Racket Programming Assignment #2: Racket Functions and Recursion

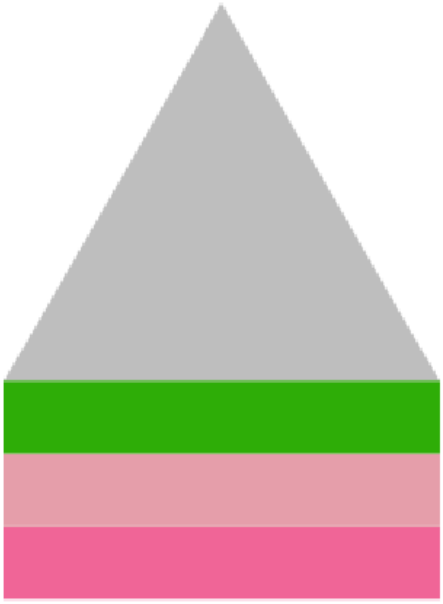
Learning Abstract

This assignment features multiple programs that use recursion. Many of the programs also use the 2htdp/image library in Racket.

Task 1: Colorful Permutations of Tract Houses

Demo for house:

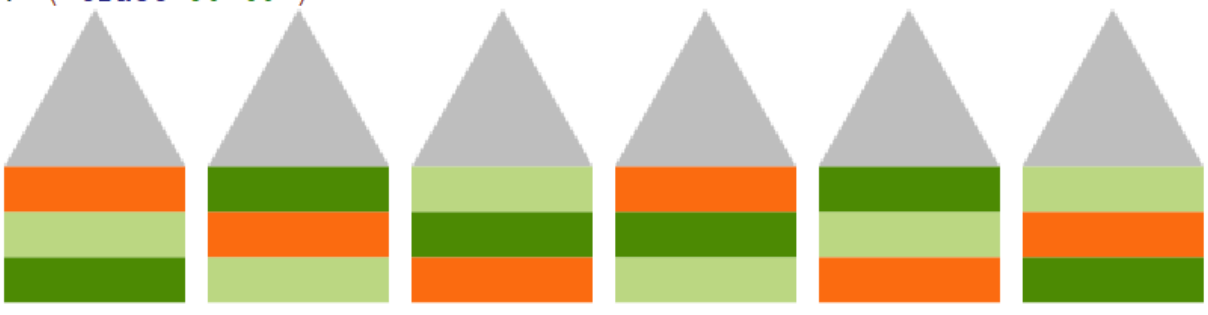
```
> ( house 200 100 ( random-color ) ( random-color ) ( random-color ) )
```



```
>
```

Demo for tract:

```
> ( tract 80 60 )
```



```
>
```

The code:

```
#lang racket
( require 2htdp/image)
( define ( random-color ) ( color ( random 256 ) ( random 256 ) ( random 256 ) ) )
( define ( floor-of-house width height color )
( rectangle width height "solid" color ) )

( define ( house width height color1 color2 color3 )
   ( define height-of-floor ( / height 3 ) )
   ( define floor1 ( floor-of-house width height-of-floor color1 ) )
   ( define floor2 ( floor-of-house width height-of-floor color2 ) )
   ( define floor3 ( floor-of-house width height-of-floor color3 ) )
   ( define roof ( triangle width "solid" "gray" ) )
   ( above roof floor3 floor2 floor1  )
   )

( define ( tract width height )
   ( define color1 ( random-color ) )
   ( define color2 ( random-color ) )
   ( define color3 ( random-color ) )
   ( define house1 ( house width height color1 color2 color3 ) )
   ( define house2 ( house width height color2 color3 color1 ) )
   ( define house3 ( house width height color3 color1 color2 ) )
   ( define house4 ( house width height color2 color1 color3 ) )
   ( define house5 ( house width height color3 color2 color1 ) )
   ( define house6 ( house width height color1 color3 color2 ) )
   ( define space ( square 10 "solid" "white" ) )
   ( beside house1 space house2 space house3 space house4 space house5 space house6 )
   )
```

Task 2: Dice

Demo:

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

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

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

Code:

```
#lang racket
( define ( roll-die )
( random 1 7 )
   )

( define ( roll-for-1 )
   ( define roll ( roll-die ) )
   ( display roll ) ( display " " )
   ( cond
      [ ( not ( eq? roll 1 ) ) ( roll-for-1 ) ]
      )
   )

( define ( roll-for-11 )
   (roll-for-1 )
   ( define roll ( roll-die ) )
   ( display roll ) ( display " " )
   ( cond
      [ ( not ( eq? roll 1 ) ) ( roll-for-11 ) ]
      )
   )

( define ( roll-odd )
   ( define roll ( roll-die ) )
   ( display roll ) ( display " " )
   ( cond
      [ ( not ( odd? roll ) ) ( roll-odd ) ]
      )
   )
```

```
( define ( roll-odd )
   ( define roll ( roll-die ) )
   ( display roll ) ( display " " )
   ( cond
      [ ( not ( odd? roll ) ) ( roll-odd ) ]
      )
   )

( define ( roll-even )
   ( define roll ( roll-die ) )
   ( display roll ) ( display " " )
   ( cond
      [ ( not ( even? roll ) ) ( roll-even ) ]
      )
   )

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

( define ( roll-two-dice-for-a-lucky-pair )
   ( define first-roll ( roll-die ) )
   ( define second-roll ( roll-die ) )
   ( display "(" ) ( display first-roll ) ( display " " ) ( display second-roll ) ( display ")" )
   ( cond
     [ ( eq? first-roll second-roll ) ( display "" ) ]
     [ ( eq? ( + first-roll second-roll ) 7 ) ( display "" ) ]
     [ ( eq? ( + first-roll second-roll ) 11 ) ( 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:

```
> ( 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:

```
> ( 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
( 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 " " )
        )
      )
   )

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

( 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

Demo:

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

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

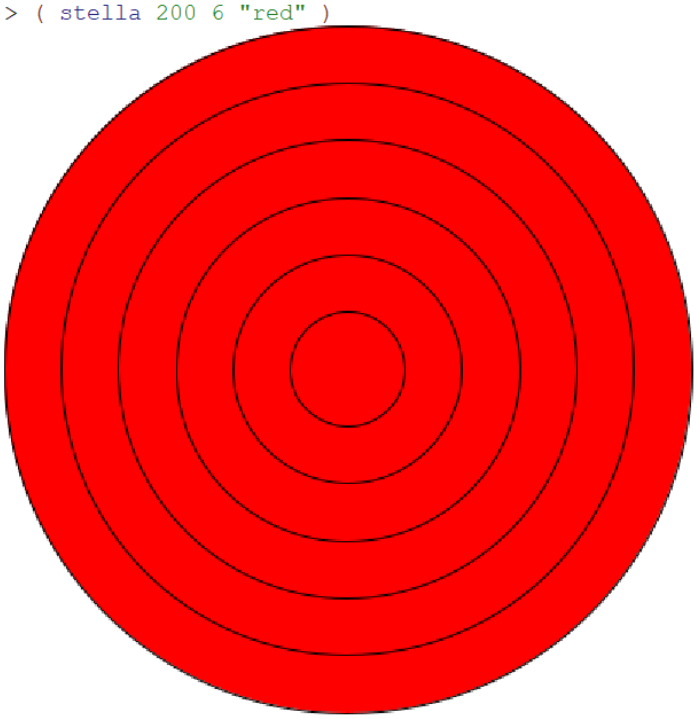
Code:

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

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

( define ( dot r )
   ( define d ( / r 2 ) )
   ( overlay
     ( circle d "solid" ( random-color ) )
     ( circle 20 "solid" "white" )
     )
   )

( define ( row n )
   ( cond
      ( ( = n 0 )
        empty-image
        )
      ( ( > n 0 )
        ( beside ( row ( - n 1 ) ) ( dot 30 ) )
        )
      )
   )

( define ( hirst-rect n c )
   ( cond
      ( ( = n 0 )
        empty-image
        )
      ( ( > n 0 )
        ( above
          ( hirst-rect ( - n 1 ) c ) ( row c ) )
        )
      )
   )

( define ( hirst-dots n )
   ( hirst-rect n n )
   )
```

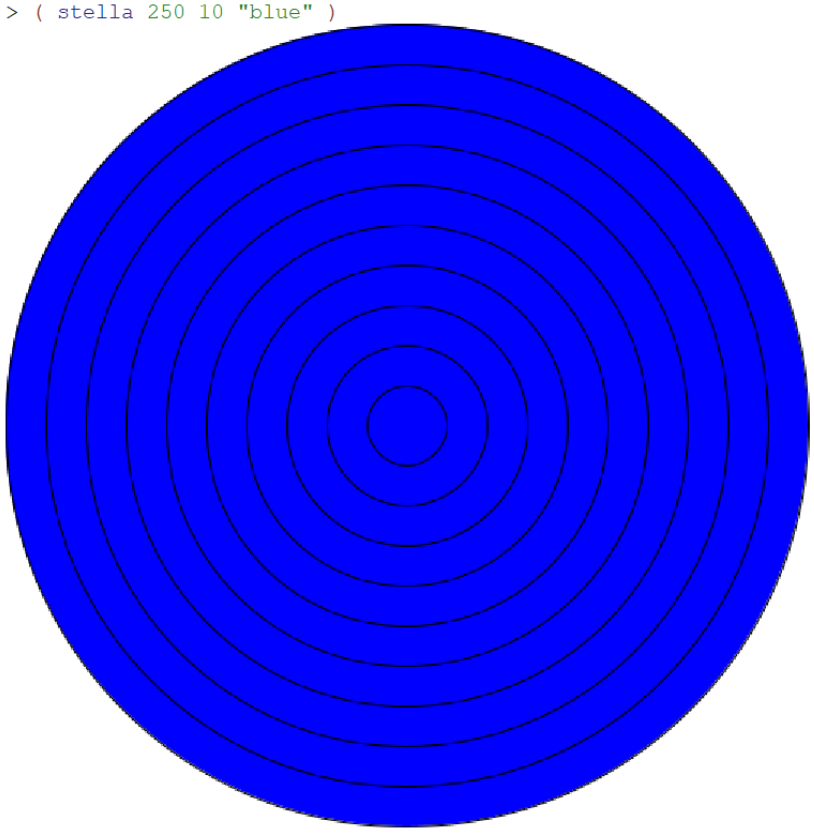
Task 5: Channeling Frank Stella

Demo:

```
> ( stella 200 6 "red" )
```



```
> ( stella 250 10 "blue" )
```



Code:

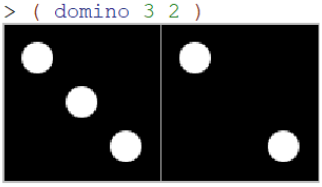
```
#lang racket
( require 2htdp/image )
( define ( stella side count color )
   ( define unit ( / side count ) )
   ( paint-stella 1 count unit color )
   )

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

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

Task 6: Dominos

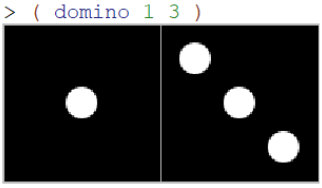
Demo:

```
> ( domino 0 1 )
```

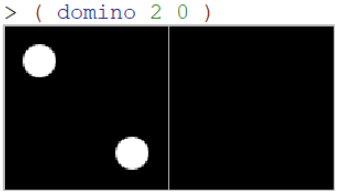
```
> ( domino 3 2 )
```



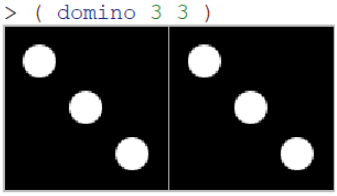
```
> ( domino 1 3 )
```



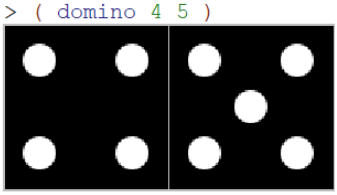
```
> ( domino 2 0 )
```



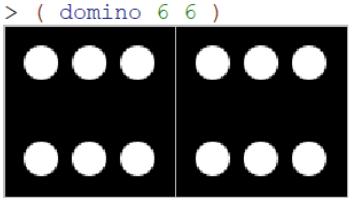
```
> ( domino 3 3 )
```



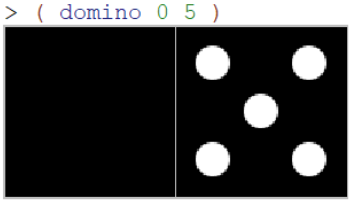
```
> ( domino 4 5 )
```



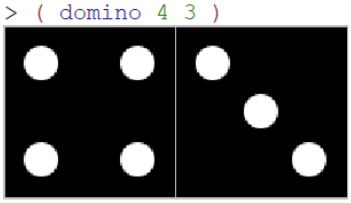
```
> ( domino 6 6 )
```



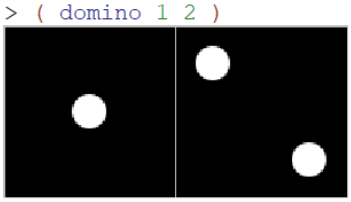
```
> ( domino 0 5 )
```



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



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



Code:

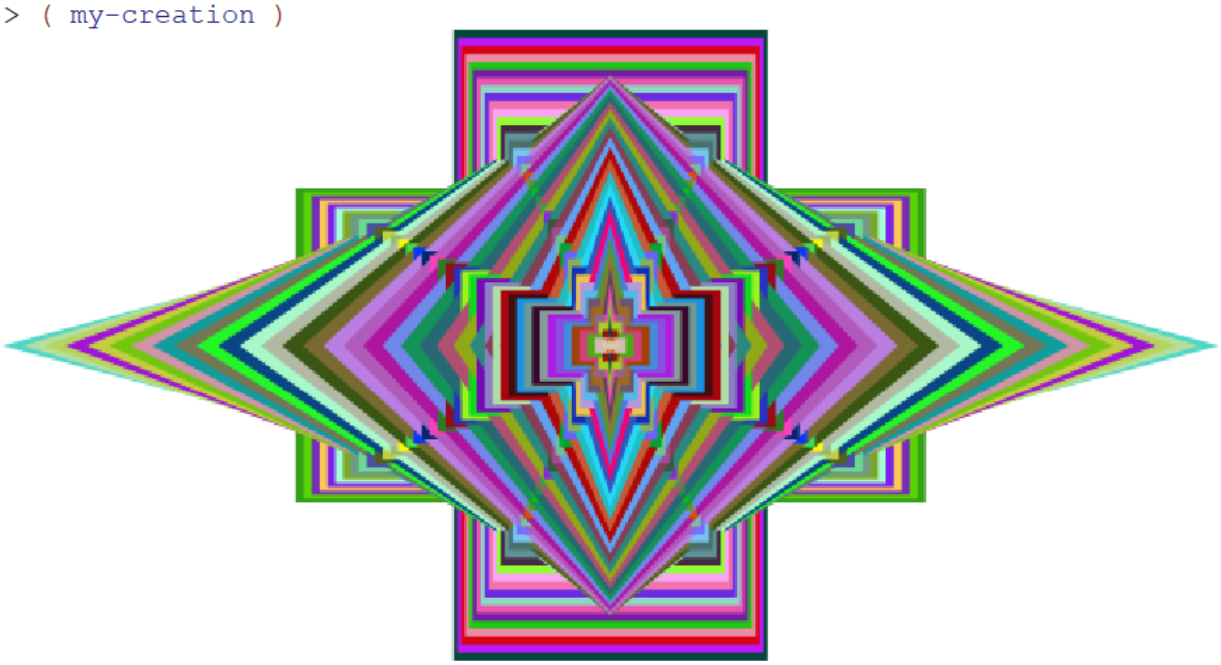
```
#lang racket
( require 2htdp/image )
;-------------------------------------------------------------------------------
; Problem parameters
;
; - Variables to denote the side of a tile and the dimensions of a pip
;
( define side-of-tile 100 )
( define diameter-of-pip ( * side-of-tile 0.2 ) )
( define 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 ) ( / 2 d ) nd
                    ( overlay/offset ( pip ) ( / 2 nd ) 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 )
      ( ( = x 1 ) tile1 )
      ( ( = x 2 ) tile2 )
      ( ( = x 3 ) tile3 )
      ( ( = x 4 ) tile4 )
      ( ( = x 5 ) tile5 )
      ( ( = x 6 ) tile6 )
     )
   )
```

Task 7: My Creation

Demo:

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



Code:

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

( define ( creation side count color )
   ( define unit ( / side count ) )
   ( paint-creation 1 count unit color )
   )

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


( define ( framed-circle side-length color )
   ( overlay
     ( rhombus side-length ( / side-length 2 ) "solid" ( random-color ) )
     ( rhombus ( / side-length 2 ) side-length "solid" ( random-color ) )
     ( rectangle side-length ( / side-length 2 ) "solid" ( random-color ) )
     ( rectangle ( / side-length 2 ) side-length "solid" ( random-color ) )
    )
   )

( define ( my-creation ) ( creation 300 40 ( random-color) ) )
```