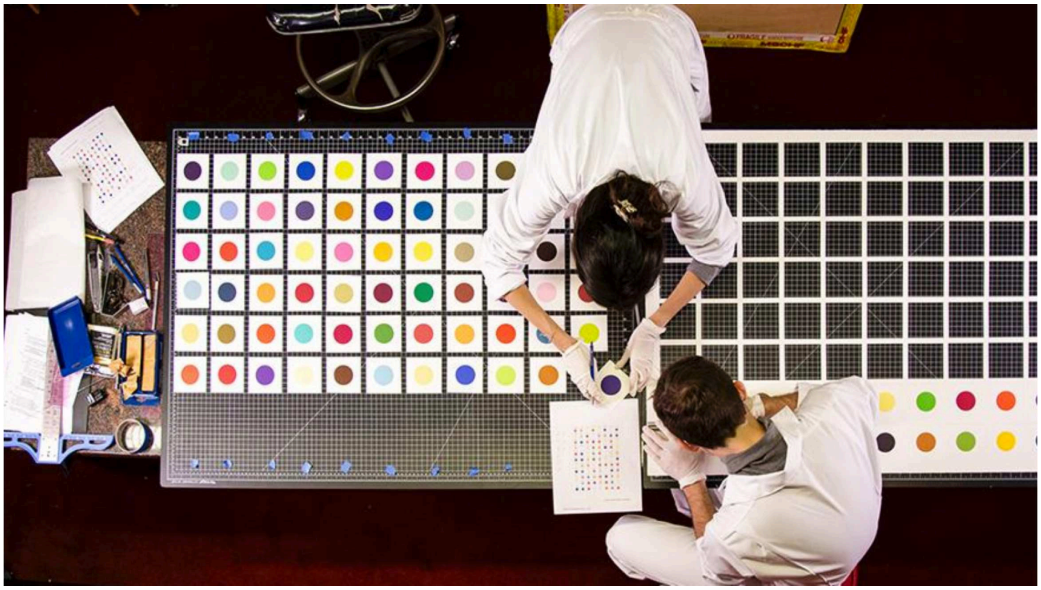
Racket Programming Assignment #2: Racket Functions and Recursion

By: Miguel Cruz



Abstract

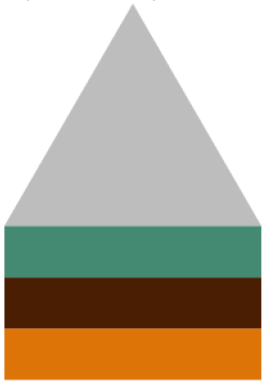
This assignment will concentrate on recursion and the use of Racket's 2htdp/image library. Recursion is a time-saving method of tackling multi-level issues without utilizing iterative programming. It's delegating the task to the function! To make the examples below, I used the racket 2htdp/image library, which allows me access to various different pre-built functions to generate shapes and other functionality.

Seven different tasks will be completed to practice recursive functions and image creation. Visual permutations in a set of three, rolling dice for specific outcomes using recursion, recursive number sequences, and recursive picture production will all be included.

Task 1: Colorful Permutations of Tract Houses

The `house` Demo

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

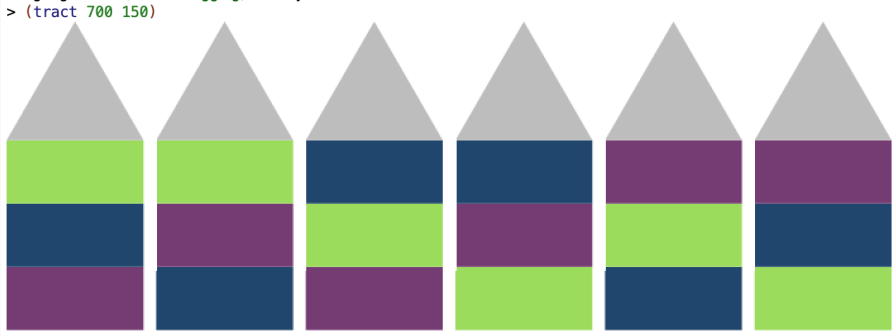


```
> (house 100 60 (random-color) (random-color) (random-color))
```

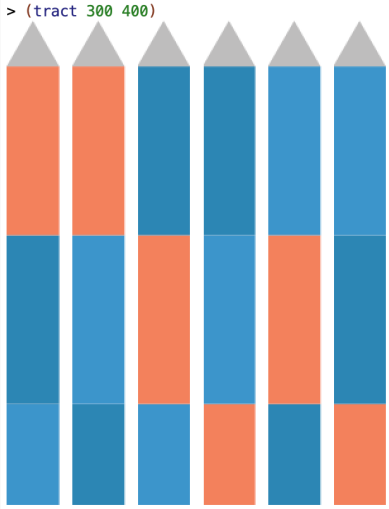
```
>
```

The `tract` Demo

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



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



```
;---------
; Programming Assignment 2: Racket Functions and Recursion

#lang racket
( require 2htdp/image )

;; Task 1: Colorful Permutations of Tract Houses (Done)

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

(define ( house-roof side color)
  (triangle side "solid" color)
 )

(define (house-body width height color)
  (rectangle width height "solid" color)
)


(define (house width height random-color1 random-color2 random-color3)
   (define roof (house-roof width (color 190 189 189 )))
   (define floor1 (house-body width height random-color1))
   (define floor2 (house-body width height random-color2))
   (define floor3 (house-body width height random-color3))
   (above roof floor1 floor2 floor3)
)

(define (tract width height)
  (define indiv-width (/ (- width 50) 6))
  (define indiv-height(/ height 3))
  (define rand-color1 (random-color))
  (define rand-color2 (random-color))
  (define rand-color3 (random-color))

  (define h1(house indiv-width indiv-height rand-color1 rand-color2 rand-color3))
  (define h2(house indiv-width indiv-height rand-color1 rand-color3 rand-color2))
  (define h3(house indiv-width indiv-height rand-color2 rand-color1 rand-color3))
  (define h4(house indiv-width indiv-height rand-color2 rand-color3 rand-color1))
  (define h5(house indiv-width indiv-height rand-color3 rand-color1 rand-color2))
  (define h6(house indiv-width indiv-height rand-color3 rand-color2 rand-color1))
  (define gap1 (rectangle 10 (* 3 indiv-height) "outline" "white"))
  (define gap2 (rectangle 10 (* 3 indiv-height) "outline" "white"))
  (define gap3 (rectangle 10 (* 3 indiv-height) "outline" "white"))
  (define gap4 (rectangle 10 (* 3 indiv-height) "outline" "white"))
  (define gap5 (rectangle 10 (* 3 indiv-height) "outline" "white"))


  (beside/align "center" h1 gap1 h2 gap2 h3 gap3 h4 gap4 h5 gap5 h6)
)
;------------------------
;; Testing Cases
;------------------------
;(house 100 60 (random-color) (random-color) (random-color))
;(tract 700 150)
;(tract 300 400)


```

Task 2: Dice

The Demo

```
> (roll-die)
5
> (roll-die)
4
> (roll-die)
3
> (roll-die)
4
> (roll-die)
2
> (roll-for-1)
5 5 1
> (roll-for-1)
1
> (roll-for-1)
3 1
> (roll-for-1)
3 5 2 4 1
> (roll-for-1)
5 3 1
> (roll-for-11)
3 5 5 3 4 13 2 5 3 3 3 3 5 12 2 2 14 2 3 3 3 3 13 5 4 3 5 2 2 3 5 3 15 2 5 2 12 4 5 5 14 4 4 4
4 12 3 2 2 5 3 2 2 3 14 2 4 4 4 14 4 3 14 3 3 2 3 4 12 11
> (roll-for-11)
5 13 14 5 5 5 5 15 2 2 4 12 4 11
> (roll-for-11)
15 13 4 13 5 4 4 15 5 4 3 4 3 3 3 12 2 2 3 2 4 5 2 3 5 3 5 5 3 5 3 4 14 3 5 14 3 2 2 5 4 5 2 2
5 2 5 12 11
> (roll-for-11)
2 2 2 4 4 13 4 2 2 5 2 11
> (roll-for-11)
3 14 5 3 13 5 2 13 2 2 4 4 15 2 3 2 2 2 4 2 4 4 2 4 4 4 4 5 2 2 5 5 3 2 4 3 4 15 5 4 4 2 3 4 4
5 2 4 2 3 11
> (roll-for-odd-even-odd)
1 1 3 1 3 2 1
> (roll-for-odd-even-odd)
4 2 4 1 5 4 1
> (roll-for-odd-even-odd)
3 1 2 1
> (roll-for-odd-even-odd)
1 3 1 5 4 2 3 1 2 1
> (roll-for-odd-even-odd)
1 1 4 5
> (roll-two-dice-for-a-lucky-pair)
(1 3)   (5 2)
> (roll-two-dice-for-a-lucky-pair)
 (2 2)
> (roll-two-dice-for-a-lucky-pair)
(1 5)  (3 2)  (1 2)   (3 3)
> (roll-two-dice-for-a-lucky-pair)
(4 5)  (4 5)  (3 2)  (3 2)  (4 1)   (2 5)
> (roll-two-dice-for-a-lucky-pair)
 (4 3)
> (roll-two-dice-for-a-lucky-pair)
(2 1)   (4 4)
> (roll-two-dice-for-a-lucky-pair)
```

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

```
;---------
; Programming Assignment 2: Racket Functions and Recursion

#lang racket
( require 2htdp/image )


;; Task 2: Dice (Done)

;; Roll Die
(define (roll-die)
  (random 1 6)
  )

;; Roll for 1
(define (roll-for-1)
  (define roll-store (roll-die))
  (cond
    ((= roll-store 1)
     (display roll-store )
     )
    ((not (= roll-store 1))
     (printf "~a " roll-store)
     (roll-for-1)
     )
    )
  )

;; Make a recursive function to roll for two consecutives 1s then stop
( define ( roll-for-11)
   (roll-for-1)
   ( define roll-store (roll-die))
   (cond
     ((= roll-store 1)
      ( display roll-store ) ( display " ")
      )
     (not (= roll-store 1)
          (display roll-store) ( display " ")
          (roll-for-11)
          )
     )
   )



;; Roll for an odd value
(define (roll-for-odd)
  (define roll-store ( roll-die))
  (cond
    ( (or (= roll-store 1) (= roll-store 3) (= roll-store 5)) ;if odd
      (display roll-store) (display " ")
      )
    ( (or (= roll-store 2)(= roll-store 4) (= roll-store 6 ));if even
      (display roll-store) (display " ")
      (roll-for-odd)
      )
    )
  )
```

```
59
60
61
62  ;; Roll for an even value
63  (define( roll-for-even)
64    (define roll-store ( roll-die))
65    (cond
66      ((or (= roll-store 2) (= roll-store 4) (= roll-store 6 )); if even
67       (display roll-store ) ( display " ")
68       )
69      ((or (= roll-store 1) (= roll-store 3) (= roll-store 5)) ;if odd
70       (display roll-store) (display " ")
71       (roll-for-even)
72       )
73      )
74    )
75
76  ;; Simulates rolls for consecutive odd-even-odd values
77  (define ( roll-for-odd-even-odd)
78    (roll-for-odd)
79    (roll-for-even)
80    (define roll-store (roll-die))
81    (cond
82      ;;ODD
83      ((or(= roll-store 1 ) (= roll-store 3) (= roll-store 5))
84      (display roll-store) (display " ")
85      )
86    ;;EVEN
87    ((or (= roll-store 2) (= roll-store 4) (= roll-store 6))
88    (display roll-store) (display " ")
89    (roll-for-odd-even-odd)
90     )
91    )
92  )
93
94
95
96  ; roll 2 dice until sum of 7 or 11 or a double turns up
97  ( define ( roll-two-dice-for-a-lucky-pair)
98     (define roll-store-one ( roll-die))
99     (define roll-store-two (roll-die))
100    (define sum (+ roll-store-one roll-store-two))
101    (cond
102      ((or (= 7 sum) ( = 11 sum) (= roll-store-one roll-store-two))
103      ( display " (" ) (display roll-store-one) (display " ")
104      (display roll-store-two) (display ") ") (display " ")
105      )
106    (else
107     (display "(") ( display roll-store-one) (display " ")
108     (display roll-store-two) (display ") ") (display " ")
109     (roll-two-dice-for-a-lucky-pair)
110     )
111    )
112  )
113
114
115
116
```

```
117
118 ;-----------------------
119 ;; Testing Cases
120 ;-----------------------
121 ;(roll-die)
122 ;(roll-for-1)
123 ;(roll-for-11)
124 ;(roll-for-odd-even-odd)
125 ;(roll-two-dice-for-a-lucky-pair)
126
127
128
129
130
131
132
133
```

Task 3: Number Sequences

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 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
```

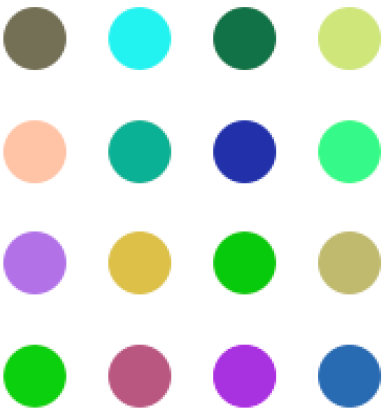
```
;---------
; Programming Assignment 2: Racket Functions and Recursion

#lang racket
( require 2htdp/image )
;; Task 3: Number Sequences (Done)

( 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 " " )
        )
      )
)

;; write a function called triangular taking one positive integer as its sole parameter
which returns the triangular number corresponding to the given value
(define (triangular n)
   (cond
      ( ( = n 1 )
        1
)
      ( else
        ( + (triangular ( - n 1 )) n )
        )
      )
)


(define sigma
  (lambda (n)
    ((lambda (y) (y y n (lambda (s) s)))
     (lambda (s i ret)
       (if (zero? i)
           (ret 0)
           (s s (- i 1)
              (if (zero? (remainder n i))
                  (lambda (x)
                    (ret (+ i x)))
                  ret)
              )
           )
       )
     )
    )
  )
```

Task 4: Hirst Dots

The Demo

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

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



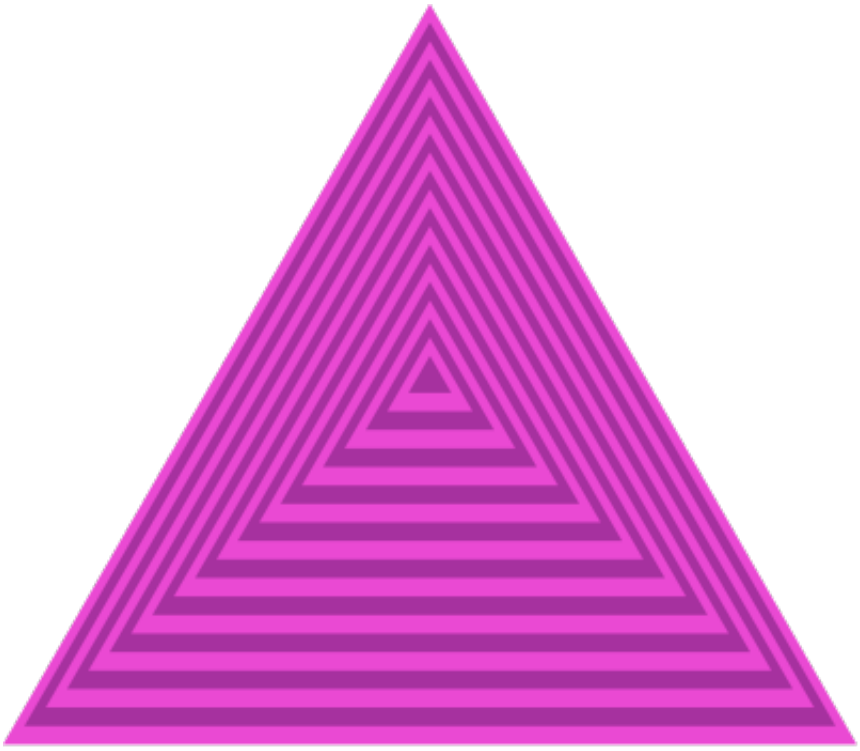
```
>
```

```
;---------
; Programming Assignment 2: Racket Functions and Recursion

#lang racket
( require 2htdp/image )
;; Task 4: Hirst Dots (Done)

;; Make Random Colors
( define ( rgb-value ) ( random 256 ) )
( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )

;; Make Dot
(define (make-dot)
  (circle 15 "solid" (random-color))
)

;; Make Row of Dots
(define (make-dot-row n)
  (cond
     ((= n 0)
       (display "\n"))
     ((> n 0)

       (display (above (beside (make-dot) (square 20 "outline" "white")) (square 20
"outline" "white")))
       (make-dot-row (- n 1))
       )
     )
)
;; Make a rectangle generator for hirst dots
(define (rectangle-of-hirst-dots r c)
  (cond
    ((= r 0)
     (display "\n")
     )
     ((> r 0)
      (make-dot-row c)
      (rectangle-of-hirst-dots (- r 1) c)
    )
  )
)

;; A square is a rectangle therefore side
(define (hirst-dots n)
(rectangle-of-hirst-dots n n)
)


;;(hirst-dots 10)
```

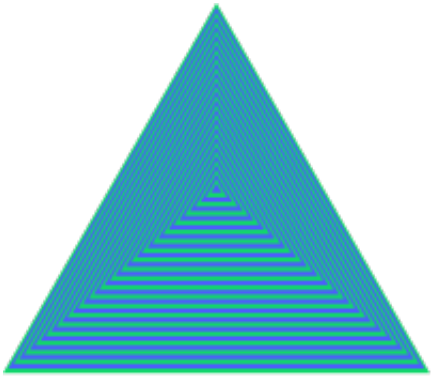
Task 5: Chanelling Frank Stella

The Demo

```
> (stella 400 20 (random-color) (random-color))
```



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

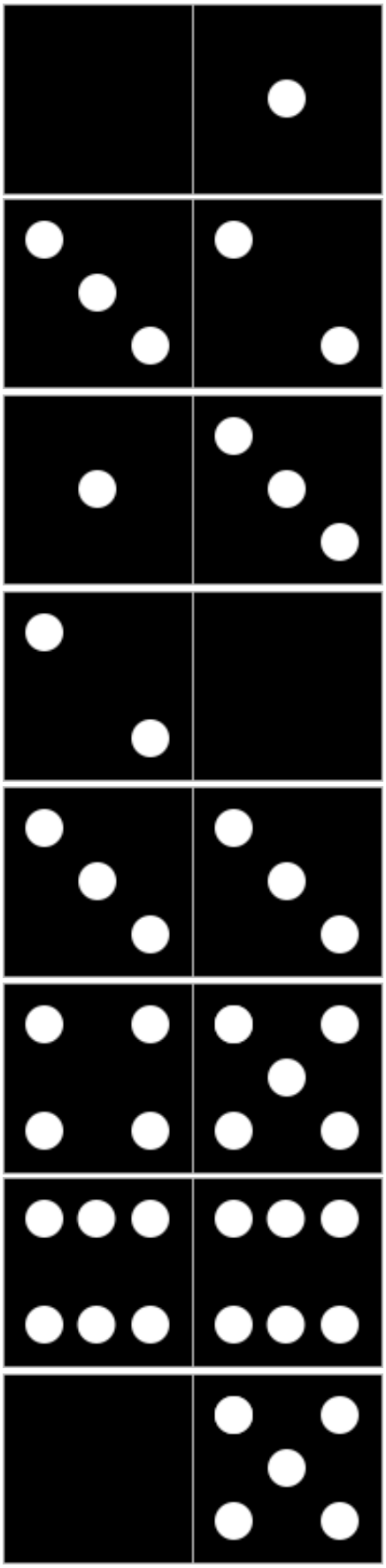


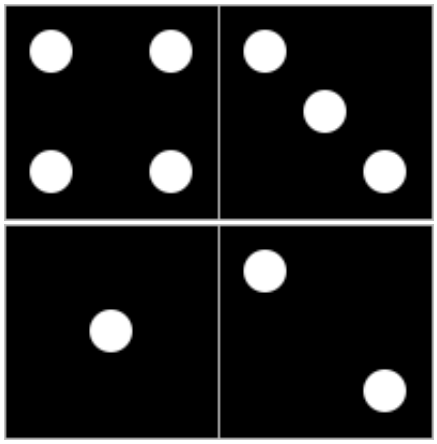
```
>
```

```
;---------
; Programming Assignment 2: Racket Functions and Recursion

#lang racket
( require 2htdp/image )
;; Task 5: Channeling Frank Stella (Done)

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

( define (stella side count color1 color2 )
   ( define delta ( / side count ) )
   ( paint-nested-triangles 1 count delta color1 color2 )
)

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

;; Testing
;;(stella 400 20 (random-color) (random-color))
```

Task 6: Dominos

Demo for 0, 1, 2, 3, 4, 5, 6 pip Dominos





>

```
;---------
; Programming Assignment 2: Racket Functions and Recursion

#lang racket
( require 2htdp/image )
;; Task 6: Dominos (Done)

;-------------------------------------------------------------------------------
; Requirements
;
; - Just the image library from Version 2 of "How to Design Programs"
;
;-------------------------------------------------------------------------------
; 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)
   )
)


```

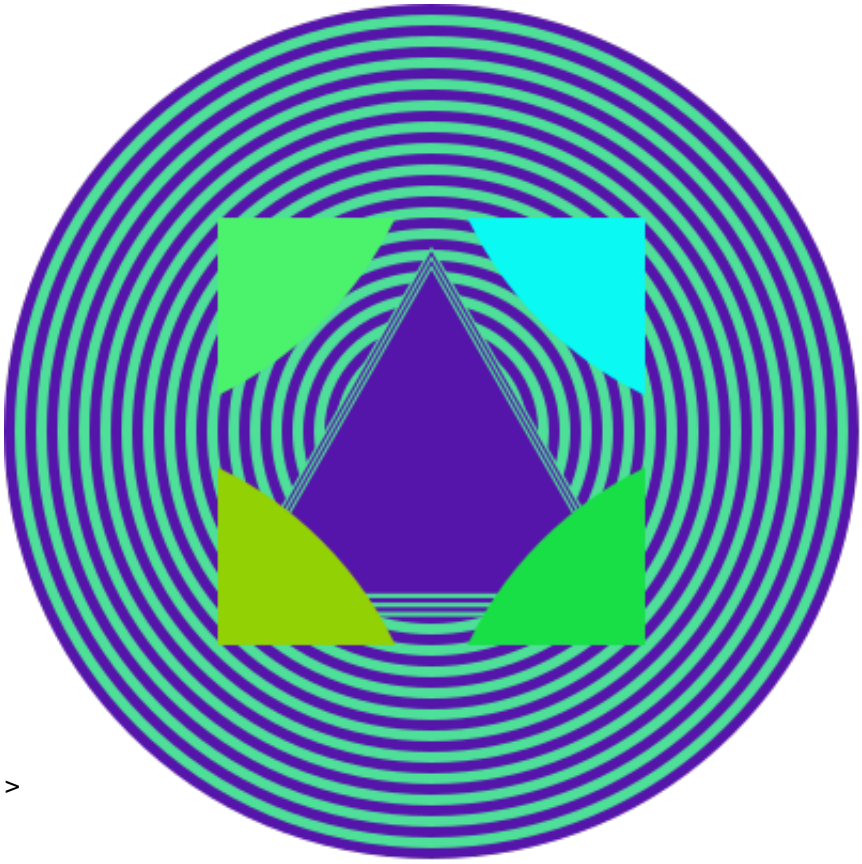
```
( define basic-tile3
   ( overlay ( pip ) basic-tile2 )
)


( define basic-tile4
   ( overlay/offset ( pip ) d d
    ( overlay/offset ( pip ) nd d
     ( overlay/offset ( pip ) nd nd
      ( overlay/offset ( pip ) d nd blank-tile)))
    )
)

( define basic-tile5
   (overlay (pip)
      ( overlay/offset ( pip ) d d basic-tile4 )
  )
)


( define basic-tile6
   ( overlay/offset ( pip ) nd d
    ( overlay/offset ( pip ) 0.5 d
     ( overlay/offset ( pip ) d d
      ( overlay/offset ( pip ) 0.5 nd
       ( overlay/offset ( pip ) d nd
        ( overlay/offset ( pip ) nd 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 )
      )
   )

;-----------------------
;; Testing Cases
;-----------------------
(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)
```

Task 7: Creation

Demo

```
> (some-circle-thing 200 40 (random-color) (random-
  color))
```



>

```
;---------
; Programming Assignment 2: Racket Functions and Recursion

#lang racket
( require 2htdp/image )


;; Task 7 Random Image (Done)

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


(define (coolthing size)
  (above
   (beside
    (crop/align "right" "bottom" 100 100 (circle size "solid" (random-color)))
    (crop/align "left" "bottom" 100 100 (circle size "solid"  (random-color))))
   (beside
    (crop/align "right" "top" 100 100 (circle size "solid"    (random-color)))
    (crop/align "left" "top" 100 100 (circle size "solid"     (random-color))))))


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


;; Make a paint nested cropped circle function
( define ( paint-nested-circle 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-circle ( + from 1 ) to delta color1 color2 )
              )
            ( overlay
              ( circle side-length "solid" color2 )
              ( paint-nested-circle ( + from 1 ) to delta color1 color2 )
              )
            )
        )
      )
)

( define (some-circle-thing side count color1 color2 )
   ( define delta ( / side count ) )
   (overlay
    (coolthing 175)
    ( paint-nested-triangles 35 count delta color2 color1 )
    ( paint-nested-circle 1 count delta color1 color2 ))

)


;(some-circle-thing 200 40 (random-color) (random-color))
```