Assignment #2

Trevor Strickland, CSC344

September 15, 2022

Learning Abstract: This assignment is an exercise in recursive programming. The aim is to demonstrate the use of iteration through recursion in a beautifully simple fashion. In this assignment, I created several functions to display little houses, dice rolls, and different number sequences.

Task 1: Colorful Permutations of Tract Houses

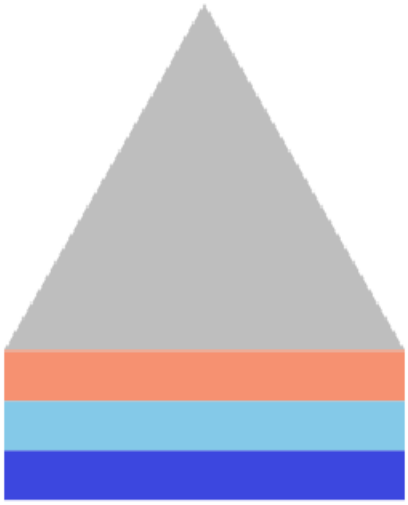
Code:

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

(define (random-color) (color (random 256) (random 256) (random 256)))

(define (house width height color-a color-b color-c)
  (define low (rectangle width height "solid" color-a))
  (define mid (above (rectangle width height "solid" color-b) low))
  (define high (above (rectangle width height "solid" color-c) mid))
  (define roof (above (triangle width "solid" "gray") high))
  roof
)

(define (tract width height)
  (define color-a (random-color))
  (define color-b (random-color))
  (define color-c (random-color))
  (define h1 (house width height color-a color-b color-c))
  (define h2 (house width height color-a color-c color-b))
  (define h3 (house width height color-b color-a color-c))
  (define h4 (house width height color-b color-c color-a))
  (define h5 (house width height color-c color-a color-b))
  (define h6 (house width height color-c color-b color-a))
  (define space (square 10 "solid" "white"))
  (display (beside h1 space h2 space h3 space h4 space h5 space h6))
)
```

Demo:

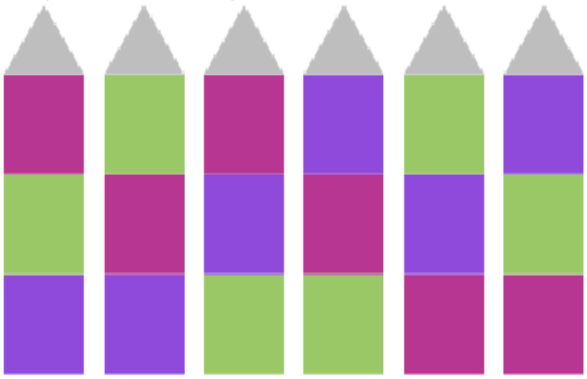
```
> (house 50 40 (random-color) (random-color) (random-color))
```

```
> (house 200 25 (random-color) (random-color) (random-color))
```

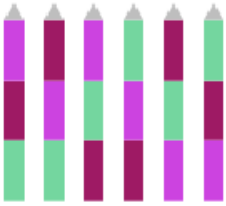


```
> (house 10 30 (random-color) (random-color) (random-color))
```

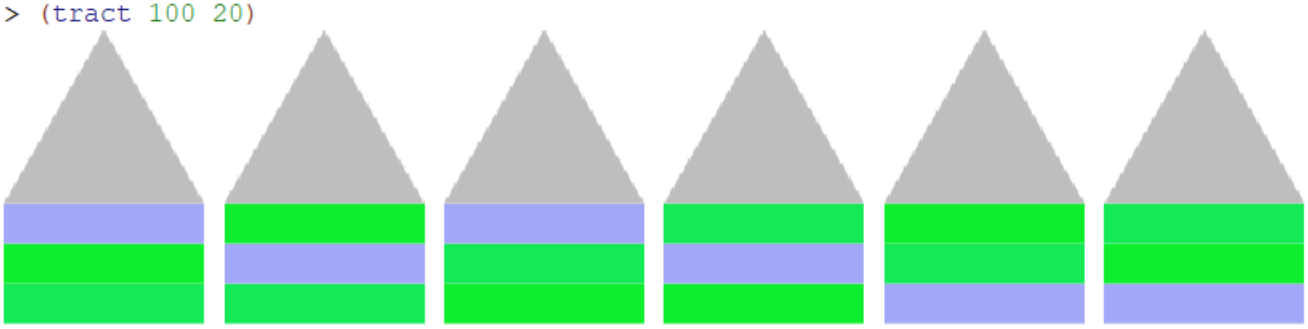
```
> (tract 40 50)
```



```
> (tract 10 30)
```



```
> (tract 100 20)
```



Task 2: Dice

Code:

```
#lang racket

(define (roll-die) (+ 1 (random 6)))

(define (roll-for-1)
  (define roll (roll-die))
  (display roll)
  (display " ")
  (cond ((> roll 1)
         (roll-for-1)
        )
  )
)

(define (roll-for-11)
  (roll-for-1)
  (define roll (roll-die))
  (display roll)
  (display " ")
  (cond ((> roll 1)
         (roll-for-11)
        )
  )
)

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

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

(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)
        )
  )
)

(define (roll-two-dice-for-a-lucky-pair)
  (define trial-a (roll-die))
  (define trial-b (roll-die))
  (display "(") (display trial-a) (display ", ") (display trial-b) (display
      ") ")
  (cond ((and (and (not (eq? 7 (+ trial-a trial-b))) (not (eq? 11 (+ trial-a
      trial-b)))) (not (eq? trial-a trial-b)))
    (roll-two-dice-for-a-lucky-pair)))
)
```

Demo:

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

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

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-add n)
  (+ n (- n 1))
)

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

(define (sigma-add n a b)
  (cond
    ((= n a)
     (+ a b)
    )
    (else (cond
      ((< a (+ n 1))
        (cond
          ((eq? (modulo n a) 0)
            (sigma-add n (+ a 1) (+ b a))
          )
        (else
          (sigma-add n (+ a 1) b)
        )
      )
```

```
    )))
  )
)

(define (sigma n) (sigma-add n 1 0))
```

Demo:

```
> (square 5)
25
> (square 100)
10000
> (sequence square 20)
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361 400
> (cube 4)
64
> (cube 9)
729
> (sequence cube 20)
1 8 27 64 125 216 343 512 729 1000 1331 1728 2197 2744 3375 4096 4913 5832 6859 8000
> (triangular 2)
3
> (triangular 5)
15
> (triangular 10)
55
> (triangular 12)
78
> (triangular 15)
120
> (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 15)
1 3 4 7 6 12 8 15 13 18 12 28 14 24 24
> (sequence sigma 25)
1 3 4 7 6 12 8 15 13 18 12 28 14 24 24 31 18 39 20 42 32 36 24 60 31
```