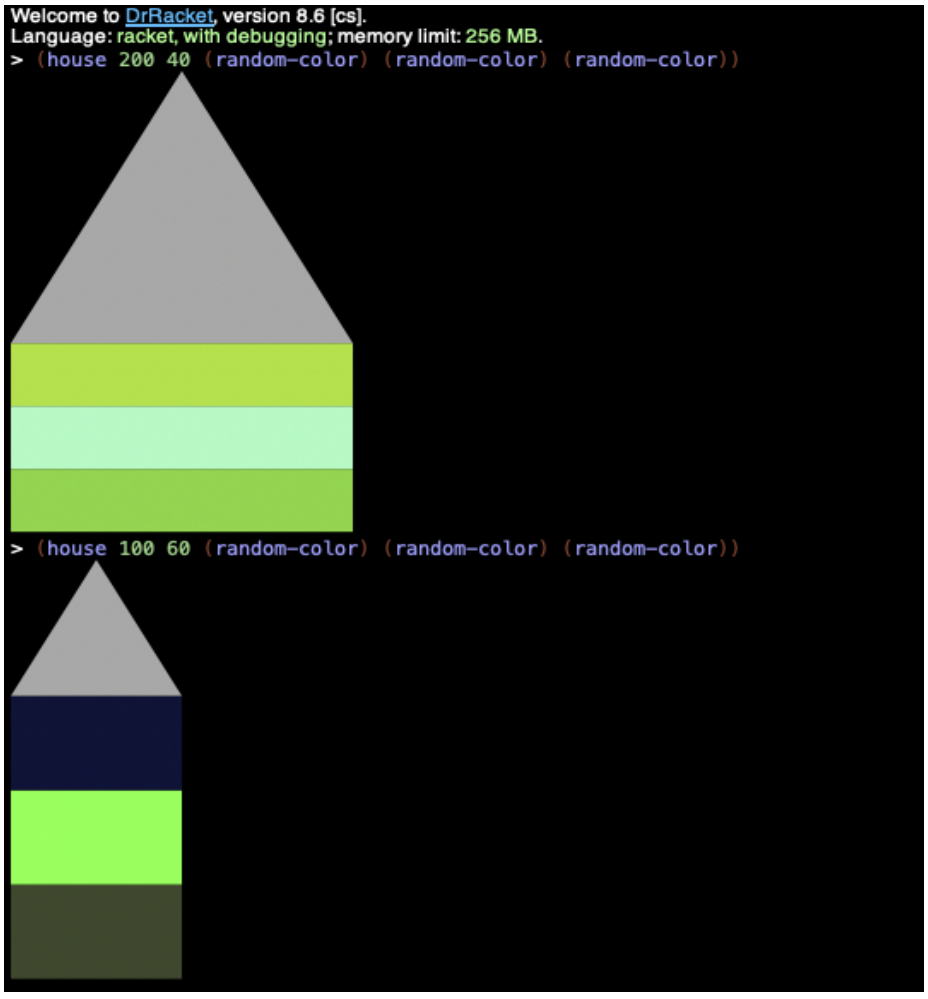
Racket Programming Assignment #2: Racket Functions and Recursion

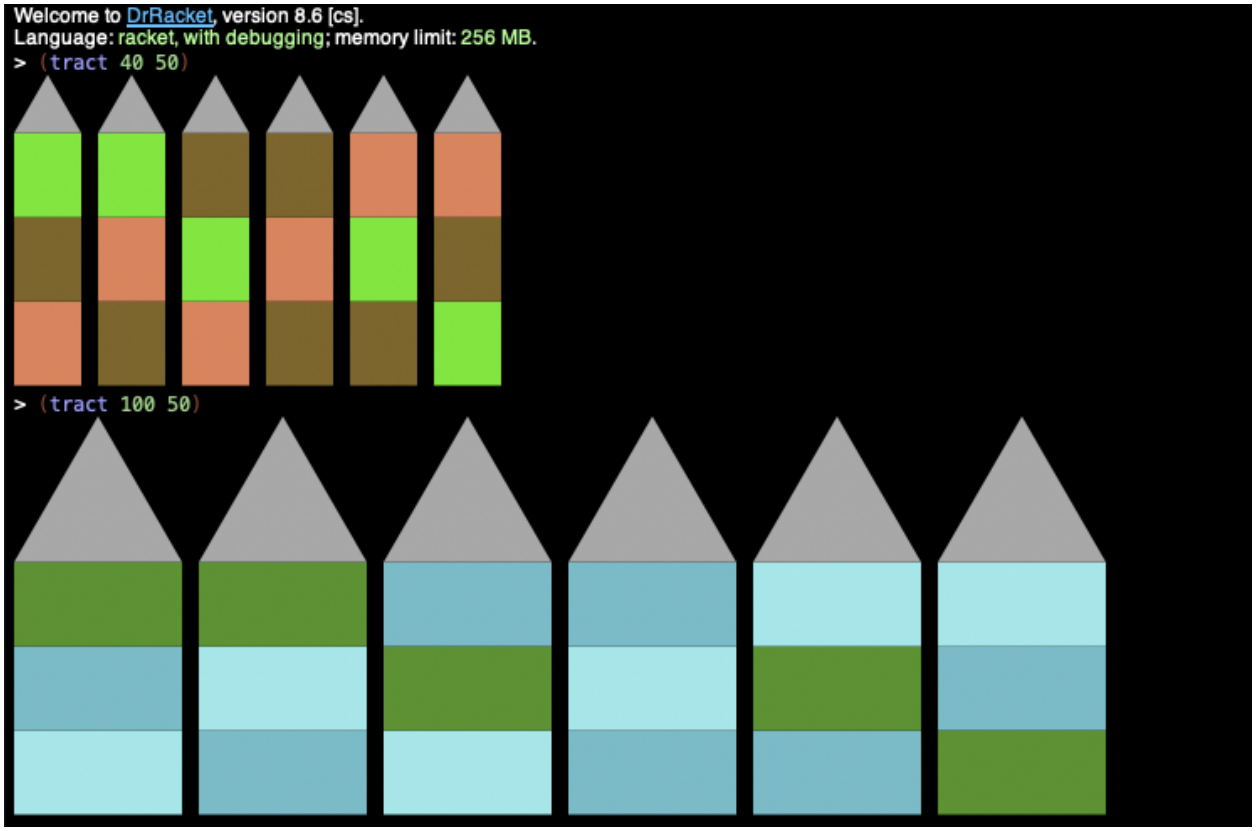
Learning Abstract
For this assignment, it features programs that produce images from the 2htdp/image which are mostly made from recursive and conditional functions.

Task 1: Colorful Permutations of Tract Houses

Demo for house



Demo for tract



The code for house and tract

```
#lang racket
(require 2htdp/image)
(define (rgb-value) ( random 256) )
(define (random-color) ( color (rgb-value) (rgb-value) (rgb-value) ))
;-----------------------------------------------------------------------------
; the house demo

(define (floor1 width height color1 )(rectangle width height 'solid color1) )
(define (floor2 width height color2)(rectangle width height 'solid color2 ))
(define (floor3 width height color3 )(rectangle width height 'solid   color3))
(define (roof width ) (triangle width "solid" "dark grey"))


(define (house width height color1 color2 color3 )
(define the-house
  ( above (roof width )
            (floor1 width height color1 )
            (floor2 width height color2 )
            (floor3 width height color3)
            ))
  the-house
    )
;-----------------------------------------------------------------------------
; the tract demo

(define space (square 10 "solid" "black"))
(define (f1 width height  )(rectangle width height 'solid (random-color)) )
(define (f2 width height)(rectangle width height 'solid (random-color) ))
(define (f3 width height )(rectangle width height 'solid   (random-color)))

(define (tract width height)
 (define 1floor (f1 width height ))
  (define 2floor (f2 width height ))
  (define 3floor (f3 width height ))
  (define houseroof (roof width))
   (define the-house ( above houseroof 1floor 2floor 3floor))
   (define the-house-2 ( above houseroof 1floor 3floor 2floor))
    (define the-house-3 ( above houseroof 2floor 1floor 3floor))
    (define the-house-4 ( above houseroof 2floor 3floor 2floor))
    (define the-house-5 ( above houseroof 3floor 1floor 2floor))
    (define the-house-6 ( above houseroof 3floor 2floor 1floor))
  (define the-tract ( beside the-house space the-house-2 space the-house-3 space the-house-4 space the-house-5 space the-house-6) )
  the-tract
 )
```

Task 2: Dice

Demo for dice

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

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

The code for dice

```
#lang racket
(define (roll-die) (+(random 6) 1))


(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-even) (*(+ (random 3) 1 ) 2))
(define (roll-odd) (- (roll-even) 1))

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

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

    )
  )
)


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

```
(define (roll-two-dice-for-a-lucky-pair)
  (display "(" )
  (define outcome (roll-die))
  (define outcome1 (roll-die))
  (display outcome) (display " ")
  (display outcome1) (display ")") (display " ")
 (define total ( + outcome outcome1 ))
  (define total-7 ( = total 7))
  (define total-11 ( = total 11) )
  (define equal ( = outcome outcome1))

  (cond
    ( (not (or total-7 total-11 equal))
       (roll-two-dice-for-a-lucky-pair)

    )
  )
  )
```

Task 3: Number Sequences

Preliminary Demo

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (square 3)
9
> (square 6)
36
> (cube 2)
8
> (cube 7)
343
> (sequence square 4)
1 4 9 16
> (sequence square 10)
1 4 9 16 25 36 49 64 81 100
> (sequence cube 8)
1 8 27 64 125 216 343 512
> (sequence cube 5)
1 8 27 64 125
```

Triangular Demo

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (triangular 1)
1
> (triangular 3)
6
> (triangular 7)
28
> (sequence triangular 10)
1 3 6 10 15 21 28 36 45 55
> (sequence triangular 15)
1 3 6 10 15 21 28 36 45 55 66 78 91 105 120
```

Sigma Demo

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 256 MB.
> (sigma 4)
7
> (sigma 7)
8
> (sigma 10)
18
> (sequence sigma 5)
1 3 4 7 6
> (sequence sigma 20)
1 3 4 7 6 12 8 15 13 18 12 28 14 24 24 31 18 39 20 42
```

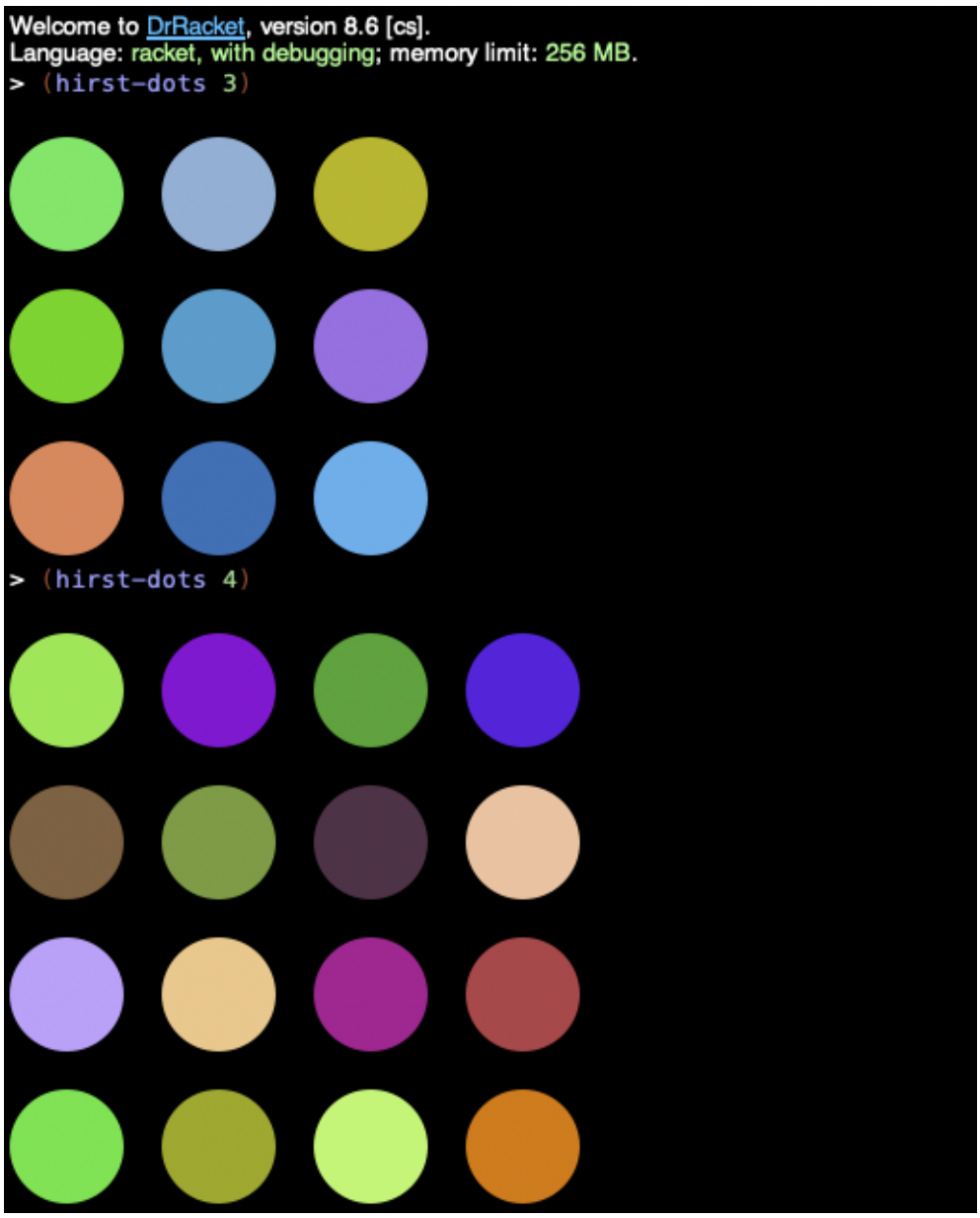
Codes for Preliminary, Triangular and Sigma

```
#lang racket

;Preliminary Demo

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


( define ( cube n )
(* n n n )
)
;-----------------------------------------------------------------------------
;Triangular Demo
( define ( triangular n )
  ( cond
    ( ( = n 1) 1)
    ( (= n 2) 3)
     ( ( > n 2)
       ( + ( triangular (- n 1)) n)
       )
     )
  )


;-----------------------------------------------------------------------------
;Sigma Demo

(define (number s d)
 ( cond
    ( (= d s )
      d)
   ( ( = ( modulo s d ) 0 )
     ( + d ( number s ( + d 1 ) ) )

     )
   ( else
     ( number s ( + d 1 ))
     )
   )
 )


(define (one) 1)
(define (sigma s ) (number s (one)))
```

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

; ( define ( sequence cube n )
;   ( cond
 ;    ((= n 1)
 ;     ( display ( cube 1 ) ) ( display " " )
 ;    )
 ;    ( else
 ;      ( sequence cube ( - n 1 ) )
 ;      ( display ( square n ) ) ( display " " )
 ;    )
 ;  )
;  )

; ( define ( sequence triangular n )
;   ( cond
 ;    ((= n 1)
 ;     ( display ( triangular 1 ) ) ( display " " )
 ;    )
 ;    ( else
 ;      ( sequence triangular ( - n 1 ) )
 ;      ( display ( sigma n ) ) ( display " " )
 ;    )
 ;  )
;  )


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

Task 4: Hirst Dots

Demo for Hirst Dots

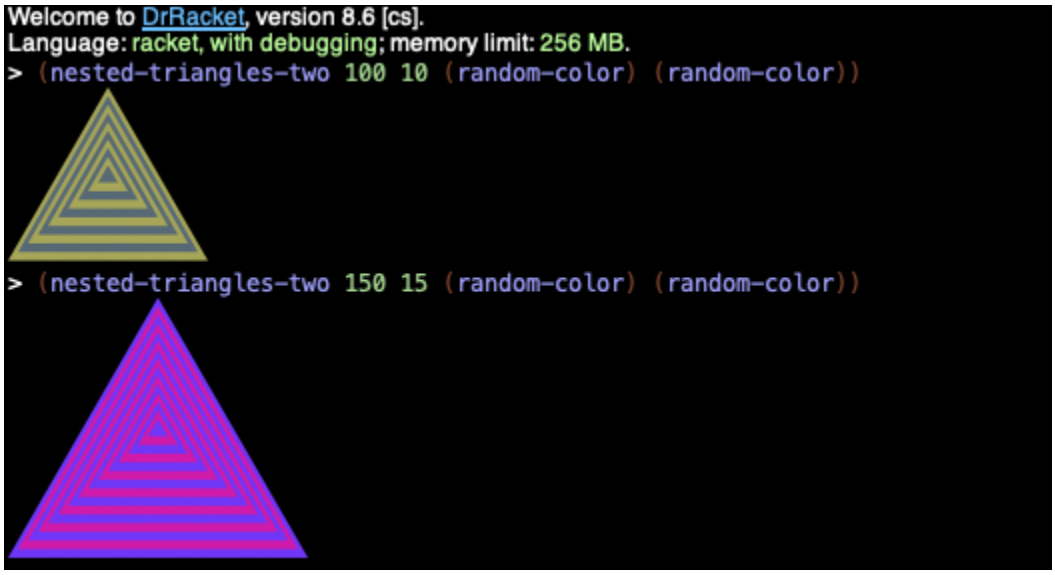


The code for Hirst Dots

```
#lang racket
(require 2htdp/image)
(define (rgb-value) ( random 256) )
(define (random-color) ( color (rgb-value) (rgb-value) (rgb-value) ))

(define space (square 20 "solid" "black"))
(define radius 30)
(define (dots radius color)(circle radius 'solid (random-color)))

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

(define (rectangle-of-circles a b )
  (cond
    ( ( = a 0 )
      empty-image
     )
    (( > a 0)
      ( above
        (rectangle-of-circles ( - a 1 ) b  ) space (row-of-circles b))

          )
     )
  )

(define (hirst-dots n)
  (rectangle-of-circles n n)
  )
```

Task 5: Channeling Frank Stella

Demo for Frank Stella

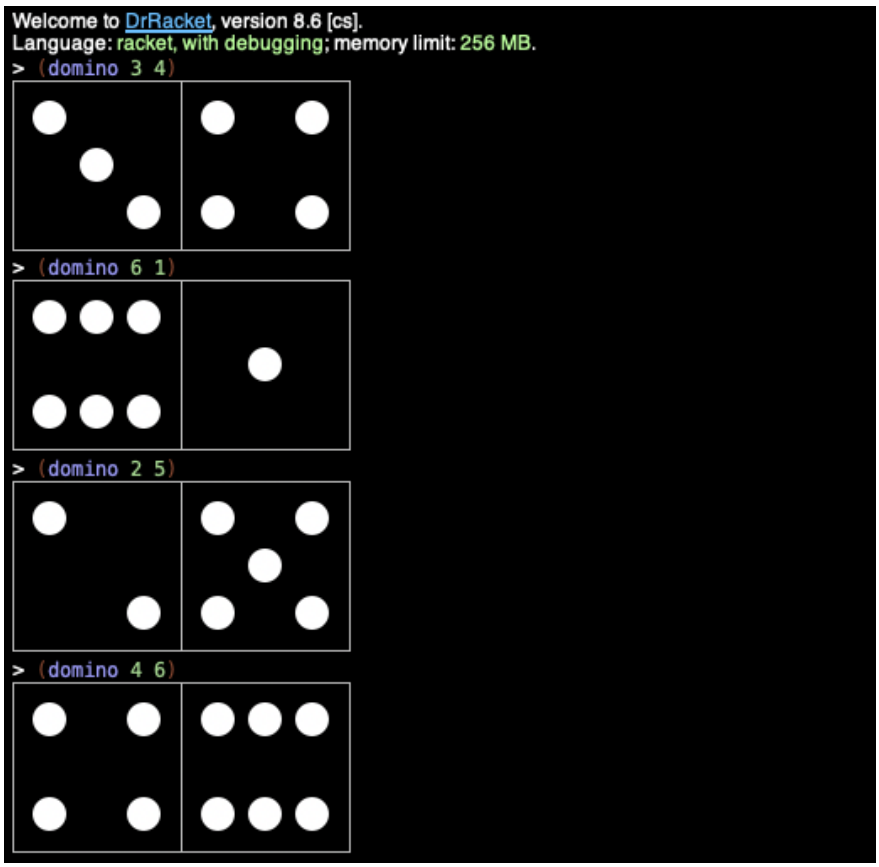


The code for Frank Stella

```
#lang racket
(require 2htdp/image)
(define (rgb-value) ( random 256) )
(define (random-color) ( color (rgb-value) (rgb-value) (rgb-value) ))

(define (nested-triangles-two height count color1 color2)
  (define pyramid ( / height count))
  ( paint-nested-triangles-two 1 count pyramid color1 color2)
  )

(define (paint-nested-triangles-two made by pyramid color1 color2)
  (define height ( * made pyramid))
  ( cond
     ( ( = made by)
          ( if ( even? made)
               (triangle height "solid" color1)
               (triangle height "solid" color2)
               )
          )
       ( ( < made  by)
         ( if (even? made )
             ( overlay
               (triangle height "solid" color1)
               ( paint-nested-triangles-two ( + made 1 ) by pyramid color1 color2)
               )
            ( overlay
               (triangle height "solid" color2)
               ( paint-nested-triangles-two ( + made 1 ) by pyramid color1 color2)
               )
            )
         )
       )
     )
```

Task 6: Dominos

Final demo for Dominos



Collected code for Dominos

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


( 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 ) nd nd
( overlay/offset ( pip ) d nd
( overlay/offset ( pip ) nd d blank-tile)
)
)
)
 )

( define basic-tile5 ( overlay ( pip ) basic-tile4 ) )
( define basic-tile6
( overlay/offset ( pip ) d d
( overlay/offset ( pip ) nd nd
( overlay/offset ( pip ) d nd
( overlay/offset ( pip ) nd d
( overlay/offset ( pip ) 0 d
( overlay/offset ( pip ) 0 nd blank-tile)
)
)
)
 )
)
  )
```

```
;-----------------------------------------------------------------------------
; 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

Creation (Image)

Code for Creation

```
#lang racket
(require 2htdp/image)
(define (rgb-value) ( random 256) )
(define (random-color) ( color (rgb-value) (rgb-value) (rgb-value) ))

;-------------------------------------------------------------------------------------------
; my-creation function to create the randomly coloured face of a rainbow bear
; made out of circles and a triangle based on radius

(define (my-creation radius)
; variables defined to create part of a rainbow bear's face
(define (face-center radius) (circle radius 'solid (random-color)))
 (define (ear-1 radius) (circle ( / radius 2) 'solid (random-color)))
  (define (ear-2 radius) (circle ( / radius 2) 'solid (random-color)))
 (define (eye-1 radius) (circle ( / radius 4) 'solid (random-color)))

  (define (eye-2 radius) (circle (/ radius 4) 'solid (random-color)))
  (define (nose radius) (triangle (/ radius 2.5) 'solid (random-color)))

;-------------------------------------------------------------------------------------------
;Creating final face of a rainbow bear
  (overlay/align "left" "top"
  (overlay/align "left" "middle" (eye-1 radius)
  (overlay/align "right" "middle" (eye-2 radius)
  (overlay/align "middle" "middle" (nose radius)
  (overlay/offset
   (rotate -270
   ( crop 0 0 ( / radius 2.5 ) ( * ( / radius 2.5) 2.25 ) ( circle ( / radius 2.5) 'solid (random-color)))) 0 (/ radius -1.5)
    (overlay/offset
   (rotate -270
     ( crop 0 0 radius ( * radius 2) ( circle radius 'solid (random-color)))) 0 (/ radius -2) (face-center radius)))))) (ear-1 radius)

     (overlay/align "right" "top" (face-center radius) (ear-2 radius)

    )
  )
  )

```