Racket Programming Assignment #2: Racket Functions and Recursion

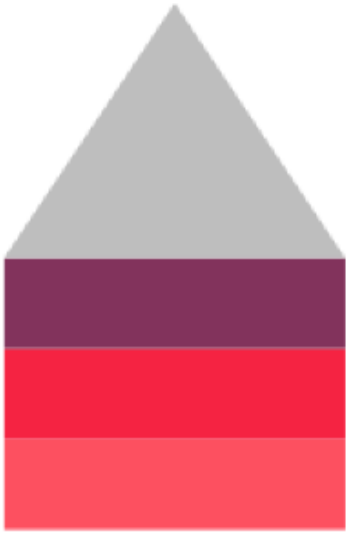
Task 1 - Colorful Permutations of Tract Houses

Code:

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

;define house function
(define (random-color) (color (random 256) (random 256) (random 256)))
(define (house width height color1 color2 color3)
  (define bottom (rectangle width height "solid" color1))
  (define middle (above (rectangle width height "solid" color2) bottom))
  (define top (above (rectangle width height "solid" color3) middle))
  (define roof (above (triangle width "solid" "gray")top))
  roof )

;define tract function
(define (tract width height)
  (define color1 (random-color))
  (define color2 (random-color))
  (define color3 (random-color))
  (define space (square 10 "solid" "white"))
  (beside
   (house width height color1 color2 color3)
   space
   (house width height color3 color2 color1)
   space
   (house width height color2 color1 color3)
   space
   (house width height color3 color1 color2)
   space
   (house width height color3 color2 color3)
   space
   (house width height color1 color3 color2)
   )
  )

```

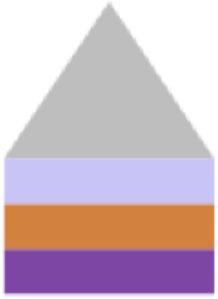
Demo:

```
> (house 130 40 (random-color) (random-color) (random-color))
```



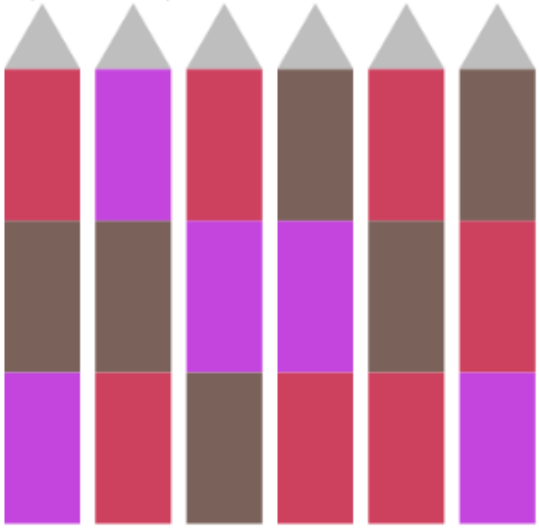
```
> (house 100 60 (random-color) (random-color) (random-color))
```

```
> (house 80 20 (random-color) (random-color) (random-color))
```

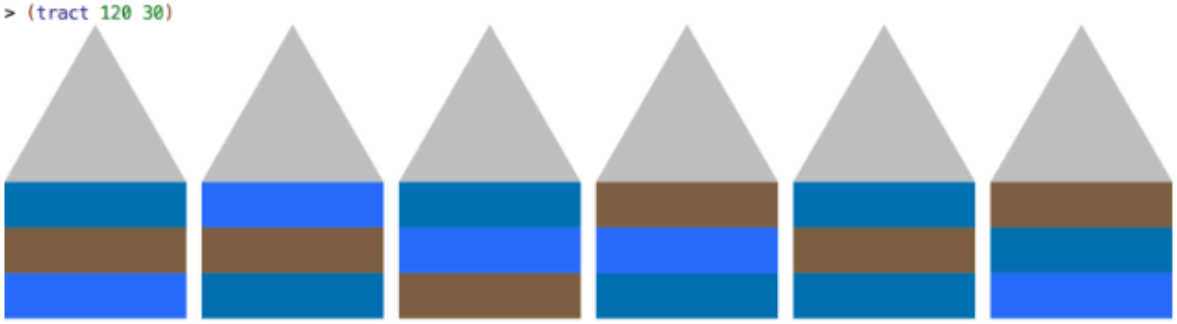


```
>
```

```
> (tract 50 100)
```



```
> (tract 120 30)
```



Task 2 - Dice
Code:

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

;roll for a 1
(define (roll-for-1)
  (define roll (roll-die))
  (display roll)
  (display " ")
  (if (= roll 1)
      (display "")
      (roll-for-1)
      )
  )

;roll for two 1's
(define (roll-for-11)
  (roll-for-1)
  (define roll-11 (roll-die))
  (display roll-11)
  (display " ")
  (if (= roll-11 1)
      (display "")
      (roll-for-11)
      )
  )

;roll for an odd number
(define (roll-for-odd)
  (define roll (roll-die))
  (display roll)
  (display " ")
  (cond
    ((even? roll)
     (roll-for-odd)
     )
    )
  )

;roll for an odd then an even number
(define (roll-for-odd-even)
  (define roll (roll-die))
  (display roll)
  (display " ")
  (cond
    ((odd? roll)
     (roll-for-odd-even)
     )
```

```
    )
  )

;roll for an odd, even, then odd number
(define (roll-for-odd-even-odd)
  (roll-for-odd-even)
  (define roll (roll-die))
  (display roll)
  (display " ")
  (cond
    ((even? roll)
     (roll-for-odd-even-odd)
     )
    )
  )

;roll for a pair of dice adding to 7 or 11, or a double pair
(define (roll-two-dice-for-a-lucky-pair)
  (define roll-1 (roll-die))
  (define roll-2 (roll-die))
  (display " (") (display roll-1) (display " , ") (display roll-2) (display ") ")
  (cond
    [ (eq? roll-1 roll-2) (display "") ]
    [ (eq? (+ roll-1 roll-2) 7) (display "") ]
    [ (eq? (+ roll-1 roll-2) 11) (display "") ]
    [ (roll-two-dice-for-a-lucky-pair) ]
    )

  )
```

Demo:

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

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

Task 3 - Number Sequences
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 ( triangle n )
   ( + n ( - n 1 ) )
)

( define ( triangular n )
   ( cond
      ( ( = n 1 ) 1 )
      ( ( > n 1 )
        ( + n ( triangular ( - n 1 ) ) )
      )
     )
   )
```

```
34
35 ( define ( sigma-add n a b)
36    ( cond
37       ( ( = n a )
38         ( + a b )
39       )
40       ( else
41         ( cond
42            ( ( < a ( + n 1 ) )
43              ( cond
44                 ( ( eq? (modulo n a ) 0 )
45                   ( sigma-add n ( + a 1 ) ( + b a ) )
46                   )
47                 ( else
48                   ( sigma-add n ( + a 1 ) b )
49                 )
50               )
51            )))
52       )
53    )
54
55 ( define ( sigma n )
56    ( sigma-add n 1 0 ) )
```

Demo:

```
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
>
```

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

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

Code:

```
1  #lang racket
2
3  (require 2htdp/image)
4  (define (rbg)(random 256))
5  (define (random-color) (color (rbg) (rbg) (rbg)))
6  (define space (square 20 "solid" "white"))
7  (define diameter 30)
8  (define (dot diameter color) (circle diameter 'solid (random-color)))
9
10 (define (row n)
11   (cond
12     ((= n 0) empty-image)((> n 0) (beside (row (- n 1))(dot diameter color)space))))
13
14 (define (column c x)
15   (cond
16     (( = c 0) empty-image)(( > c 0) (above (column (- c 1) x) space (row x)))))
17
18 (define (hirst-dots n)(column n n))
```

Demo:

Task 5 - Channeling Frank Stella

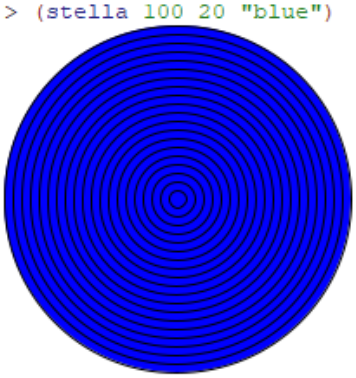
Code:

```
1  #lang racket
2  (require 2htdp/image)
3
4  (define (stella radius count color)
5    (define unit (/ radius count))
6    (paint-stella 1 count unit color)
7  )
8
9  (define (paint-stella from to unit color)
10   (define rad (* from unit))
11   (cond
12     ((= from to)
13      (framed-circle rad color))
14     ((< from to)
15      (overlay (framed-circle rad color)(paint-stella (+ from 1) to unit color))
16     )
17   )
18 )
19
20 (define (framed-circle radius color)
21   (overlay (circle radius "outline" "black")
22            (circle radius "solid" color)
23   )
24 )
```

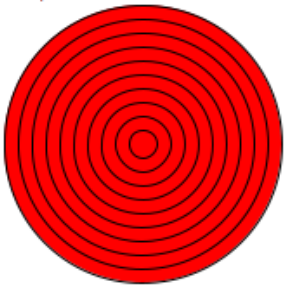
Demo:

Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.

```
> (stella 100 20 "blue")
```



```
> (stella 80 10 "red")
```



```
>
```

Task 6 - Dominoes
Original Code:

```
#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))

(define d (* diameter-of-pip 1.4))
(define nd (* -1 d))

(define blank-tile (square side-of-tile "solid" "black"))
(define (pip)(circle radius-of-pip "solid" "white"))

(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 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 (domino a b)
  (beside (tile a)(tile b))
)

(define (tile x)
  (cond
    ((= x 0) tile0)
    ((= x 1) tile1)
    ((= x 2) tile2)
    ((= x 3) tile3)
  )
)
```

Extended Code:

```
#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))

(define d (* diameter-of-pip 1.4))
(define nd (* -1 d))

(define blank-tile (square side-of-tile "solid" "black"))
(define (pip)(circle radius-of-pip "solid" "white"))

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

(define basic-tile5
  (overlay (pip) basic-tile4)
)

(define basic-tile6
  (overlay/offset (pip) 0 d
    (overlay/offset (pip) 0 nd
      (overlay basic-tile4 basic-tile2)
    )
  )
)

(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))
```

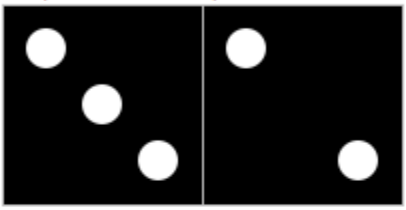
```
55
56 (define (domino a b)
57   (beside (tile a)(tile b))
58 )
59
60 (define (tile x)
61   (cond
62     ((= x 0) tile0)
63     ((= x 1) tile1)
64     ((= x 2) tile2)
65     ((= x 3) tile3)
66     ((= x 4) tile4)
67     ((= x 5) tile5)
68     ((= x 6) tile6)
69   )
70 )
```

Demo for 0, 1, 2, and 3 pip dominos:

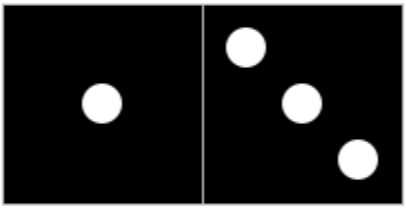
Language: racket, with debugging; memory limit: 128 MB.

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

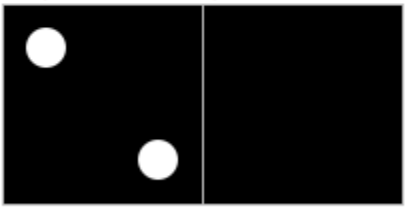
```
> (domino 3 2)
```



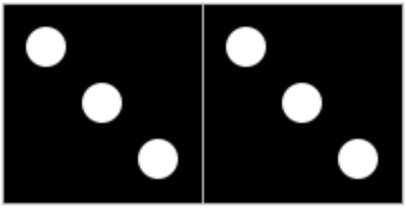
```
> (domino 1 3)
```



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



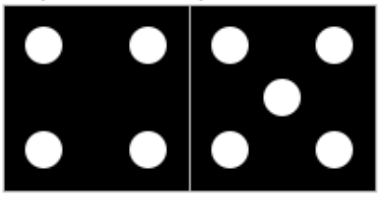
```
> (domino 3 3)
```



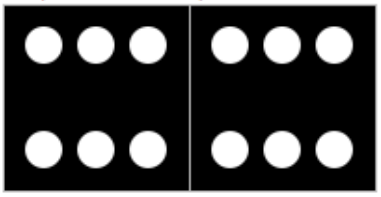
```
>
```

Final domino rendering demo:

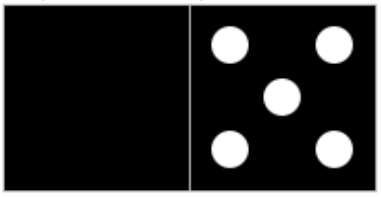
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (domino 4 5)
```



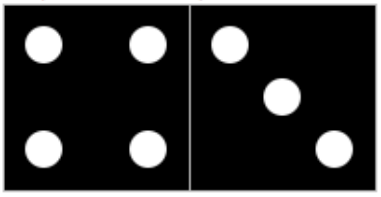
```
> (domino 6 6)
```



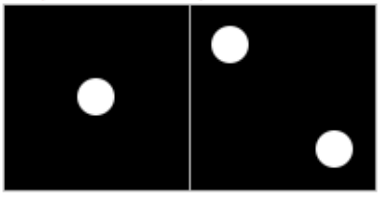
```
> (domino 0 5)
```



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



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



```
>
```

Task 7 - Creation

Code:

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

(define background (square 400 "solid" "black"))
(define big-ball (circle 70 "solid" "white"))
(define medium-ball (circle 60 "solid" "white"))
(define small-ball (circle 50 "solid" "white"))
(define carrot (triangle 20 "solid" "orange"))
(define little-circle (circle 5 "solid" "black"))
(define arms (rectangle 100 10 "solid" "brown"))
(define eye-space (rectangle 30 10 "solid" "white"))
(define button-space (circle 10 "solid" "white"))
(define eye-position (circle 40 "solid" "white"))


(define set-small-ball
  (overlay
   carrot
   small-ball)
)


(define set-buttons
  (overlay
   (above
    little-circle
    button-space
    little-circle
    button-space
    little-circle
   ) medium-ball)
)


(define set-medium-ball
 (beside
  arms set-buttons arms
 )
)

(define make-snowman
  (above
     set-small-ball
     set-medium-ball
     big-ball
  )
)

(define my-creation
  (overlay
     make-snowman
     background
  )
)
```

Demo:

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> my-creation
```