# Racket Programming Assignment #2: Racket Functions and Recursion

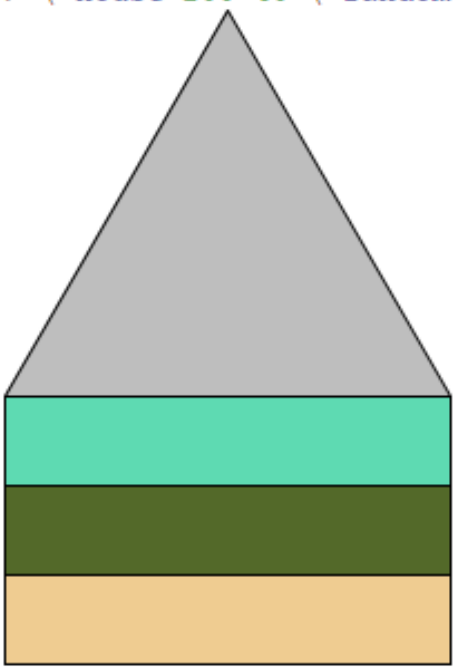
Brandon LaPointe

## Learning Abstract

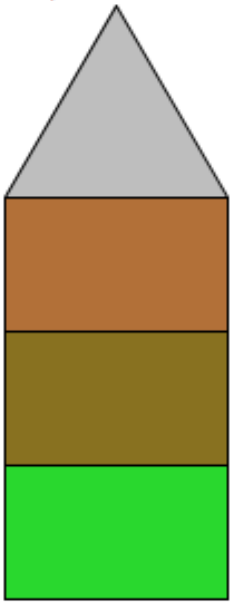
This assignment features more intuitive interactions in the Racket programming language using functions and recursion. I learned a bit about function calls and recursive computations within Lisp. The first task required a function called house to create an image of a house consisting of 3 floors and a roof with all 3 floors sharing the same size characteristics but being different colors. The task also required a function to create a tract of 6 houses, side by side, representing all of the permutations on 3 random colors. The second task involved creating a program to mimic the rolling of a single die which was then used to create recursive functions to roll until certain outcomes were achieved. The third task consisted of creating recursive functions to run through number sequences up to the input number before outputting the results. The fourth task required a recursive program to be made to create a grid of Hirst dots. The fifth task involved creating monochromatic and two-tone images resembling a simplified version of the work of artist Frank Stella using recursive functions. The sixth task consisted of creating a program to resemble dominos using overlay/offset within 2htdp/image to create each tile. The last task required a creative program to develop a personal creation by manipulating images found within the 2htdp/image library.

## Task 1: Colorful Permutations of Tract Houses – House

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( house 200 40 ( random-color ) ( random-color ) ( random-color ) )
```



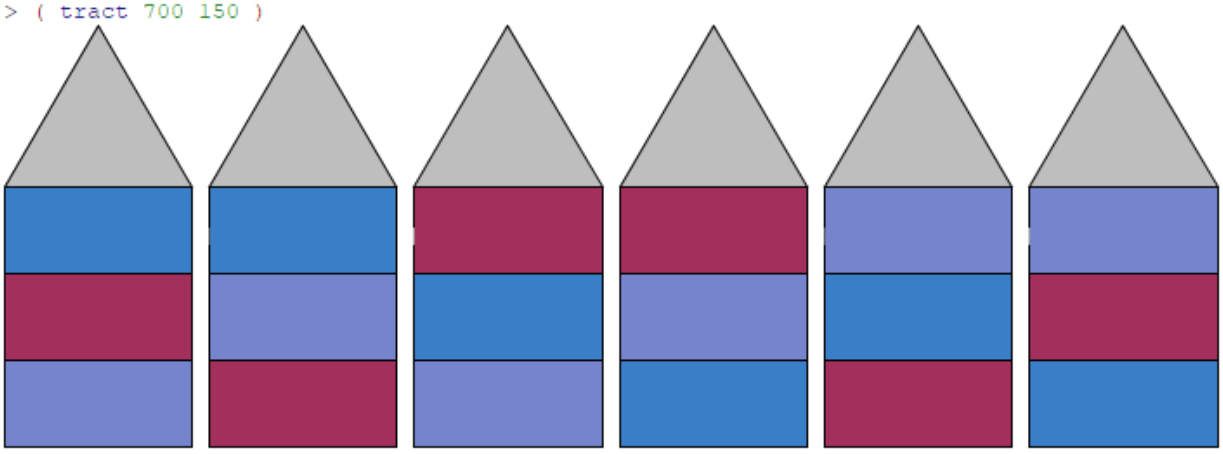
```
> ( house 100 60 ( random-color ) ( random-color ) ( random-color ) )
```



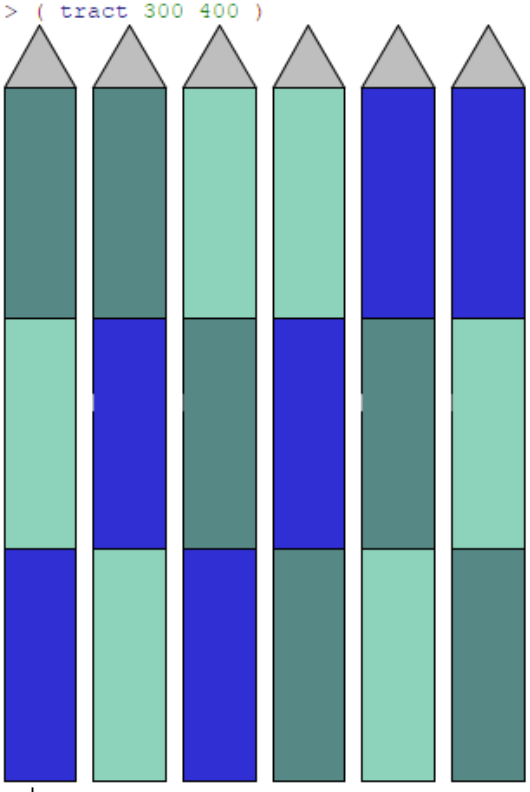
```
>
```

## Task 1: Colorful Permutations of Tract Houses – Tract

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



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



```
>
```

# Task 1: Code

```
#lang racket
; -----------------------------------------------------------------------
; Program that renders houses and tracts of houses
;

; -----------------------------------------------------------------------
; Import the image library from htdp, 2nd edition
;
(require 2htdp/image)

; -----------------------------------------------------------------------
; Generate a house of 3 random colored rectangle floors and a grey triangle roof
;
( define ( house w h floor1 floor2 floor3 )
  ( above
    ( overlay
      ( triangle w "outline" "black" )
      ( triangle w "solid" "grey" )
    )
    ( overlay
      ( rectangle w h "outline" "black" )
      ( rectangle w h "solid" floor3 )
    )
    ( overlay
      ( rectangle w h "outline" "black")
      ( rectangle w h "solid" floor2 )
    )
    ( overlay
      ( rectangle w h "outline" "black" )
      ( rectangle w h "solid" floor1 )
    )
  )
)

; -----------------------------------------------------------------------
; Generate a tract of 6 houses side by side, seperated by a space of 10 pixels using the same random color scheme
;
( define ( tract w h )
  ( define width-of-house ( / ( - w 50 ) 6 ) )
  ( define height-of-floor ( / h 3 ) )
  ( define color1 ( random-color ) )
  ( define color2 ( random-color ) )
  ( define color3 ( random-color ) )
  ( define space ( square 10 "solid" "white" ) )
  ( beside
    ( house width-of-house height-of-floor color3 color2 color1 )
    space
    ( house width-of-house height-of-floor color2 color3 color1 )
    space
    ( house width-of-house height-of-floor color3 color1 color2 )
    space
    ( house width-of-house height-of-floor color1 color3 color2 )
    space
    ( house width-of-house height-of-floor color2 color1 color3 )
    space
    ( house width-of-house height-of-floor color1 color2 color3 )
  )
```

```
)

; ---------------------------------------------------------------------
; Helper functions for rendering the floors
;
( define ( random-color )
  ( color
    ( rgb-value ) ( rgb-value ) ( rgb-value )
  )
)

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

## Task 2: Dice

```
> ( roll-die )
2
> ( roll-die )
6
> ( roll-die )
5
> ( roll-die )
4
> ( roll-die )
3
> ( roll-for-1 )
3 5 5 5 2 2 4 4 1
> ( roll-for-1 )
3 2 1
> ( roll-for-1 )
6 1
> ( roll-for-1 )
1
> ( roll-for-1 )
2 5 6 2 2 2 6 2 6 3 4 6 3 6 1
> ( roll-for-11 )
6 1 5 1 3 5 1 1
> ( roll-for-11 )
2 3 1 4 5 4 2 6 3 3 4 3 2 4 6 5 3 5 2 2 5 6 3 3 5 6 2 4 1 1
> ( roll-for-11 )
5 2 3 6 6 1 6 6 3 6 1 4 5 2 3 2 5 1 6 6 2 1 2 1 6 2 3 6 1 1
> ( roll-for-11 )
1 1
> ( roll-for-11 )
4 4 3 3 5 6 6 6 2 4 6 6 5 6 2 2 6 5 5 3 5 4 3 6 6 4 6 5 4 3 4 3 2 2 5 1 4 4 2 6 2 5 1 1
> ( roll-for-odd-even-odd )
5 1 1 3 3 4 2 2 4 3 5 1 2 3
> ( roll-for-odd-even-odd )
3 3 2 1 4 2 6 1 1 4 4 4 2 3 1 1 4 3
> ( roll-for-odd-even-odd )
3 6 5
> ( roll-for-odd-even-odd )
1 1 4 3 5 6 6 1 4 2 4 6 2 4 6 4 6 3 3 6 5 1 6 4 5 3 1 4 1
> ( roll-for-odd-even-odd )
6 1 2 6 6 2 5 1 4 4 5 4 5
> ( roll-two-dice-for-a-lucky-pair )
(4 1) (3 4)
> ( roll-two-dice-for-a-lucky-pair )
(4 3)
> ( roll-two-dice-for-a-lucky-pair )
(1 2) (5 2)
> ( roll-two-dice-for-a-lucky-pair )
(3 6) (2 6) (4 4)
> ( roll-two-dice-for-a-lucky-pair )
(4 5) (5 1) (6 1)
> ( roll-two-dice-for-a-lucky-pair )
(3 2) (2 6) (4 2) (4 6) (2 5)
> ( roll-two-dice-for-a-lucky-pair )
(4 2) (3 6) (1 6)
> ( roll-two-dice-for-a-lucky-pair )
(5 1) (4 3)
> ( roll-two-dice-for-a-lucky-pair )
(2 2)
> ( roll-two-dice-for-a-lucky-pair )
(1 4) (1 3) (1 1)
>
```

# Task 2: Code

```
#lang racket
; -----------------------------------------------------------------------
; Program that generates random numbers between 1 - 6 (inclusive) to mimic
; a rolling die. The program also uses recursion to mimic rolling the die
; until a 1 is rolled, rolling the die until two 1's are rolled consecutively,
; rolling the die until an odd, even, odd combination are rolled consecutively,
; and rolling a pair of dice until a lucky pair is rolled.
; Lucky Pair = ( 7, 11, or doubles )
;
; Commands:
; ( roll-die )
; ( roll-for-1 )
; ( roll-for-11 )
; ( roll-for-odd-even-odd )
; ( roll-two-dice-for-a-lucky-pair )

; -----------------------------------------------------------------------
; Generate a random number between 1 - 6
( define ( roll-die )
  ( + ( random 6 ) 1 )
)

; -----------------------------------------------------------------------
; Rolls die until a 1 1s rolled
( define ( roll-for-1 )
  ( define result 0 )
  ( set! result ( roll-die ) )
  ( cond
    ( ( = result 1 )
      ( display result )
    )
    ( ( not ( = result 1 ) )
      ( display result ) ( display " " )
      ( roll-for-1 )
    )
   )
)
; -----------------------------------------------------------------------
; Rolls die until two 1s in a row are rolled
( define ( roll-for-11 )
  ( define result 0 )
  ( set! result ( roll-die ) )
  ( cond
    ( ( = result 1 )
      ( display result ) ( display " " )
      ( set! result ( roll-die ) )
      ( cond
        ( ( = result 1 )
          ( display result )
        )
        ( ( not ( = result 1 ) )
          ( display result ) ( display " " )
          ( roll-for-11 )
        )
      )
    )
    ( ( not ( = result 1 ) )
      ( display result ) ( display " " )
      ( roll-for-11 )
```

```
    )
   )
)

; ------------------------------------------------------------------------
; Rolls die until three consecutive die produce an odd, even, odd combo
( define ( roll-for-odd-even-odd )
  ( define result 0 )
  ( set! result ( roll-die ) )
  ( cond
    ( ( or ( = result 1 ) ( = result 3 ) ( = result 5 ) )
      ( display result ) ( display " " )
      ( set! result ( roll-die ) )
      ( cond
        ( ( or ( = result 2 ) ( = result 4 ) ( = result 6 ) )
          ( display result ) ( display " " )
          ( set! result ( roll-die ) )
          ( cond
            ( ( or ( = result 1 ) ( = result 3 ) ( = result 5 ) )
              ( display result )
            )
            ( ( or ( = result 2 ) ( = result 4 ) ( = result 6 ) )
              ( display result ) ( display " " )
              ( roll-for-odd-even-odd )
            )
          )
        )
        ( ( or ( = result 1 ) ( = result 3 ) ( = result 5 ) )
          ( display result ) ( display " " )
          ( roll-for-odd-even-odd )
        )
      )
    )
    ( ( or ( = result 2 ) ( = result 4 ) ( = result 6 ) )
      ( display result ) ( display " " )
      ( roll-for-odd-even-odd )
    )
  )
)

; ------------------------------------------------------------------------
; Rolls a pair of dice until a lucky pair is rolled ( 7 , 11 , or doubles )
( define ( roll-two-dice-for-a-lucky-pair )
  ( define die1 0 )
  ( define die2 0 )
  ( define ( roll ) ( and ( set! die1 ( roll-die ) ) ( set! die2 ( roll-die ) ) ) )
  ( roll )
  ( cond
    ( ( or ( = ( + die1 die2 ) 7 ) ( = ( + die1 die2 ) 11 ) ( eq? die1 die2 ) )
      ( display "(") ( display die1 ) ( display " " ) ( display die2 ) ( display ")" )
    )
    ( else
      ( display "(") ( display die1 ) ( display " " ) ( display die2 ) ( display ")" ) ( display " " )
      ( roll-two-dice-for-a-lucky-pair )
    )
  )
)
```

## Task 3: Number Sequences

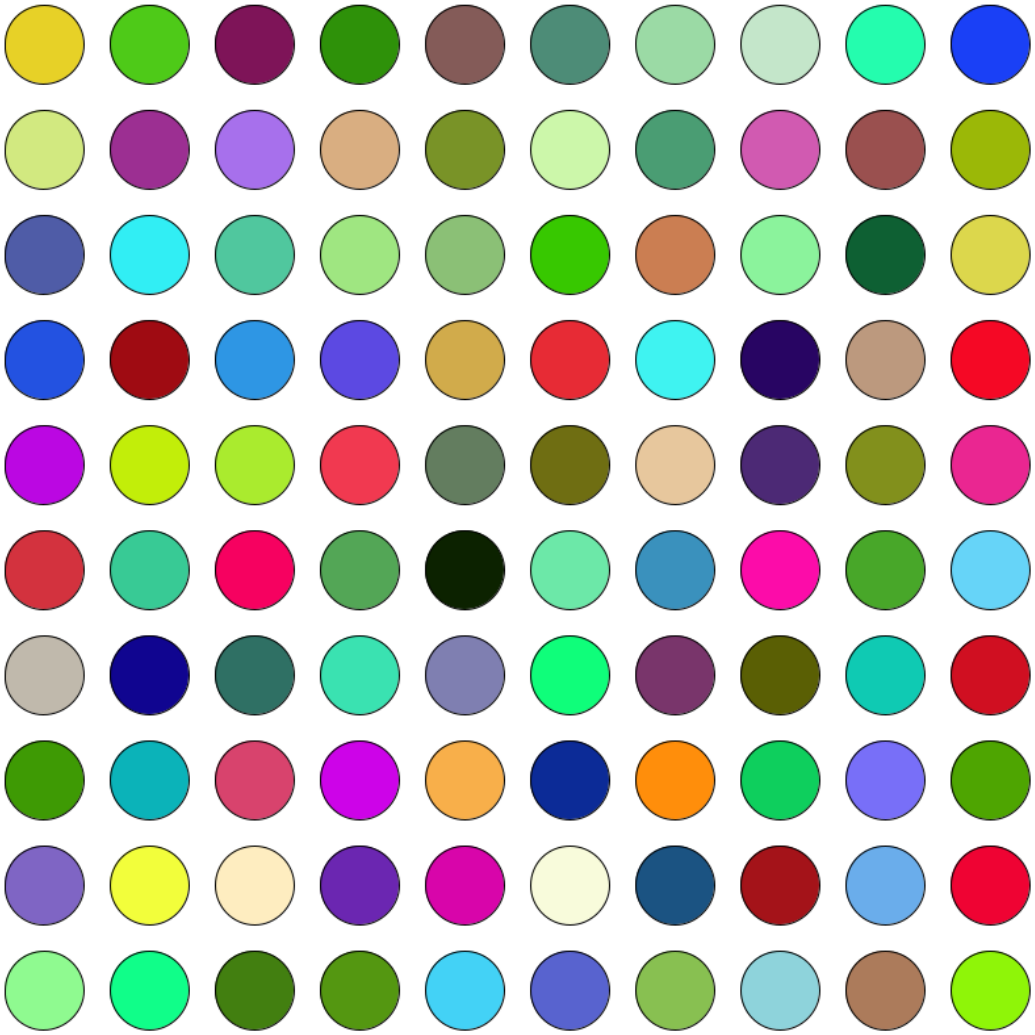
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( square 5 )
25
> ( square 10 )
100
> ( sequence square 15 )
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225
> ( cube 2 )
8
> ( cube 3 )
27
> ( sequence cube 15 )
1 8 27 64 125 216 343 512 729 1000 1331 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
>
```

# Task 3: Code

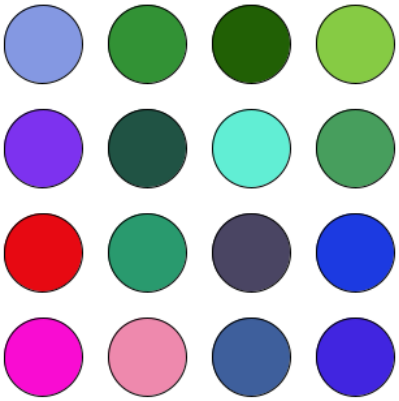
```
#lang racket
; -----------------------------------------------------------------------
; Program that mimics the "Preliminary Code" that consists of three functions:
; (1) a function to compute the square of a number,
; (2) a function to compute the cube of a number, and
; (3) a higher order function to compute the first n elements in a number sequence,
;     assuming that functions exist to compute the nth element of the sequence.
;

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

; -----------------------------------------------------------------------
; Sigma Numbers
;
( define ( sigma n )
  ( sigmath n n )
)

( define ( sigmath n orig )
  ( cond
    ( ( = n 1 )
     1
    )
    ( ( = ( modulo orig n ) 0 )
     ( + n ( sigmath ( - n 1 ) orig ) )
    )
    ( else
     ( sigmath ( - n 1 ) orig )
    )
  )
)
```

# Task 4: Hirst Dots

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( hirst-dots 10 )
```



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



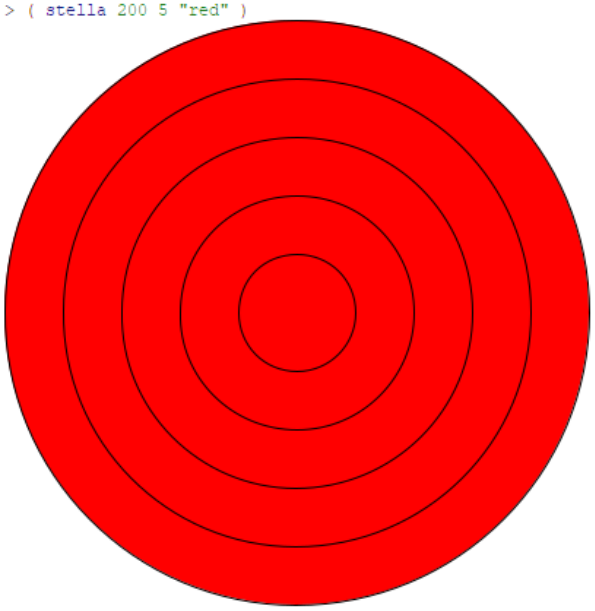
```
>
```

# Task 4: Code

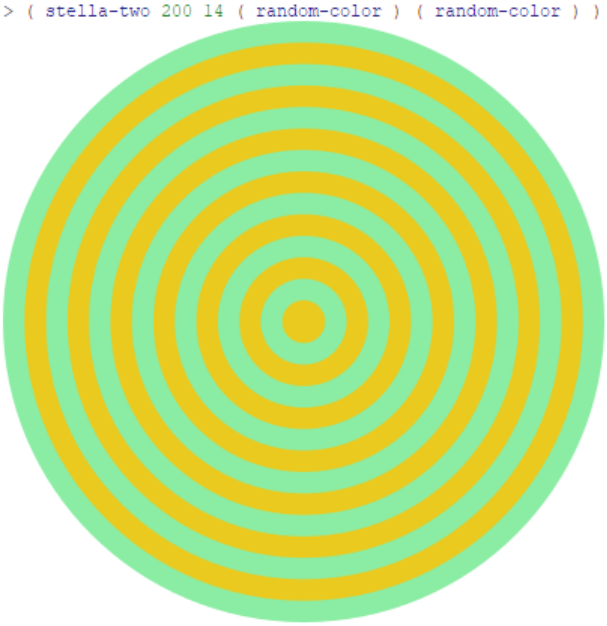
```
#lang racket
; -----------------------------------------------------------------------
; Program that creates square arrangements of Hirst dots. The dots have
; a diameter of 30 pixels each and are spaced 20 pixels apart.
;
; -----------------------------------------------------------------------
; Import the image library from htdp, 2nd edition
;
(require 2htdp/image)
; -----------------------------------------------------------------------
; Generate a row of tiles of specified length and tile type
;
( define ( row-of-tiles n tile )
  ( cond
    ( ( = n 0 )
      empty-image
    )
    ( ( > n 0 )
      ( beside ( row-of-tiles ( - n 1 ) random-color-tile ) ( space ) ( random-color-tile ) )
    )
  )
)

; -----------------------------------------------------------------------
; Generate a rectangle of tiles of specified row count, column count,
; and tile type
;
( define ( rectangle-of-tiles r c tile )
  ( cond
    ( ( = r 0 )
      empty-image
     )
    ( ( > r 0 )
      ( above
        ( rectangle-of-tiles ( - r 1 ) c  random-color-tile ) ( space ) ( row-of-tiles c random-color-tile ) )
    )
  )
)

; -----------------------------------------------------------------------
; Generate a square grid of randomly colored circlular tiles of specified grid size
;
( define ( hirst-dots n )
  ( rectangle-of-tiles n n random-color-tile )
)

; -----------------------------------------------------------------------
; Implement the tile generators
;
( define ( random-color-tile )
  ( overlay
    ( circle 30 "outline" "black" )
    ( circle 30 "solid" ( random-color ) )
  )
)

( define ( space )
  ( square 20 "solid" "white" )
)

; -----------------------------------------------------------------------
; Random Color Functions
;
( define ( random-color )
  ( color
    ( rgb-value ) ( rgb-value ) ( rgb-value )
  )
)

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

# Task 5: Channeling Frank Stella

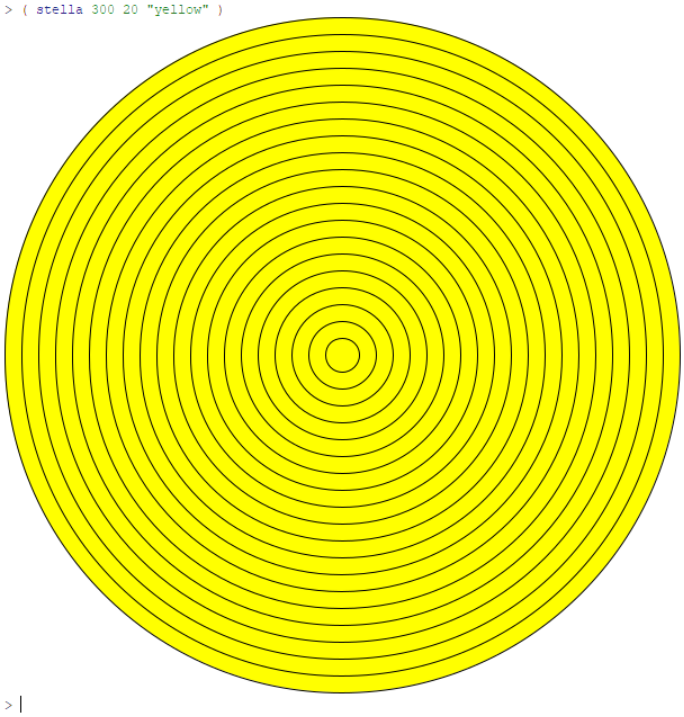
```
> ( stella 200 5 "red" )
```



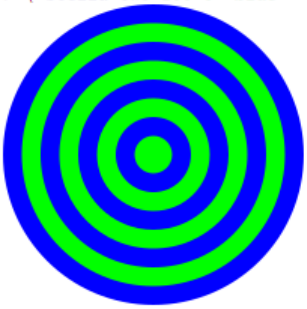
```
> ( stella-two 200 14 ( random-color ) ( random-color ) )
```



```
> ( stella 300 20 "yellow" )
```



```
> ( stella-two 100 8 "blue" "green" )
```



```
>
```

```
> |
```

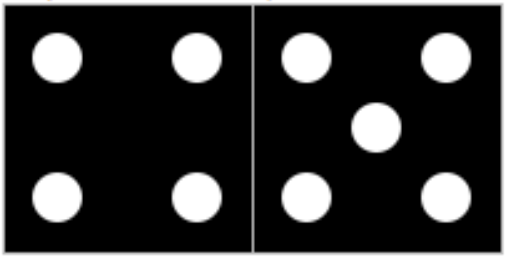
# Task 5: Code

```
#lang racket

; -----------------------------------------------------------------------
; Program that displays graphical images in the spirit of Frank Stella.
;
; Commands
; ( stella side count color )
; ( stella-two side count color1 color2 )
;
; Examples
; ( stella 100 5 "cyan" )
; ( stella 750 3 ( random-color ) )
; ( stella-two 500 14 "cyan" "green" )
; ( stella-two 500 14 ( random-color ) ( random-color ) )
;

; -----------------------------------------------------------------------
; Import the image library from htdp, 2nd edition
;
( require 2htdp/image )

; -----------------------------------------------------------------------
; Monochromatic Stella
;
( define ( stella side count color )
  ( define unit ( / side count ) )
  ( paint-stella 1 count unit color )
)

( define ( paint-stella from to unit color)
  ( define side-length ( * from unit ) )
  ( cond
    ( ( = from to )
      ( framed-shape side-length color )
    )
    ( ( < from to )
      ( overlay
        ( framed-shape side-length color )
        ( paint-stella ( + from 1 ) to unit color )
      )
    )
  )
)

( define ( framed-shape side-length color )
  ( overlay
     ( circle side-length "outline" "black" )
     ( circle side-length "solid" color )
  )
)

; -----------------------------------------------------------------------
; Two-Tone Stella
;
( define ( stella-two side count color1 color2 )
  ( define delta ( / side count ) )
  ( paint-stella-two 1 count delta color1 color2 )
)

( define ( paint-stella-two from to delta color1 color2 )
  ( define side-length ( * from delta ) )
  ( cond
```

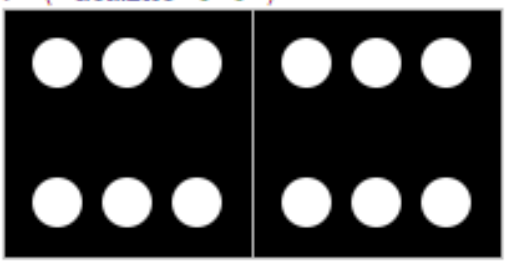
```
    ( ( = from to )
      ( if ( even? from )
          ( circle side-length "solid" color1 )
          ( circle side-length "solid" color2 )
      )
    )
    ( ( < from to )
      ( if ( even? from )
          ( overlay
            ( circle side-length "solid" color1 )
            ( paint-stella-two ( + from 1 ) to delta color1 color2 )
          )
          ( overlay
            ( circle side-length "solid" color2 )
            ( paint-stella-two ( + from 1 ) to delta color1 color2 )
          )

      )
    )
  )
)

; ---------------------------------------------------------------------
; Random Color Functions
;
( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )

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

## Task 6: Dominos

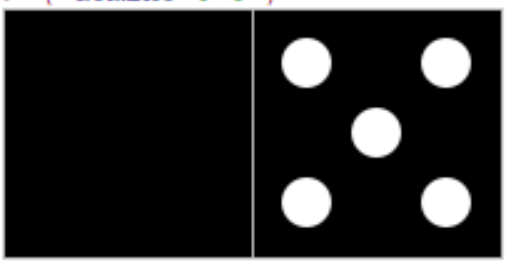
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( domino 4 5 )
```



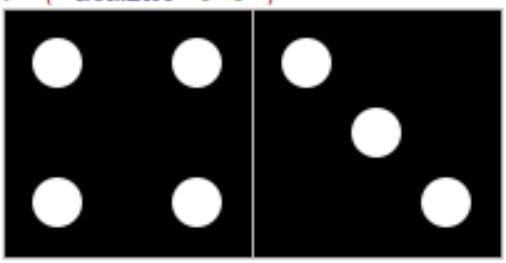
```
> ( domino 6 6 )
```



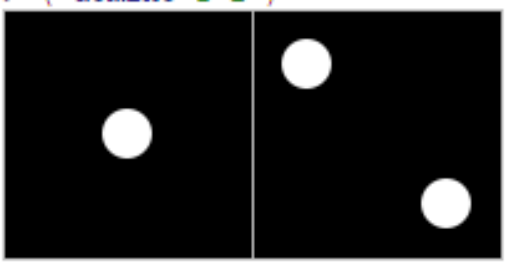
```
> ( domino 0 5 )
```



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



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



```
>
```

# Task 6: Code

```
#lang racket

;-----------------------------------------------------------------------------
; Program that renders dominos with values between 0 - 6 (inclusive)
;
; Command
; ( domino side1value side2value )
;
; Examples
; ( domino 6 5 )
; ( domino 3 1 )
; ( domino 2 0 )
;

;-----------------------------------------------------------------------------
; Import 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
;
( define blank-tile ( square side-of-tile "solid" "black" ) )

( define ( pip ) ( circle radius-of-pip "solid" "white" ) )

;-----------------------------------------------------------------------------
; 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 ) 0 d
    ( overlay/offset ( pip ) 0 nd basic-tile4 )
  )
)
;-------------------------------------------------------------------------------
; The framed 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
;
( 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

```
> ( my-creation )
```

```
>
```

# Task 7: Code

```
#lang racket

; -----------------------------------------------------------------------
; Program that displays a graphical image with the combined use
; of functions, both recursive and non-recursive, and features within
; the 2htdp/image library.
;
; Commands
; ( my-creation )
; ( my-obnoxious-creation )
;

; -----------------------------------------------------------------------
; Import the image library from htdp, 2nd edition
;
( require 2htdp/image )

; -----------------------------------------------------------------------
; My Creation
;
( define ( my-creation )
  ( overlay
    (add-solid-curve (ellipse 100 100 "solid" "black")
                     20 20 0 1
                     80 80 0 1
                     "white")
    ( rotate 310 ( swoosh ( circle 220 "outline" "black" ) 99 ) )
    ( circle 100 "solid" "white" )
    ( rotate 45 ( stella 600 ( + ( random 3 ) 1 ) ( random-color ) ) )
    ( rotate 45 ( stella-two 700 19 ( random-color ) ( random-color ) ) )
  )
)

; -----------------------------------------------------------------------
; My Obnoxious Creation
;
( define ( my-obnoxious-creation )
  ( overlay
    (add-solid-curve (ellipse 100 100 "solid" "black")
                     20 20 0 1
                     80 80 0 1
                     "white")
    ( rotate 180 ( swoosh ( circle 220 "outline" "black" ) 99 ) )
    ( circle 100 "solid" "white" )
    ( rotate 45 ( stella 600 ( + ( random 3 ) 1 ) ( random-color ) ) )
    ( stella-two 700 19 ( random-color ) ( random-color ) )
    ( stella 1000 ( + ( random 3 ) 1 ) ( random-color ) )
    ( koch-snowflake )
    ( rotate 45 ( stella 900 ( + ( random 3 ) 1 ) ( random-color ) ) )
    ( rotate 45 ( stella 1200 ( + ( random 3 ) 1 ) ( random-color ) ) )
    ( rotate 23 ( stella 1200 ( + ( random 3 ) 1 ) ( random-color ) ) )
    ( rotate 68 ( stella 1200 ( + ( random 3 ) 1 ) ( random-color ) ) )
    ( stella 1400 ( + ( random 3 ) 1 ) ( random-color ) )
  )
)

; -----------------------------------------------------------------------
; Monochromatic Stella
;
( define ( stella side count color )
  ( define unit ( / side count ) )
  ( paint-stella 1 count unit color )
```

```
)

( define ( paint-stella from to unit color)
  ( define side-length ( * from unit ) )
  ( cond
    ( ( = from to )
     ( framed-shape side-length color )
    )
    ( ( < from to )
     ( overlay
      ( framed-shape side-length color )
      ( paint-stella ( + from 1 ) to unit color )
     )
    )
  )
)

( define ( framed-shape side-length color )
  ( overlay
    ( spin-alot ( triangle side-length "outline" "black" ) )
    ( spin-alot ( triangle side-length "solid" color ) )
  )
)

; ---------------------------------------------------------------------
; Two-Tone Stella
;
( define ( stella-two side count color1 color2 )
  ( define delta ( / side count ) )
  ( paint-stella-two 1 count delta color1 color2 )
)

( define ( paint-stella-two from to delta color1 color2 )
  ( define side-length ( * from delta ) )
  ( cond
    ( ( = from to )
     ( if ( even? from )
        ( spin-alot ( triangle side-length "solid" color1 ) )
        ( spin-alot ( triangle side-length "solid" color2 ) )
     )
    )
    ( ( < from to )
     ( if ( even? from )
        ( overlay
          ( spin-alot ( triangle side-length "solid" color1 ) )
          ( paint-stella-two ( + from 1 ) to delta color1 color2 )
        )
        ( overlay
          ( spin-alot ( triangle side-length "solid" color2 ) )
          ( paint-stella-two ( + from 1 ) to delta color1 color2 )
        )

      )
    )
  )
)

; ---------------------------------------------------------------------
; Koch Curve Function
;
(define (koch-curve n)
   (cond
    [ ( zero? n ) ( overlay ( square 3 "outline" "black" ) ( square 3 "solid" "green") ) ]
    [ else
     ( local [ ( define smaller ( koch-curve ( - n 1 ) ) ) ]
       ( beside/align "bottom"
```

```
                smaller
                ( rotate 60 smaller )
                ( rotate -60 smaller )
                smaller )
        )
    ]
  )
)

; ---------------------------------------------------------------------
; Koch Snowflake Function
;
( define ( koch-snowflake )
  ( above
    ( beside
      ( rotate 60 ( koch-curve 5 ) )
      ( rotate -60 ( koch-curve 5 ) )
    )
    ( flip-vertical ( koch-curve 5 ) )
  )
)

; ---------------------------------------------------------------------
; Swoosh Function
;
( define ( swoosh image s )
   ( cond
    [ ( zero? s ) image ]
    [ else ( swoosh
           ( overlay/align "center" "top"
             ( circle ( * s 1/2 ) "solid" "white" )
             ( rotate 4 image ) )
           ( - s 1 )
      )
    ]
  )
)

; ---------------------------------------------------------------------
; Random Color Functions
;
( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )

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

; ---------------------------------------------------------------------
; Spin Alot Function
;
(define (spin-alot t)
   (local [ ( define ( spin-more i θ )
            ( cond
              [ ( = θ 360 ) i ]
              [ else
                ( spin-more ( overlay i ( rotate θ t ) )
                        ( + θ 1 ) )
              ]
             )
           )
        ]
        (spin-more t 0)
   )
)
```