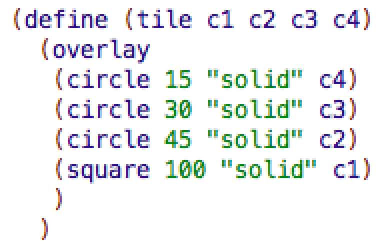
Second Racket Programming Assignment

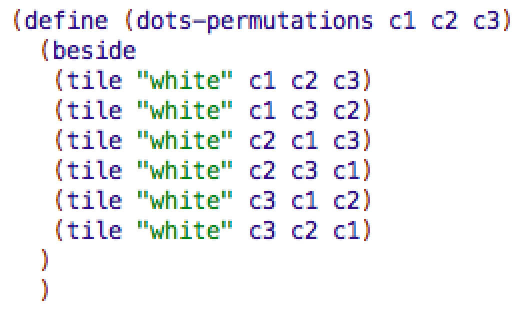
Abstract
This document contains various images generated by the DrRacket program with use of the 2htdp/image library. The library contains functions that can generate shapes and colors and also allows for manipulation of the shapes. With the library I was able to create images based on concepts such as permutations, iteration, and recursion. Some images invoke famous artists such as Frank Stella or Damien Hirst, while the others were personally designed.

Task 1 - Permutations of Randomly Colored Stacked Dots

Code



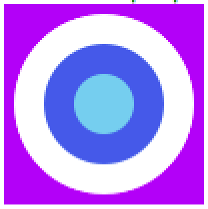
```
(define (tile c1 c2 c3 c4)
  (overlay
   (circle 15 "solid" c4)
   (circle 30 "solid" c3)
   (circle 45 "solid" c2)
   (square 100 "solid" c1)
   )
  )
```



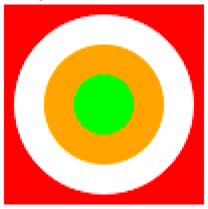
```
(define (dots-permutations c1 c2 c3)
  (beside
   (tile "white" c1 c2 c3)
   (tile "white" c1 c3 c2)
   (tile "white" c2 c1 c3)
   (tile "white" c2 c3 c1)
   (tile "white" c3 c1 c2)
   (tile "white" c3 c2 c1)
  )
  )
```

Demo

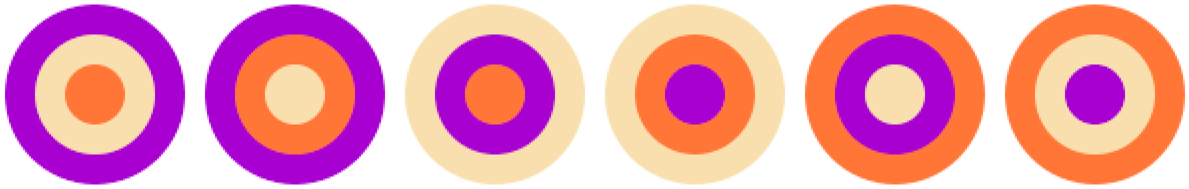
```
> (tile "purple" "white" "royal blue" "sky blue")
```



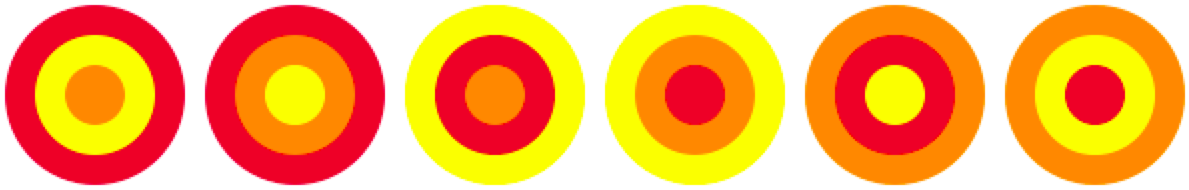
```
> (tile "red" "white" "orange" "lime")
```



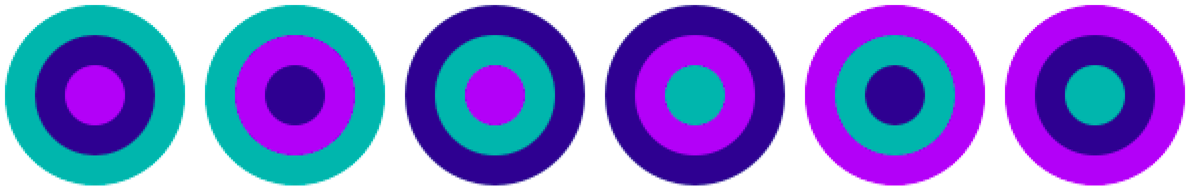
```
> (dots-permutations "dark orchid" "wheat" "coral")
```



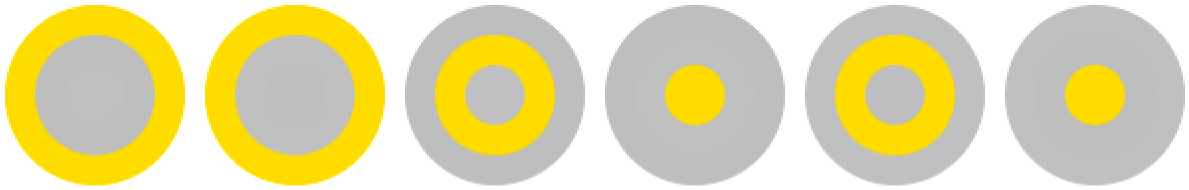
```
> (dots-permutations "crimson" "yellow" "dark orange")
```



```
> (dots-permutations "light sea green" "navy" "purple")
```



```
> (dots-permutations "gold" "gray" "silver")
```



```
>
```

Task 2 - Number Sequences

Code

```
(define (natural-sequence n)
  (cond
    ((> n 0)
     (natural-sequence (- n 1))
     (display n) (display " ")
     )
    )
  )

(define (copies c n)
  (cond
    ((> n 0)
     (copies c (- n 1))
     (display c) (display " ")
    )
  )
  )

(define (special-natural-sequence n)
  (cond
    ((> n 0)
     (special-natural-sequence (- n 1))
     (copies n n)
     )
    )
  )
```

Demo

```
> (natural-sequence 5)
1 2 3 4 5
> (natural-sequence 18)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
> (natural-sequence 9)
1 2 3 4 5 6 7 8 9
> (natural-sequence 25)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
> (copies "a" 11)
a a a a a a a a a a a
> (copies 9 9)
9 9 9 9 9 9 9 9 9
> (copies 7 3)
7 7 7
> (copies 13 5)
13 13 13 13 13
> (special-natural-sequence 5)
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
> (special-natural-sequence 20)
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 10 10
10 10 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11 11 12 12 12 12 12 12 12 12 12 12 12 12 13
13 13 13 13 13 13 13 13 13 13 13 13 14 14 14 14 14 14 14 14 14 14 14 14 14 14 15 15 15 15 15 15
15 15 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17
17 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 19 19 19 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20
> (special-natural-sequence 10)
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 10 10
10 10 10 10 10 10 10 10
> (special-natural-sequence 8)
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8
>
```

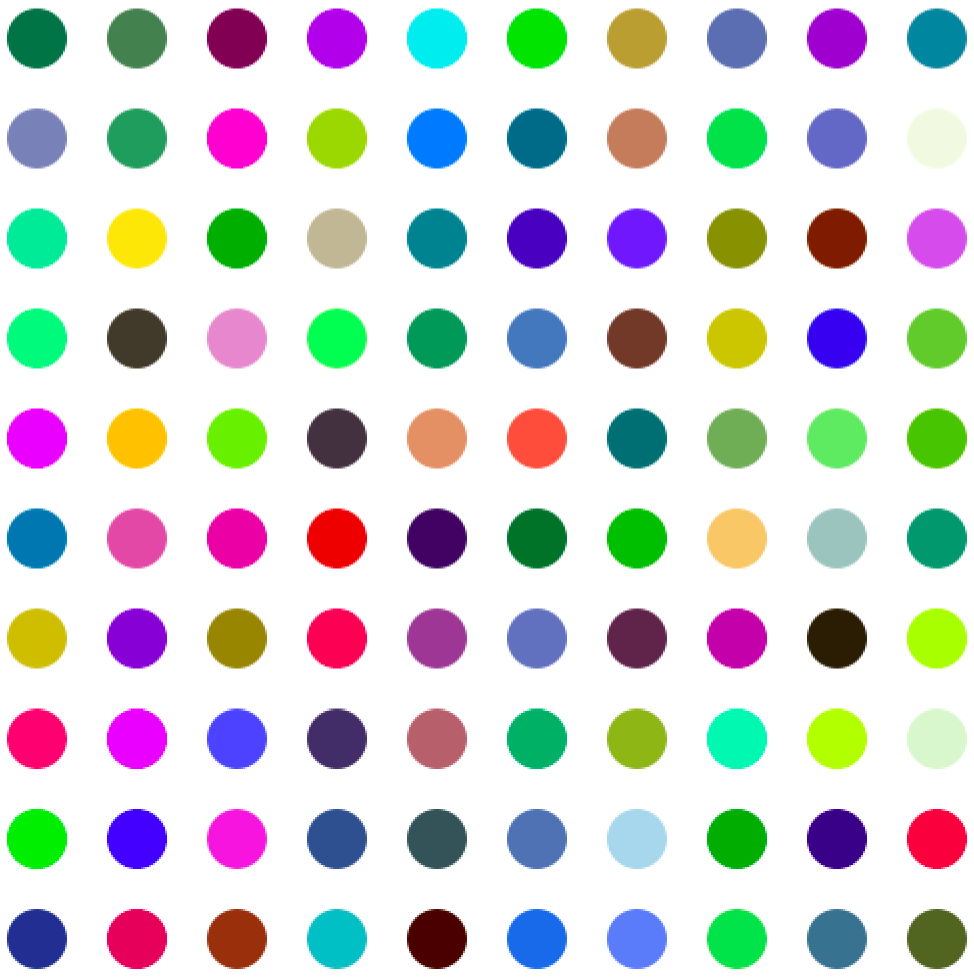
Task 3 - Hirst Dots

Code

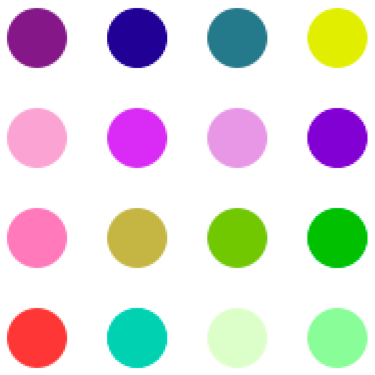
```
(define (hirst-dots n) (hirst-rectangle n n))

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

(define (hirst-rectangle r c)
  (cond
    ((= r 0)
     empty-image
     )
    ((> r 0)
     (above (hirst-rectangle (- r 1) c) (hirst-row c))
     )
    )
  )

(define (hirst-dot)
  (overlay (random-dot) (square 50 "solid" "white")))
; enclosing the size 30 dot within a size 50 square ensures that the dots are 20 pixels apart

(define (random-dot) (circle 15 "solid" (random-color)))
( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )
( define ( rgb-value ) ( random 256 ) )
```

Demo

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



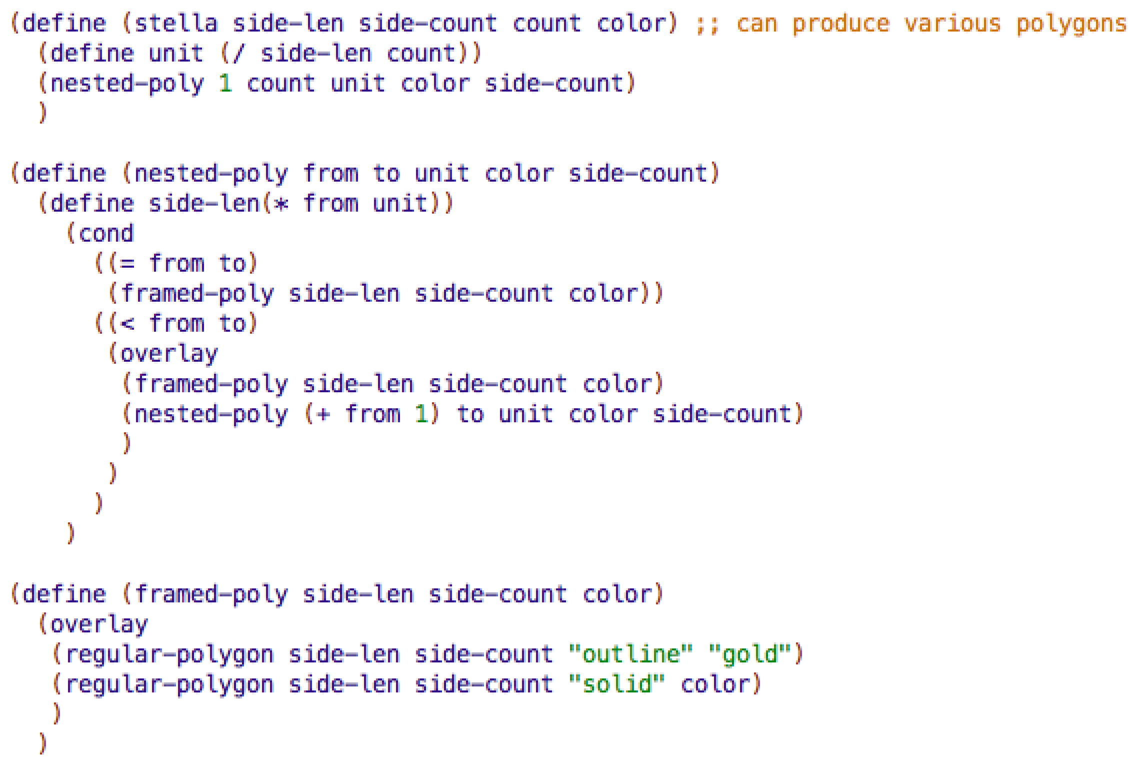
```
> (hirst-dots 4)
```



```
> |
```

Task 4 - Stella Thing

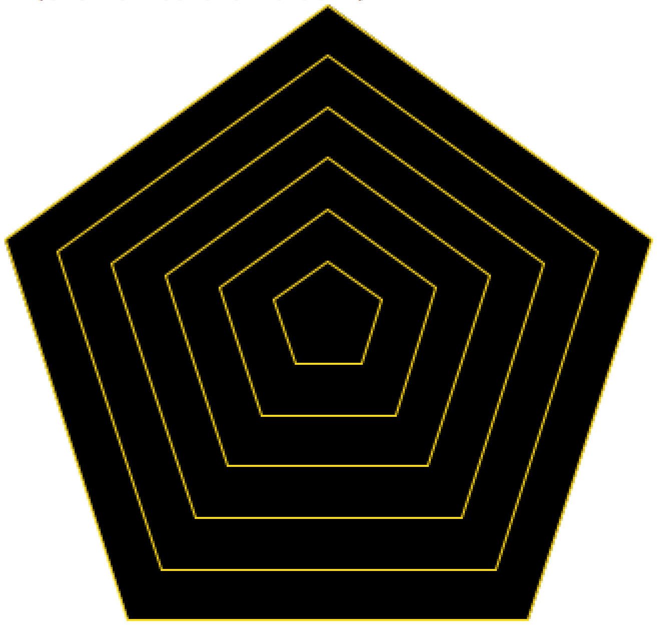
Code



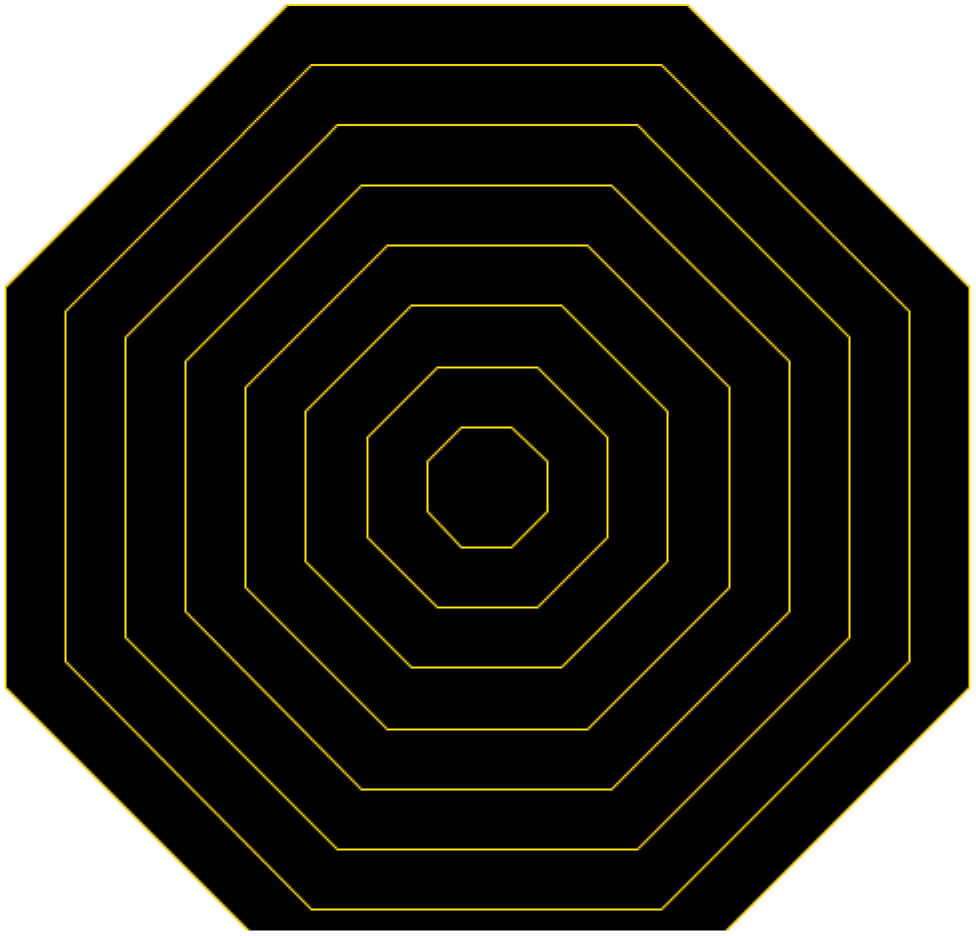
```
(define (stella side-len side-count count color) ;; can produce various polygons
  (define unit (/ side-len count))
  (nested-poly 1 count unit color side-count)
  )

(define (nested-poly from to unit color side-count)
  (define side-len(* from unit))
    (cond
      ((= from to)
       (framed-poly side-len side-count color))
      ((< from to)
       (overlay
        (framed-poly side-len side-count color)
        (nested-poly (+ from 1) to unit color side-count)
        )
       )
      )
    )

(define (framed-poly side-len side-count color)
  (overlay
   (regular-polygon side-len side-count "outline" "gold")
   (regular-polygon side-len side-count "solid" color)
   )
  )
```

Demo

```
> (stella 200 5 6 "black")
```

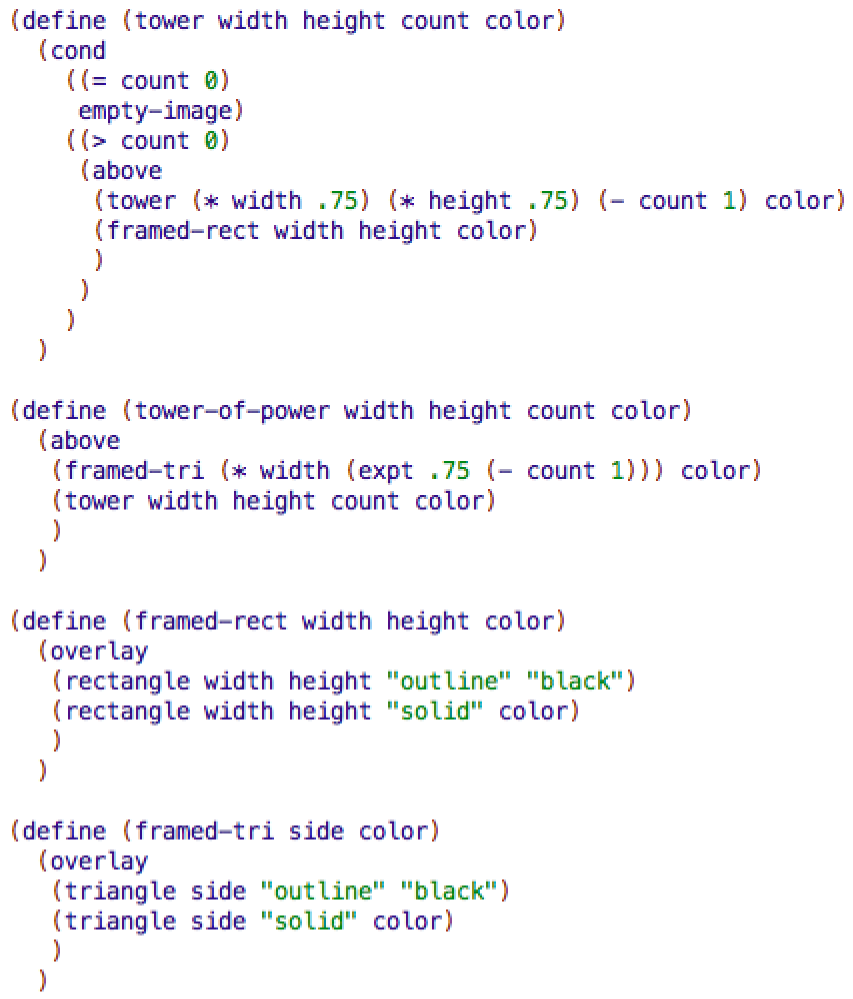


```
> (stella 200 8 8 "black")
```



Task 5 - Creation

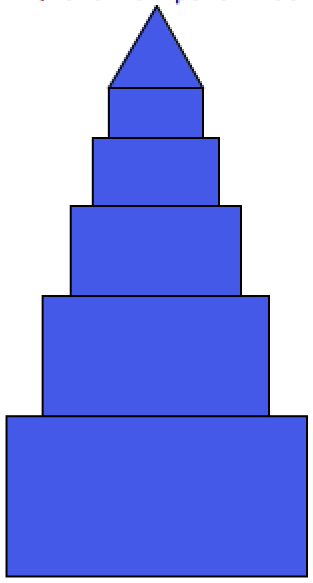
Code



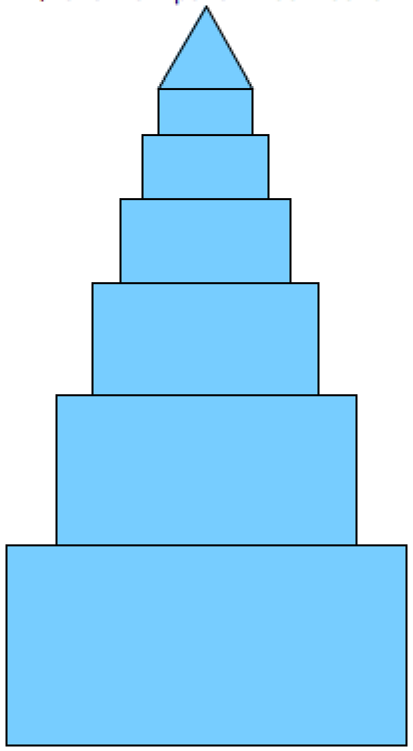
```
(define (tower width height count color)
  (cond
    ((= count 0)
     empty-image)
    ((> count 0)
     (above
      (tower (* width .75) (* height .75) (- count 1) color)
      (framed-rect width height color)
      )
     )
    )
  )

(define (tower-of-power width height count color)
  (above
   (framed-tri (* width (expt .75 (- count 1))) color)
   (tower width height count color)
   )
  )

(define (framed-rect width height color)
  (overlay
   (rectangle width height "outline" "black")
   (rectangle width height "solid" color)
   )
  )

(define (framed-tri side color)
  (overlay
   (triangle side "outline" "black")
   (triangle side "solid" color)
   )
  )
```

Demo

```
> (tower-of-power 150 80 5 "royal blue")
```



```
> (tower-of-power 200 100 6 "light sky blue")
```



```
>
```