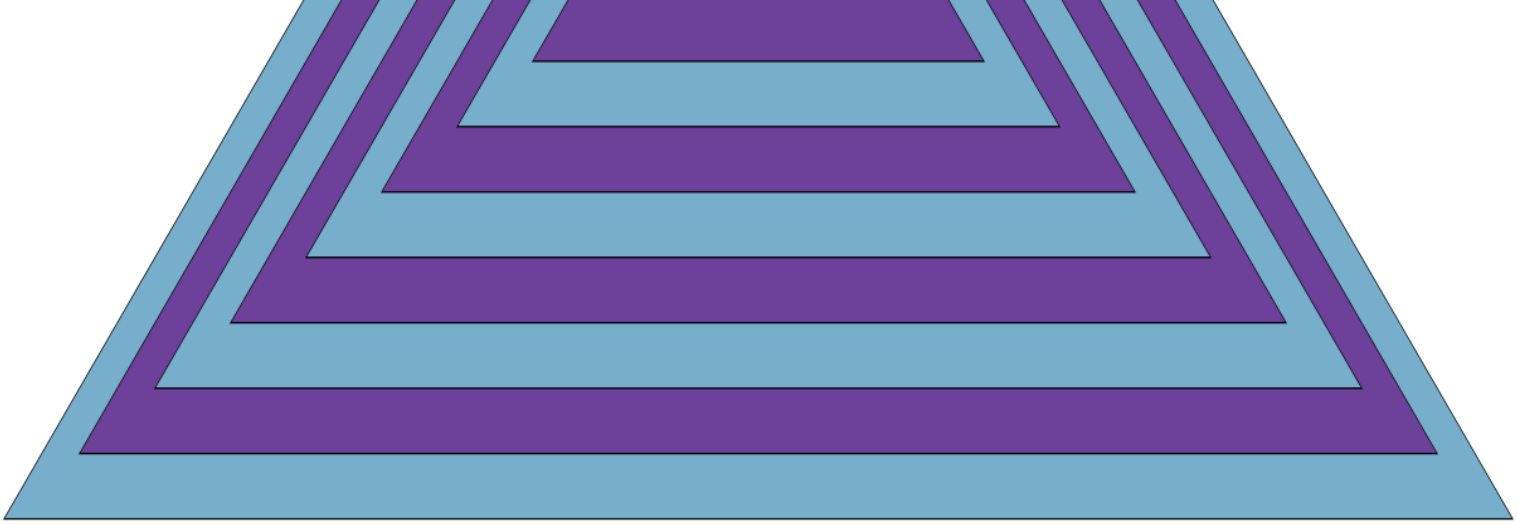
```
      (dot)
      )
     (else
      (beside (dotRow (- n 1)) (invisSquare) (dot))))
  )

(define (dotCol x y)
  (cond
    ((= y 1)
     (dotRow x)
     )
    (else
     (above  (dotRow x) (invisSquare) (dotCol x (- y 1)))))
  )

(define (hirstDots n)
  (dotCol n n)
  )
```

Task 5: Frank Stella

Demo:







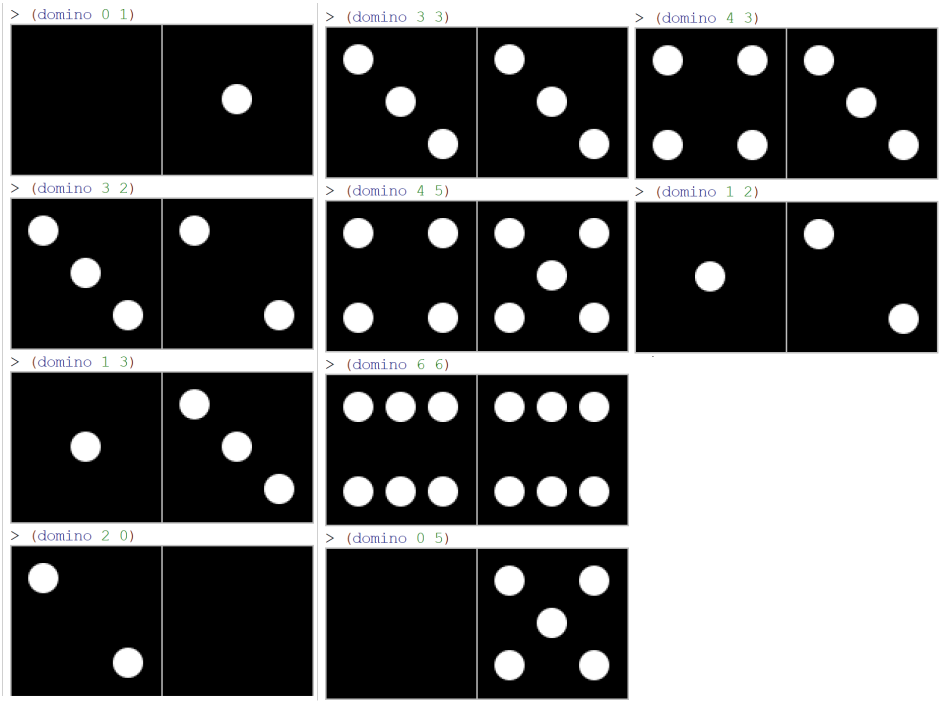
Code:

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

(define (stella size count c1 c2)

  (paint-nested-tris 1 size count c1 c2)
  )

( define (framed-tri size color )
   ( overlay
     ( triangle (- size 3) "solid" color )
     ( triangle size "solid" "black" )
     )
   )
( define ( nested-tris side count c1 c2 )
   ( define unit ( / side count ) )
   ( paint-nested-tris 1 count unit c1 c2 )
   )

( define ( paint-nested-tris from to unit c1 c2)
   ( define side-length ( * from unit ) )
   ( cond
      ( ( = from to )
        ( framed-tri side-length c1 )
        )
      ( ( < from to )
        (if (= (modulo from 2) 0)
```

```
            ( overlay
              ( framed-tri side-length c1 )
              ( paint-nested-tris ( + from 1 ) to unit c1 c2 )
              )

            ( overlay
              ( framed-tri side-length c2 )
              ( paint-nested-tris ( + from 1 ) to unit c1 c2 )
              )

            )
          )
        )
    )


(define (rc) (color (random 256) (random 256) (random 256)))
```

Task 6: Dominos

Demo:



Code:

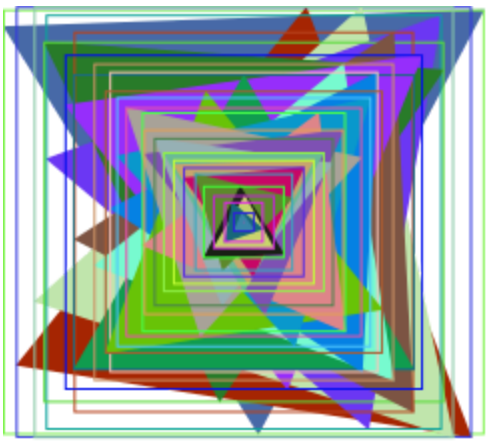
```
#lang racket
;-------------------------------------------------------------------
------------
; Requirements
;
; - Just the image library from Version 2 of "How to Design
Programs"
;
( require 2htdp/image )
;-------------------------------------------------------------------
------------
; Problem 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 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 )
)
)
```

Task 7: Creation

Demo:



Code:

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

(define (rc) (color (random 256) (random 256) (random 256)))

(define (creation)
  (rotate-tri 1 25 0)
  )
```

```
(define (rotate-tri side count degree)
  (cond
    ( (= count 0)
      empty-image
      )
    ( ( > count 0)
      (define temp-tri (rotate degree (triangle side 'solid (rc) ) ) )
)
      (overlay/offset
       (color-frame (rc) temp-tri) (* degree (random 1)) (* degree
(random 1))
       (rotate-tri (+ side 10) (- count 1) (random 360))
       )
      )
    )
  )
```