Racket Programming Assignment #2: Racket Functions and Recursion

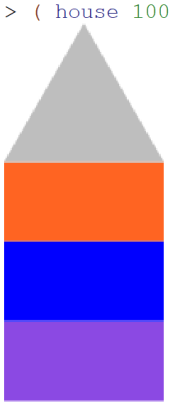
Learning Abstract

This assignment features programs that generate images in the context of the 2htdp/image library, most of which are recursive in nature.

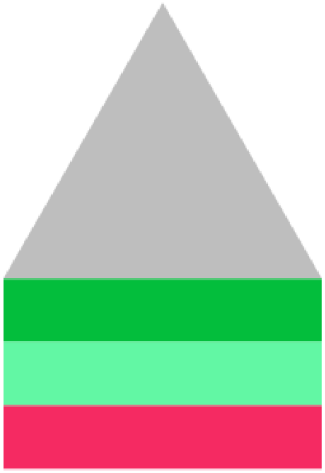
Task 1: Colorful Permutations of Tract Houses

Demo for house

```
> ( house 100 50 (random-color) 'blue (color 255 100 34) )
```

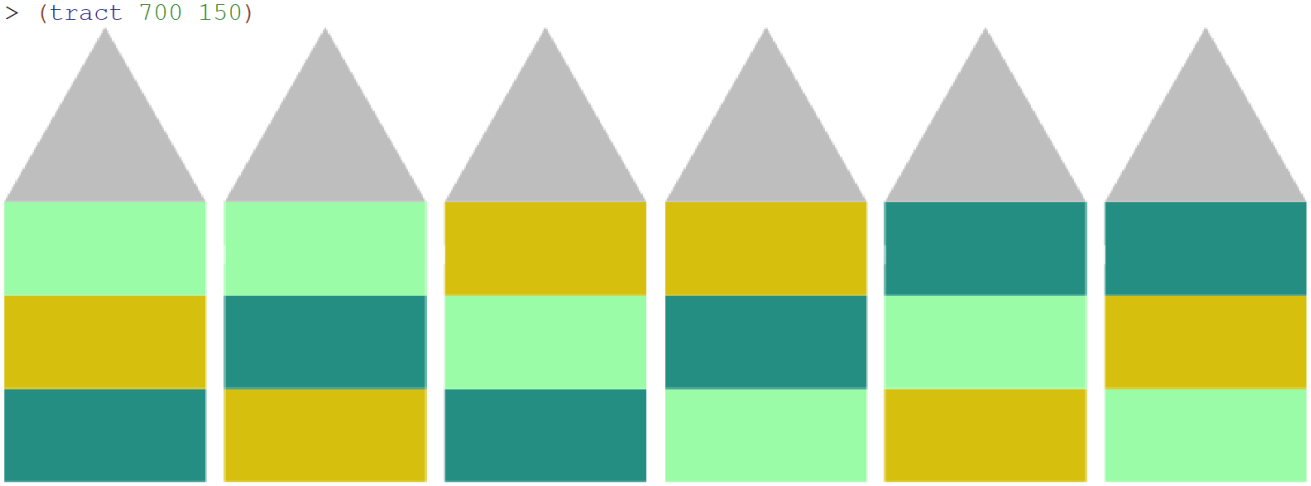


```
> ( house 200 40 (random-color) (random-color) (random-color))
```

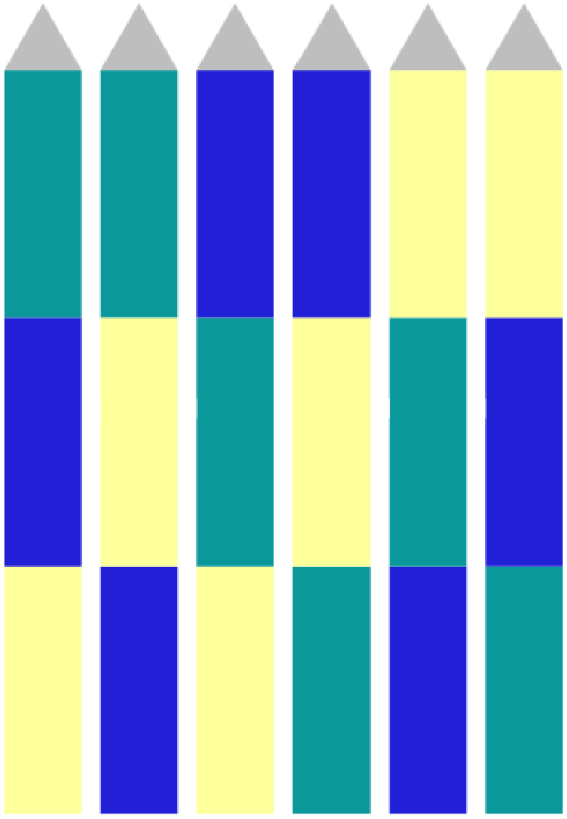


Demo for tract

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



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



The code ...

★ 1: house.rkt × | 2: tract.rkt × +

```
#lang racket
(require 2htdp/image)
( define (random-color) ( color ( rgb-value) (rgb-value) (rgb-value) ) )
( define (rgb-value) ( random 256) )
( define (house width height color1 color2 color3)
   (define floor1
     ( rectangle width height 'solid color1))
   (define floor2
     ( rectangle width height 'solid color2))
   (define floor3
     ( rectangle width height 'solid color3))
   (define roof
     ( triangle width 'solid 'gray ))

 (display (above roof floor3 floor2 floor1)) (display "\n")
)
```

★ 1: house.rkt × | 2: tract.rkt × +

```
#lang racket
(require 2htdp/image)
( define (random-color) ( color ( rgb-value) (rgb-value) (rgb-value) ) )
( define (rgb-value) ( random 256) )
( define (tract width height )
   ( define gap 10 )
   ( define floor-height ( / height 3 ) )
   ( define floor-width ( / ( - width ( * 5 gap) ) 6 ) )
   (define floor1
     ( rectangle floor-width floor-height 'solid (random-color)))
   (define floor2
     ( rectangle floor-width floor-height 'solid (random-color)))
   (define floor3
     ( rectangle floor-width floor-height 'solid (random-color)))
   (define roof
     ( triangle floor-width 'solid 'gray ))
   (define perm1 (above roof floor3 floor2 floor1))
   (define perm2 (above roof floor3 floor1 floor2))
   (define perm3 (above roof floor2 floor3 floor1))
   (define perm4 (above roof floor2 floor1 floor3))
   (define perm5 (above roof floor1 floor3 floor2))
   (define perm6 (above roof floor1 floor2 floor3))
   (define spacer ( square gap 'solid 'white))

   ( display ( beside perm1 spacer perm2 spacer perm3 spacer perm4 spacer perm5 spacer perm6 ) )
)
```

Task 2: Dice

Demo ...

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

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

Code ...

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

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

```
36
37 ( define ( roll-for-odd-even-odd )
38    ( roll-for-odd )
39    ( define ( roll-for-odd-even )
40       ( define outcome (roll-die) )
41       ( display outcome ) ( display " " )
42       ( cond
43          [(odd? outcome)( roll-for-odd-even )]
44       )
45    )
46    ( roll-for-odd-even )
47    ( define outcome (roll-die) )
48       ( display outcome ) ( display " " )
49       ( cond
50          [(even? outcome)( roll-for-odd-even-odd )]
51       )
52 )
53
54 ( define ( roll-two-dice-for-a-lucky-pair )
55    ( define roll1 (roll-die) ) ( define roll2 (roll-die) )
56    ( display "(" )( display roll1 ) ( display " " ) ( display roll2 )( display ") " )
57    ( define sum ( + roll1 roll2) )
58    ( cond
59       [( = roll1 roll2)( display "\n" )]
60       [( = sum 7 )( display "\n" )]
61       [( = sum 11 )( display "\n" )]
62       [ else ( roll-two-dice-for-a-lucky-pair )]
63    )
64 )
```

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
>
```

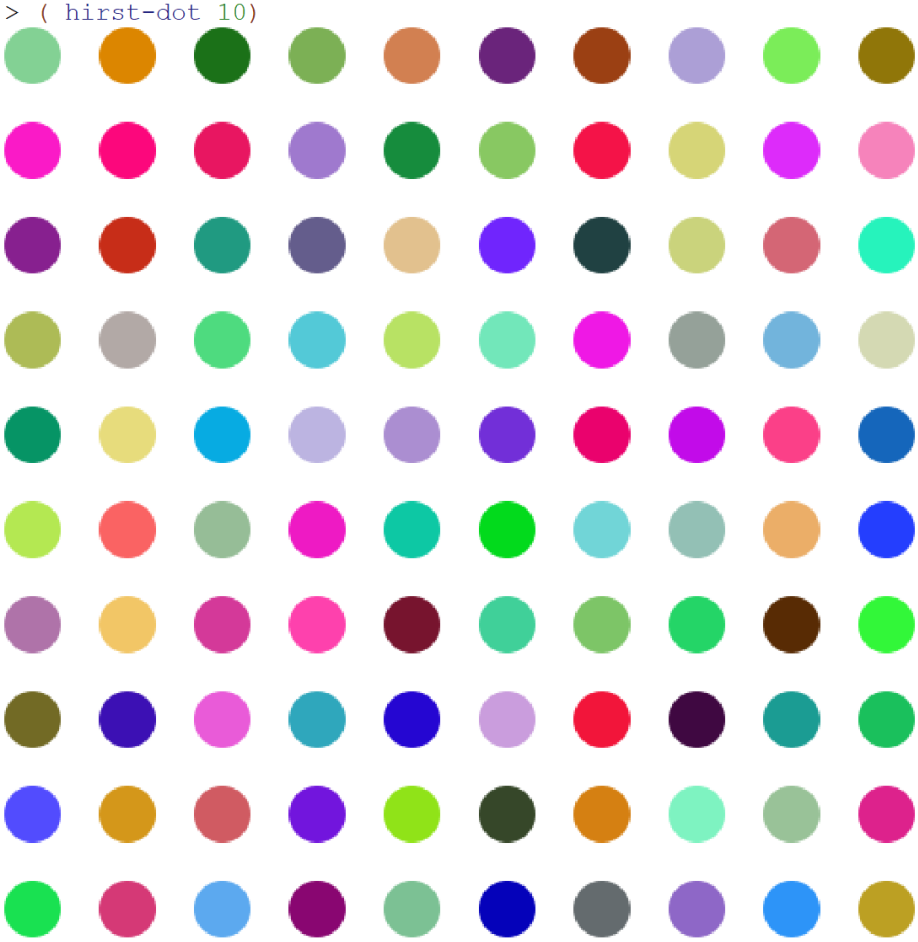
Code ...

```
#lang racket
( define ( square n )
   ( * n n )
)
( define ( cube n )
   ( * n n n )
)
( define ( triangular n )
   ( cond
      [( = n 1 ) 1 ]
      [( > n 1 ) ( + n ( triangular( - n 1) ))]
   )
)
( define ( sigma n )
   ( define ( sum-of-div orig check )
      ( cond
         [( = check 1 ) 1]
         [( = (remainder orig check) 0) ( + check ( sum-of-div orig ( - check 1 )))]
         [else (sum-of-div orig ( - check 1 ))]
      )
   )
   (sum-of-div n n)
)
( define ( sequence name n )
   ( cond
      ( ( = n 1 )
       ( display ( name 1 ) ) ( display " " )
      )
      ( else
        ( sequence name ( - n 1 ) )
        ( display ( name n ) ) ( display " " )
      )
   )
)
```

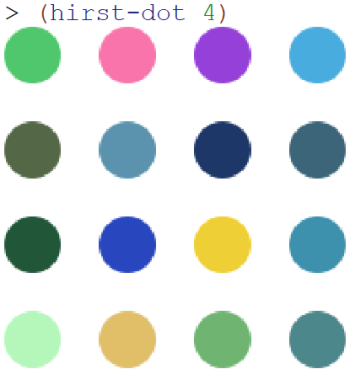
Task 4: Hirst Dots

Demo ...

```
> ( hirst-dot 10)
```



```
> (hirst-dot 4)
```



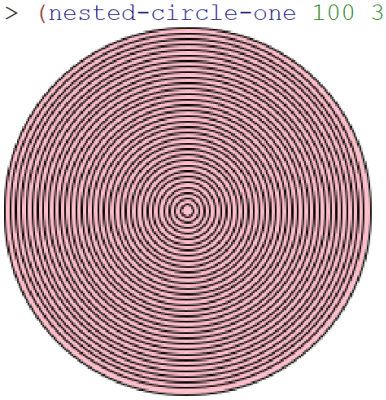
Code ...

```
#lang racket
(require 2htdp/image)
( define (random-color) ( color ( rgb-value) (rgb-value) (rgb-value) ) )
( define (rgb-value) ( random 256) )
( define gap ( circle 10 "solid" "white" ) )
( define ( hirst-dot size )
   ( define ( draw-row lenght )
      ( define dot ( circle 15 "solid" ( random-color ) ) )
      ( cond
         [( = lenght 0 ) empty-image ]
         [( > lenght 1 ) ( beside dot gap ( draw-row ( - lenght 1 ) ) )]
         [( = lenght 1 ) ( beside dot ( draw-row ( - lenght 1 ) ) )]
      )
   )
   ( define ( draw-colm height width )
      ( cond
         [( = height 0 ) empty-image ]
         [( > height 0 ) ( above ( draw-row width ) gap (draw-colm ( - height 1 ) width ) )]
      )
   )
   ( draw-colm size size )
)
```

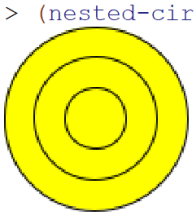
Task 5: Chanelling Frank Stella

Demo ...

```
> (nested-circle-one 100 33 "pink")
```



```
> (nested-circle-one 50 3 "yellow")
```



```
>
```

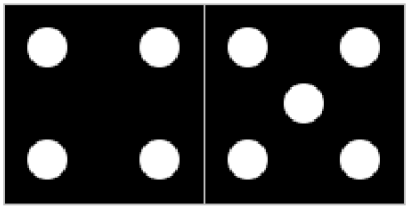
Code ...

```
#lang racket
( require 2htdp/image )
( define ( nested-circle-one rad count color )
   ( define unit ( / rad count ) )
   ( paint-nested-circle-one 1 count unit color )
)
( define ( paint-nested-circle-one from to unit color)
   ( define rad-length ( * from unit ) )
   ( cond
      ( ( = from to )
        ( framed-circle rad-length color )
      )
      ( ( < from to )
        ( overlay
          ( framed-circle rad-length color )
          ( paint-nested-circle-one ( + from 1 ) to unit color )
        )
      )
   )
)
( define ( framed-circle rad-length color )
   ( overlay
     ( circle rad-length "outline" "black" )
     ( circle rad-length "solid" color )
   )
)
```

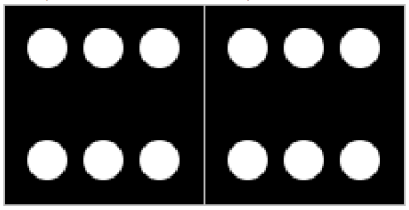
Task 6: Dominos

Final demo ...

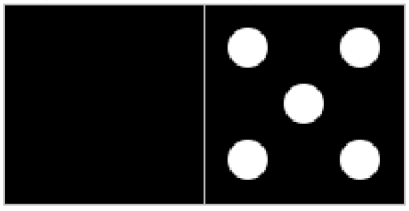
```
> ( domino 4 5 )
```



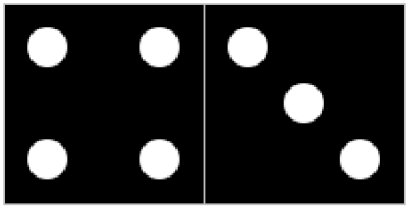
```
> ( domino 6 6 )
```



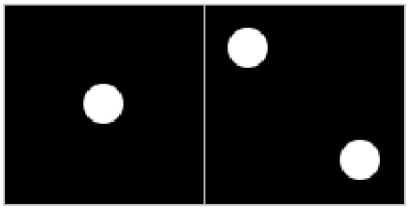
```
> ( domino 0 5 )
```



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



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



```
>
```

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

Task 7: Creation

Creation (image) ...

```
> myCreation
```

```
>
```

Code ...

```
#lang racket
( require 2htdp/image )
( define background ( circle 90 'solid (color 48 94 196)))
( define redspace (rotate 110 ( wedge 90 180 'solid (color 200 0 4))))
( define myCreation
  ( overlay/offset (text "λ" 255 "white") 5 0 ( overlay/align "left" "bottom" redspace background )
)
```