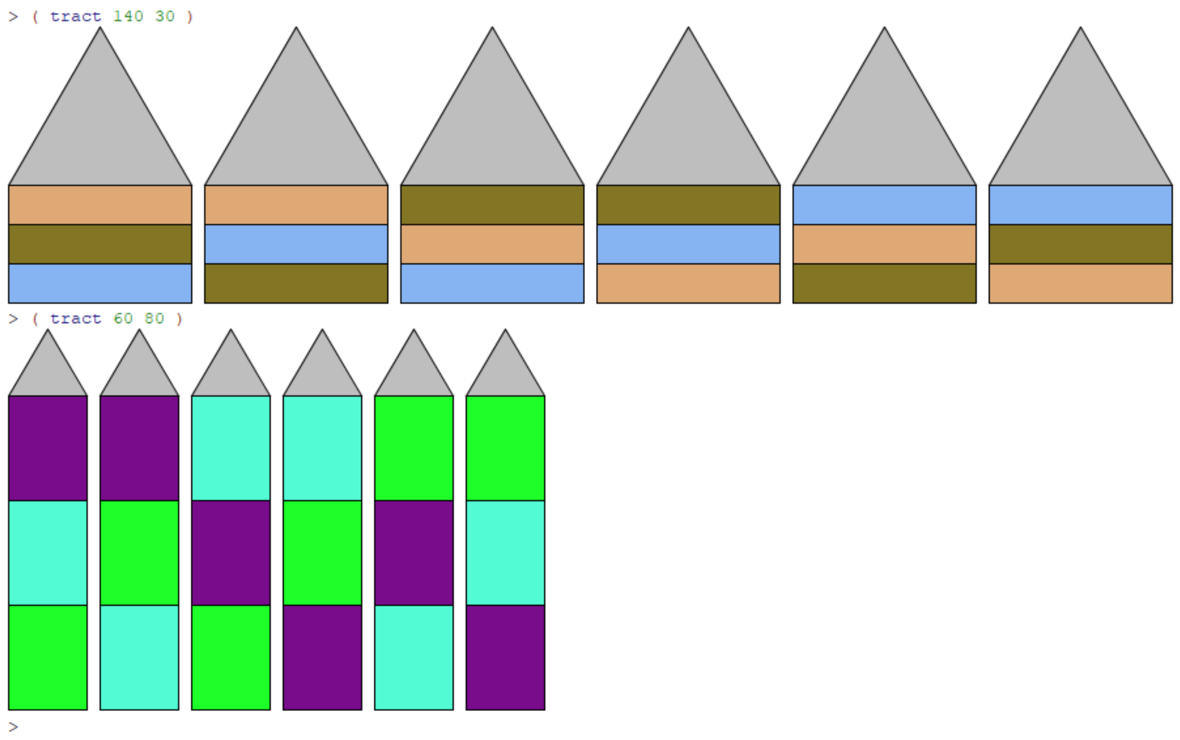
Racket Programming Lab 2

Abstract

This lab is about getting comfortable with recursion and image generation in Racket. Several themes are repeated throughout the lab. First image generation, then recursion, then the two topics combined.

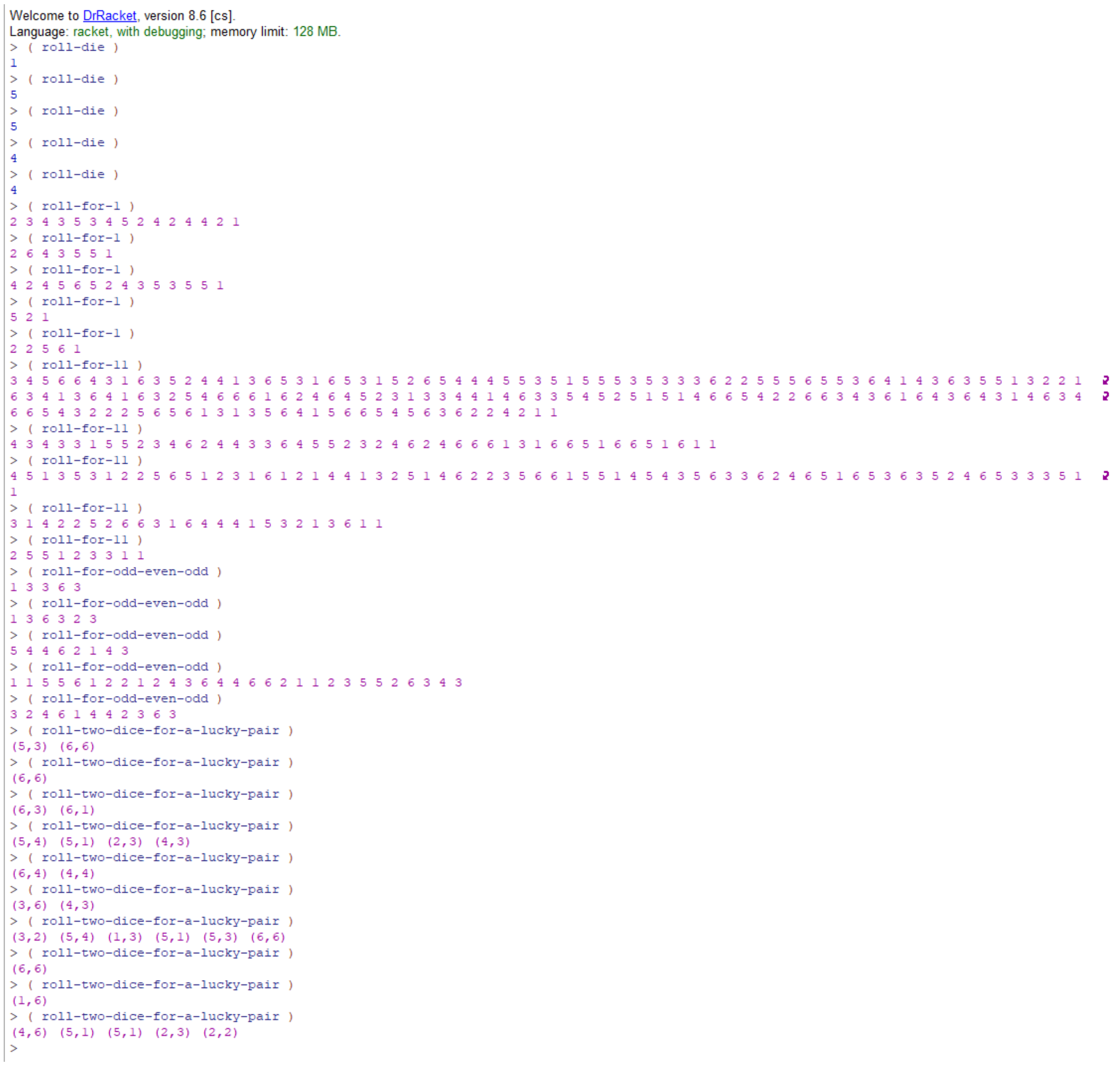
Task 1: House & Tract



```
> ( tract 140 30 )
> ( tract 60 80 )
>
```

```
#lang racket
( require 2htdp/image)
( define (random-color)
   ( color (random 256) (random 256) (random 256) )
   )

( define (house-floor width height color)
   ( overlay
     ( rectangle width height 'outline 'black )
     ( rectangle width height 'solid color )
     )
   )

( define (house-roof width )
   ( overlay
     ( triangle width 'outline 'black )
    ( triangle width 'solid 'gray )
    )
   )

( define (house width height color1 color2 color3 )
   ( above
     ( house-roof width )
     ( house-floor width height color1 )
     ( house-floor width height color2 )
     ( house-floor width height color3 )
     )
   )

(define (tract width height)
  ( define color1 (random-color) )
  ( define color2 (random-color) )
  ( define color3 (random-color) )
  ( beside
    ( house width height color1 color2 color3 )
    ( square 10 'solid 'white)
    ( house width height color1 color3 color2 )
    ( square 10 'solid 'white)
    ( house width height color2 color1 color3 )
    ( square 10 'solid 'white)
    ( house width height color2 color3 color1 )
    ( square 10 'solid 'white)
    ( house width height color3 color1 color2 )
    ( square 10 'solid 'white)
    ( house width height color3 color2 color1 )
    )
  )
```

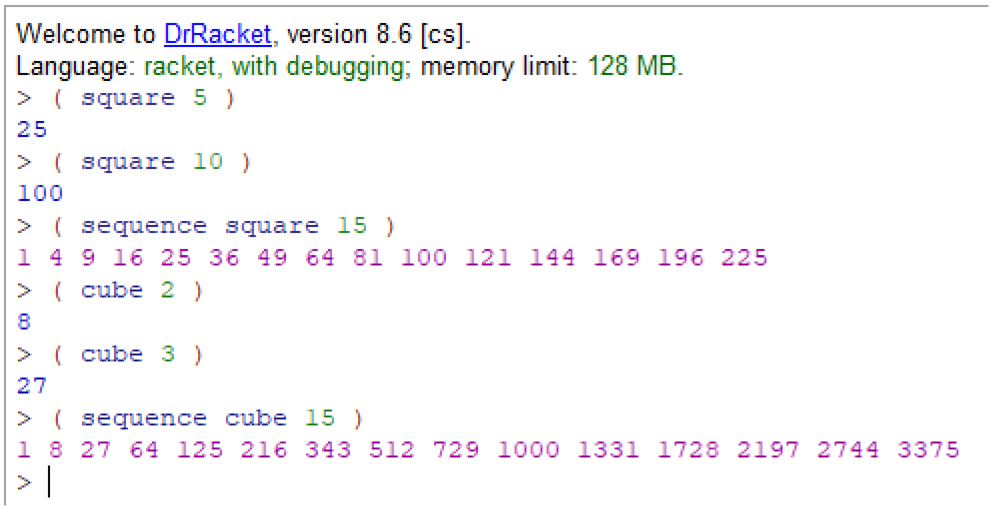
Task 2: Dice



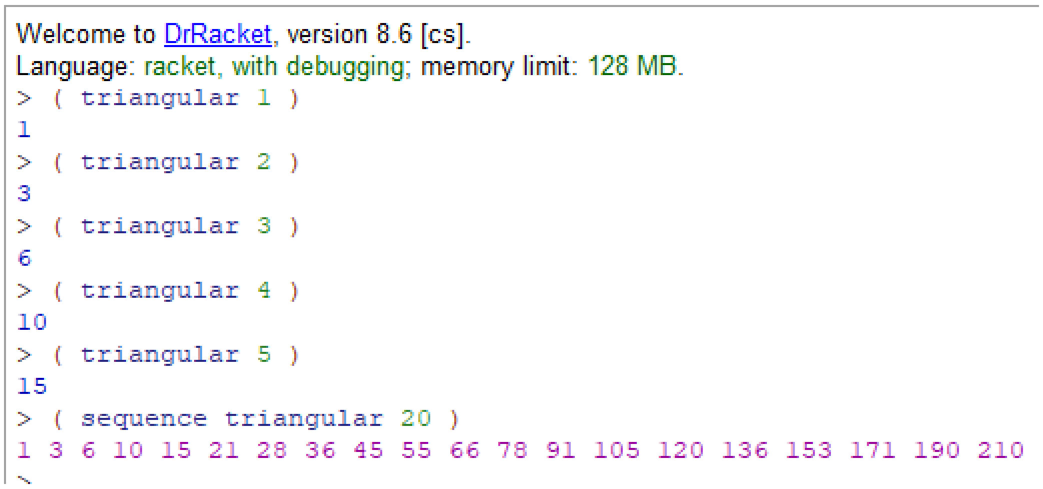
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( roll-die )
1
> ( roll-die )
5
> ( roll-die )
5
> ( roll-die )
4
> ( roll-die )
4
> ( roll-for-1 )
2 3 4 3 5 3 4 5 2 4 2 4 4 2 1
> ( roll-for-1 )
2 6 4 3 5 5 1
> ( roll-for-1 )
4 2 4 5 6 5 2 4 3 5 3 5 5 1
> ( roll-for-1 )
5 2 1
> ( roll-for-1 )
2 2 5 6 1
> ( roll-for-11 )
3 4 5 6 6 4 3 1 6 3 5 2 4 4 1 3 6 5 3 1 6 5 3 1 5 2 6 5 4 4 4 5 5 3 5 1 5 5 5 3 5 3 3 3 6 2 2 5 5 5 6 5 5 3 6 4 1 4 3 6 3 5 5 1 3 2 2 1
6 3 4 1 3 6 4 1 6 3 2 5 4 6 6 6 1 6 2 4 6 4 5 2 3 1 3 3 4 4 1 4 6 3 3 5 4 5 2 5 1 5 1 4 6 6 5 4 2 2 6 6 3 4 3 6 1 6 4 3 6 4 3 1 4 6 3 4
6 6 5 4 3 2 2 2 5 6 5 6 1 3 1 3 5 6 4 1 5 6 6 5 4 5 6 3 6 2 2 4 2 1 1
> ( roll-for-11 )
4 3 4 3 3 1 5 5 2 3 4 6 2 4 4 3 3 6 4 5 5 2 3 2 4 6 2 4 6 6 6 1 3 1 6 6 5 1 6 6 5 1 6 1 1
> ( roll-for-11 )
4 5 1 3 5 3 1 2 2 5 6 5 1 2 3 1 6 1 2 1 4 4 1 3 2 5 1 4 6 2 2 3 5 6 6 1 5 5 1 4 5 4 3 5 6 3 3 6 2 4 6 5 1 6 5 3 6 3 5 2 4 6 5 3 3 3 5 1
1
> ( roll-for-11 )
3 1 4 2 2 5 2 6 6 3 1 6 4 4 4 1 5 3 2 1 3 6 1 1
> ( roll-for-11 )
2 5 5 1 2 3 3 1 1
> ( roll-for-odd-even-odd )
1 3 3 6 3
> ( roll-for-odd-even-odd )
1 3 6 3 2 3
> ( roll-for-odd-even-odd )
5 4 4 6 2 1 4 3
> ( roll-for-odd-even-odd )
1 1 5 5 6 1 2 2 1 2 4 3 6 4 4 6 6 2 1 1 2 3 5 5 2 6 3 4 3
> ( roll-for-odd-even-odd )
3 2 4 6 1 4 4 2 3 6 3
> ( roll-two-dice-for-a-lucky-pair )
(5,3) (6,6)
> ( roll-two-dice-for-a-lucky-pair )
(6,6)
> ( roll-two-dice-for-a-lucky-pair )
(6,3) (6,1)
> ( roll-two-dice-for-a-lucky-pair )
(5,4) (5,1) (2,3) (4,3)
> ( roll-two-dice-for-a-lucky-pair )
(6,4) (4,4)
> ( roll-two-dice-for-a-lucky-pair )
(3,6) (4,3)
> ( roll-two-dice-for-a-lucky-pair )
(3,2) (5,4) (1,3) (5,1) (5,3) (6,6)
> ( roll-two-dice-for-a-lucky-pair )
(6,6)
> ( roll-two-dice-for-a-lucky-pair )
(1,6)
> ( roll-two-dice-for-a-lucky-pair )
(4,6) (5,1) (5,1) (2,3) (2,2)
>
```

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

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

( define ( roll-for-11 )
   ( roll-for-1 )
   ( define deciding-roll (roll-die) )
   ( display deciding-roll )
   ( display " " )
   ( if ( = deciding-roll 1)
        ( display "" )
        ( roll-for-11 )
        )
   )

( define ( roll-for-odd )
   ( define roll ( roll-die ) )
   ( display roll )
   ( display " " )
   ( if ( = ( modulo roll 2) 1 )
        ( display "")
        ( roll-for-odd )
        )
   )

( define ( roll-for-odd-even )
   ( roll-for-odd )
   ( define roll (roll-die) )
   ( display roll )
   ( display " " )
   ( if ( = ( modulo roll 2 ) 0)
        ( display "")
        ( roll-for-odd-even )
        )
   )

( define ( roll-for-odd-even-odd )
   ( roll-for-odd-even )
   ( define roll (roll-die) )
   ( display roll )
   ( display " " )
   ( if ( = ( modulo roll 2 ) 1)
        ( display "")
        ( roll-for-odd-even-odd )
        )
   )

( define (display-tuple num1 num2)
   ( display "(" )
   ( display num1 )
   ( display "," )
   ( display num2 )
   ( display ")" )
   ( display " " )
   )

( define ( roll-two-dice-for-a-lucky-pair )
   ( define roll1 ( roll-die ) )
   ( define roll2 ( roll-die ) )
   ( display-tuple roll1 roll2 )
   ( cond
      ( ( = ( + roll1 roll2) 7 )
        ( display "" )
        )

      ( ( = ( + roll1 roll2) 11 )
        ( display "" )
        )

      ( ( = roll1 roll2 )
        ( display "" )
        )
      ( else ( roll-two-dice-for-a-lucky-pair ) )
      )
   )
```

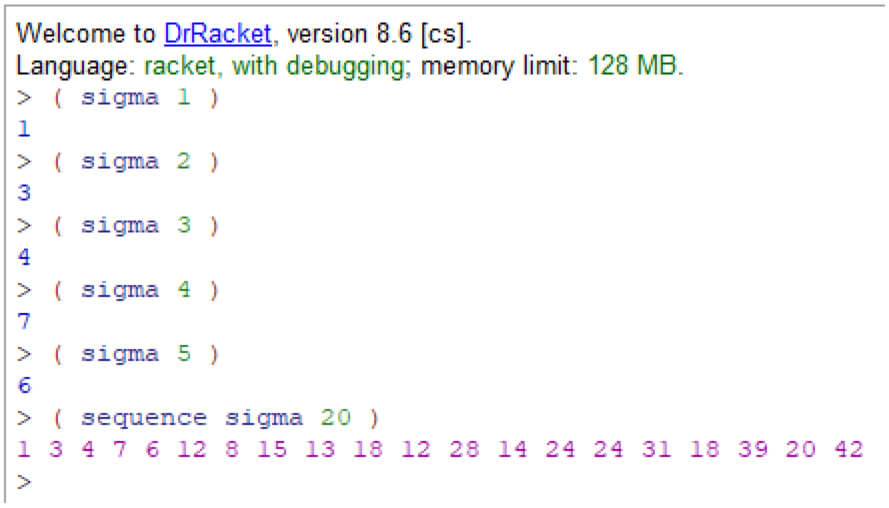
Task 3: Number Sequences



```
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
> 
```



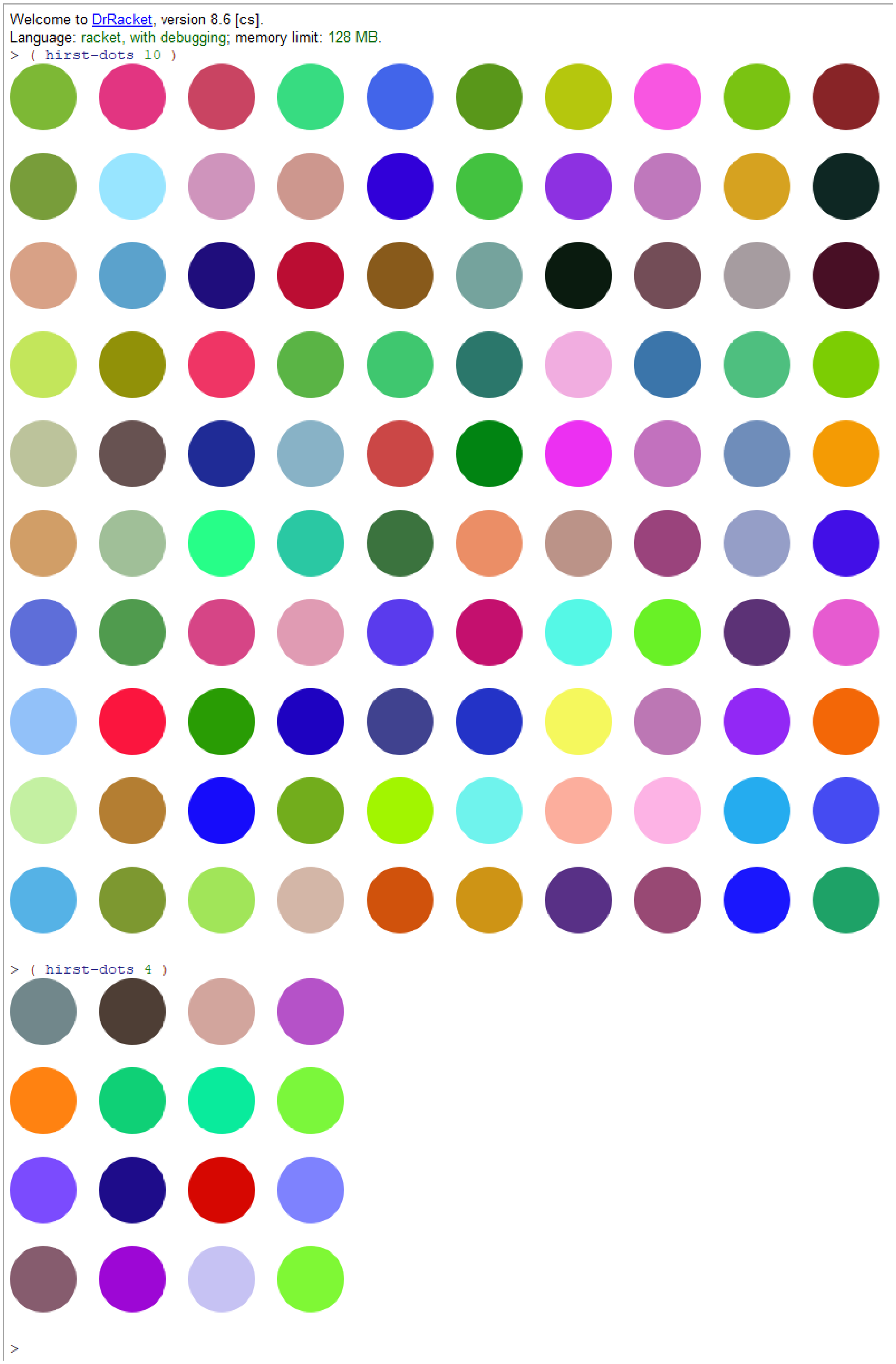
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( triangular 1 )
1
> ( triangular 2 )
3
> ( triangular 3 )
6
> ( triangular 4 )
10
> ( triangular 5 )
15
> ( sequence triangular 20 )
1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210
>
```



```
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
>
```

```
#lang racket

( define ( square n )
   (* n n)
   )
( define ( cube n )
   (* n n n)
   )

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

( define ( sigma n )
   ( get-prime-sums n 0 1 )
   )

( define (get-prime-sums n sum i )
   ( cond
      (
       ( = n i )
       ( + sum n )
       )
      (
       ( = ( modulo n i) 0 )
       ( get-prime-sums n ( + sum i ) ( + i 1 ) )
       )
      ( else ( get-prime-sums n sum ( + i 1) ) )
      )
   )

( define ( sequence name n )
   ( cond
      ( (= n 1)
        ( display ( name 1 ) ) ( display " " )
        )
      ( else
        ( sequence name ( - n 1 ) )
        ( display ( name n ) ) ( display " " )
        )
      )
   )
```

Task 4: Hirst Dots



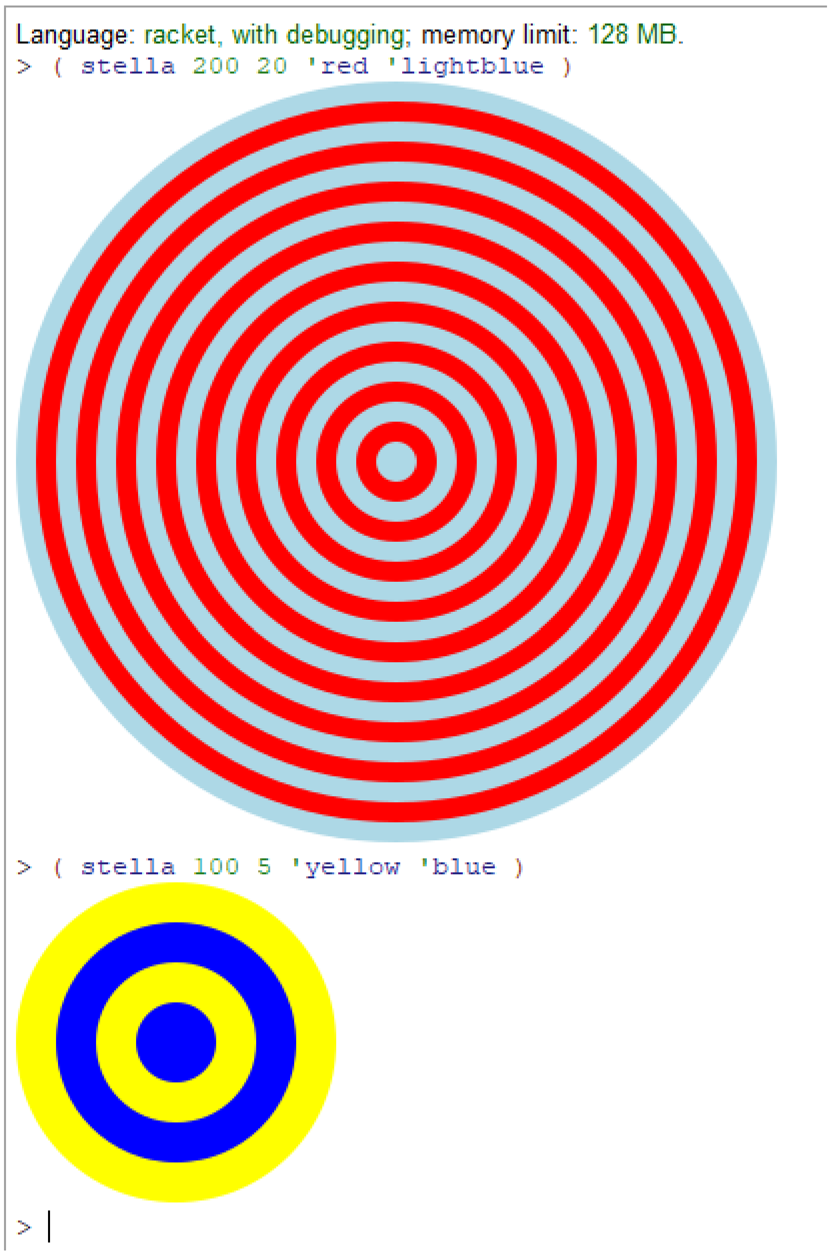
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( hirst-dots 10 )
```

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

```
>
```

```
#lang racket
( require 2htdp/image )
( define ( random-color )
   ( color ( random 256 ) ( random 256 ) ( random 256 ) 255 )
   )

( define transparent ( color 0 0 0 0 ) )

( define ( draw-dot )
   ( circle 30 'solid ( random-color ) )
   )

( define ( row-of-dots n row )
   ( cond
      (
       ( = n 0 )
       row
       )
      (
       ( > n 0)
       ( define new-row ( beside row ( draw-dot ) (square 20 'solid transparent ) ) )
       ( row-of-dots (- n 1) new-row)
       )
      )
   )

( define (draw-row n)
   ( row-of-dots n empty-image )
   )

( define ( set-of-rows n r im )
   ( cond
      (
       ( = n 0 )
       im
       )
      (
       ( > n 0)
       ( define new-set ( above im ( draw-row r ) (square 20 'solid transparent ) ) )
       ( set-of-rows (- n 1) r new-set )
       )
      )
   )

( define ( hirst-dots n )
   ( set-of-rows n n empty-image )
   )
```

Task 5: Channelling Frank Stella



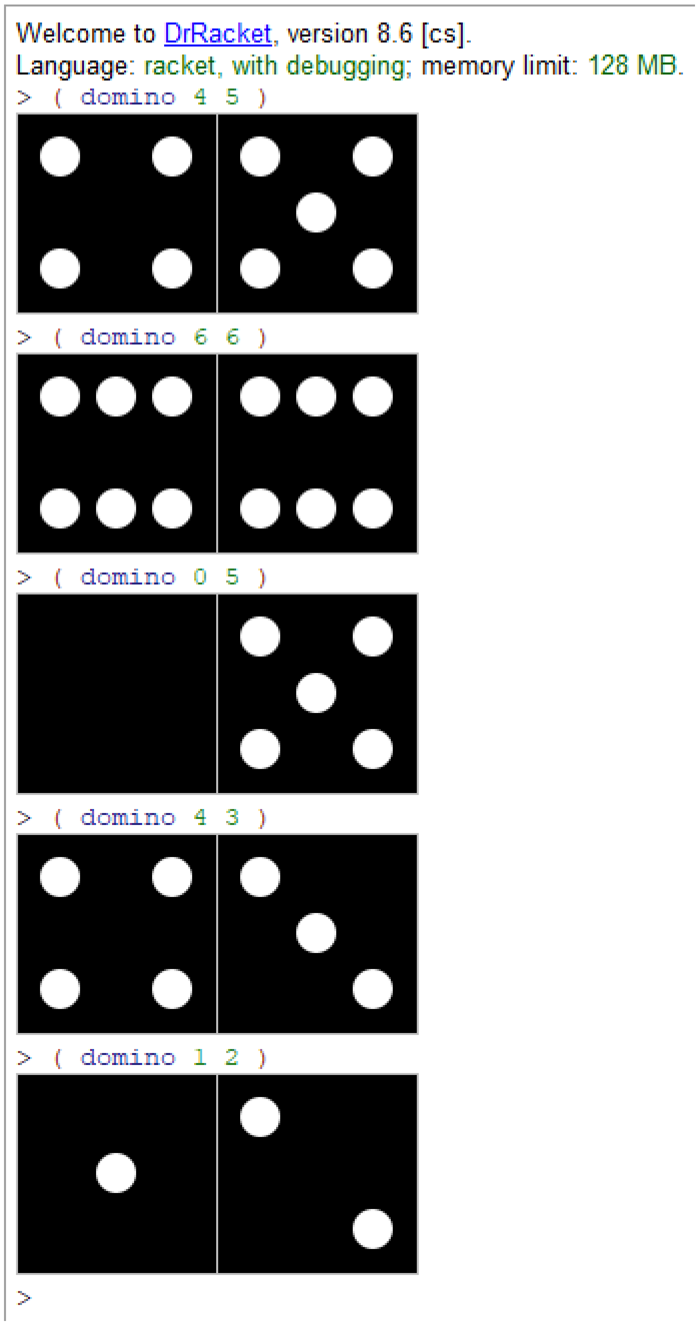
```
Language: racket, with debugging; memory limit: 128 MB.
> ( stella 200 20 'red 'lightblue )
```

```
> ( stella 100 5 'yellow 'blue )
```

```
> |
```

```
#lang racket
( require 2htdp/image )
( define ( stella size count color1 color2 )
   ( define step ( / size count))
   ( draw-stella size step 1 count color1 color2 empty-image )
   )

( define ( draw-stella size step start count color1 color2 img )
   ( define diameter ( * step start ))
   ( cond
      ( ( = start count )
        img
       )
      (
       ( > count start )
       ( if
         ( even? start )
                (
        draw-stella size step ( + start 1) count color1 color2 ( overlay img ( circle
diameter 'solid color1 ) )
        )
                       (
        draw-stella size step ( + start 1) count color1 color2 ( overlay img ( circle
diameter 'solid color2 ) )
        )
         )
       )
      )
   )
```

Task 6: Dominoes

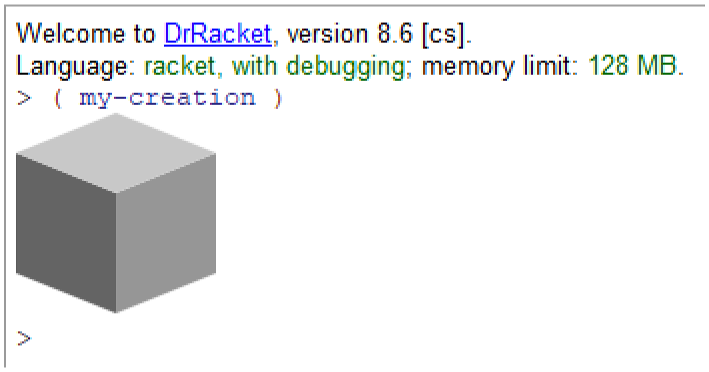


```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( domino 4 5 )
> ( domino 6 6 )
> ( domino 0 5 )
> ( domino 4 3 )
> ( domino 1 2 )
>
```

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

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

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



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

```
>
```

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

( define color1 ( color 200 200 200 ) )
( define color2 ( color 100 100 100 ) )
( define color3 ( color 150 150 150 ) )

( define (my-creation)
   ( overlay/offset
     ( polygon ( list
                 ( make-posn -50 0)
                 ( make-posn 0 20)
                 ( make-posn 50 0)
                 ( make-posn 0 -20)
                 )
               "solid"
               color1
               )
     0
     40
     ( overlay/offset
       ( polygon ( list
                   ( make-posn -50 0)
                   ( make-posn -50 60)
                   ( make-posn 0 80)
                   ( make-posn 0 20)
                   )
                 "solid"
                 color2
                 )
       50
       0
       ( polygon ( list
                   ( make-posn 0 0)
                   ( make-posn 0 60)
                   ( make-posn -50 80)
                   ( make-posn -50 20)
                   )
                 "solid"
                 color3
                 )
       )
```

```
        )
    )
```