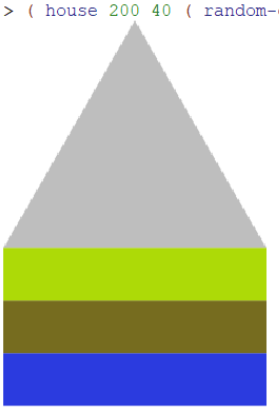
Sven Kappeler - Racket Assignment #2

Learning Abstract

This assignment is a continuation into using the Racket programming language. The purpose of this assignment is to get myself acquainted with using recursion in Racket

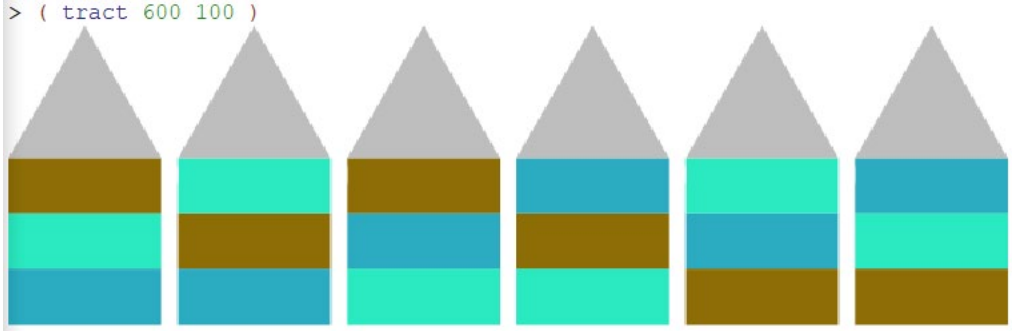
Task 1: Houses and Tracts

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



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

```
> ( tract 600 100 )
```



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

( define ( random-color )
   ( color ( rand ) ( rand ) ( rand ) )
 )

( define ( rand ) ( random 256 ) )

( define ( house width height f1-color f2-color f3-color )
   ( define f1 ( rectangle width height "solid" f1-color ) )
   ( define f2 ( rectangle width height "solid" f2-color ) )
   ( define f3 ( rectangle width height "solid" f3-color ) )
   ( define roof ( triangle width "solid" "Gray" ) )
   ( define complete-house ( above roof f3 f2 f1 ) )
   complete-house
)

( define ( tract width height )
   ( define house-width ( / ( - width 50 ) 6 ) )
   ( define house-height ( / height 3 ) )
   ( define color1 ( random-color ) )
   ( define color2 ( random-color ) )
   ( define color3 ( random-color ) )
   ( define space ( square 10 "solid" "White" ) )
   ( define house1 ( house house-width house-height color1 color2 color3 ) )
   ( define house2 ( house house-width house-height color1 color3 color2 ) )
   ( define house3 ( house house-width house-height color2 color1 color3 ) )
   ( define house4 ( house house-width house-height color2 color3 color1 ) )
   ( define house5 ( house house-width house-height color3 color1 color2 ) )
   ( define house6 ( house house-width house-height color3 color2 color1 ) )
   ( define complete-tract ( beside house1 space house2 space house3 space house4 space house5 space house6 ) )
   complete-tract
)
```

Task 2: Dice

```
> ( roll-dice )
1
> ( roll-dice )
4
> ( roll-dice )
1
> ( roll-dice )
5
> ( roll-dice )
6
```

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

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

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

```
> ( roll-for-lucky-pair )
(5 5)(4 5)(5 5)(6 6)(5 4)(4 4)(3 4)
> ( roll-for-lucky-pair )
(5 2)
> ( roll-for-lucky-pair )
(4 3)
> ( roll-for-lucky-pair )
(2 4)(2 1)(5 5)(5 3)(3 6)(5 3)(1 2)(6 3)(1 4)(1 4)(3 6)(2 1)(1 5)(2 2)(2 4)(2 6)(6 6)(5 2)
> ( roll-for-lucky-pair )
(5 1)(4 3)
> ( roll-for-lucky-pair )
(5 3)(2 5)
> ( roll-for-lucky-pair )
(4 3)
> ( roll-for-lucky-pair )
(6 4)(5 4)(4 5)(2 6)(6 3)(6 3)(1 3)(5 6)(2 6)(6 2)(2 3)(4 6)(5 5)(1 3)(6 4)(4 3)
> ( roll-for-lucky-pair )
(2 5)
> ( roll-for-lucky-pair )
(6 1)
```

```
#lang racket
 ( define ( roll-dice )
    ( + ( random 6 ) 1 )
 )

( define ( roll-for-1 )
    ( define result ( roll-dice ) )
    ( display result ) ( display " " )
    ( cond
       ( ( not ( eq? result 1 ) )
         ( roll-for-1 )
       )
    )
)

( define ( roll-for-1s )
    ( roll-for-1 )
    ( define result ( roll-dice ) )
    ( display result ) ( display " " )
    ( cond
       ( ( not ( eq? result 1 ) )
         ( roll-for-1s )
       )
    )
)

( define ( roll-for-even )
    ( define result ( roll-dice ) )
    ( display result ) ( display " " )
    ( cond
       ( ( not ( even? result ) )
       ( roll-for-even )
       )
    )
)

( define ( roll-for-odd )
    ( define result ( roll-dice ) )
    ( display result ) ( display " " )
    ( cond
       ( ( not ( odd? result ) )
       ( roll-for-odd )
       )
    )
)

( define ( roll-for-odd-even )
    ( roll-for-odd )
    ( define result ( roll-dice ) )
    ( display result ) ( display " " )
       ( cond
          ( ( not ( even? result ) )
          ( roll-for-odd-even )
          )
       )
)
```

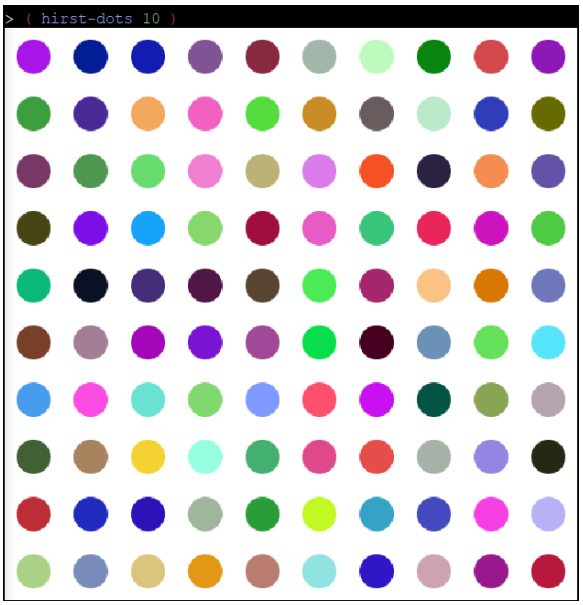
```
( define ( roll-for-odd-even-odd )
    ( roll-for-odd-even )
    ( define result ( roll-dice ) )
    ( display result ) ( display " " )
       ( cond
          ( ( not ( odd? result ) )
          ( roll-for-odd-even-odd )
          )
       )
)

( define ( roll-for-lucky-pair )
    ( define result0 ( roll-dice ) )
    ( define result1 ( roll-dice ) )
    ( display "(" ) ( display result0 ) ( display " " ) ( display result1 ) ( display ")" )
    ( cond
       ( ( not ( eq? ( + result0 result1) 7 ) )
       ( roll-for-lucky-pair )
       )
    )
)
```

Task 3: Number Sequence

```
> ( triangular 3 )
6
> ( triangular 5 )
15
> ( triangular 10 )
55
>
> ( sigma 3 )
4
> ( sigma 5 )
6
> ( sigma 10 )
18
> ( sigma 20 )
42
```

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

( define ( sigma n )
   ( sigma-function n n )
)

( define ( sigma-function x n )
   ( cond
      ( ( = n 1 ) 1 )
      ( else
        ( cond
         ( ( = ( remainder x n ) 0 )
           ( + ( sigma-function x ( - n 1 ) ) n ) )
           ( else
             ( sigma-function x ( - n 1 ) )
            )
         )
       )
    )
)
```

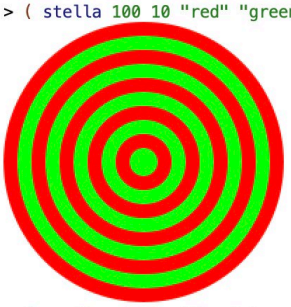
Task 4: Hirst Dots



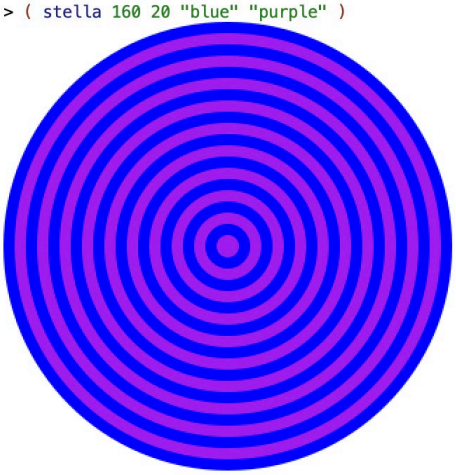

```
#lang racket
( require 2htdp/image )

( define ( random-color )
   ( color ( rand ) ( rand ) ( rand ) )
   )

( define ( rand ) ( random 255 ) )

( define ( dot )
      ( overlay ( circle 15 "solid" ( random-color ) )  ( square 50 "solid" "white" ) ) )


( define ( dot-row r )
   ( cond
      ( ( = r 1 )
        ( dot ) )
      ( else
        ( beside ( dot ) ( dot-row ( - r 1 ) ) )
      )
   )
)

( define ( dot-column r c )
   ( cond
      ( ( = c 1 )
        ( dot-row r )
      )
     ( else
       ( above ( dot-row r ) ( dot-column r ( - c 1 ) ) ) )
    )
)

( define ( hirst-dots n )
   ( dot-column n n ) )
```

Task 5: Frank Stella

```
> ( stella 100 10 "red" "green" )
```



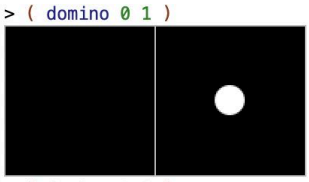
```
> ( stella 160 20 "blue" "purple" )
```



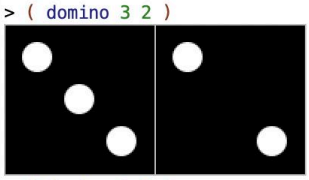
```
#lang racket

( require 2htdp/image )

( define ( rgb-value ) ( random 256 ) )
( define ( random-color ) ( rgb-value ) ( rgb-value ) ( rgb-value ) )

( define ( stella radius count color1 color2 )
   ( define delta ( / radius count ) )
   ( paint-nested-circles-two 1 count delta color1 color2 )
)

( define ( paint-nested-circles-two from to delta color1 color2 )
   ( define radius-length ( * from delta ) )
   ( cond
      ( ( = from to )
        ( if ( even? from )
             ( circle radius-length "solid" color1 )
             ( circle radius-length "solid" color2 )
        )
      )
      ( ( < from to )
        ( if ( even? from )
             ( overlay
               ( circle radius-length "solid" color1 )
               ( paint-nested-circles-two ( + from 1 ) to delta color1 color2 )
             )
             ( overlay
               ( circle radius-length "solid" color2 )
               ( paint-nested-circles-two ( + from 1 ) to delta color1 color2 )
             )
        )
      )
    )
)
```

Task 6: Dominos

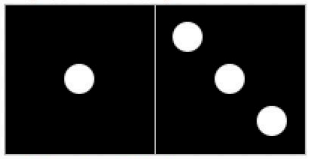
```
> ( domino 0 1 )
```



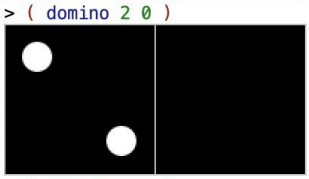
```
> ( domino 3 2 )
```



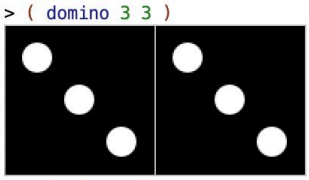
```
> ( domino 1 3 )
```



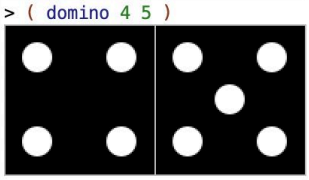
```
> ( domino 2 0 )
```



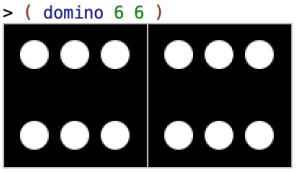
```
> ( domino 3 3 )
```



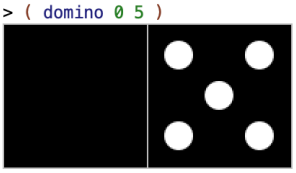
```
> ( domino 4 5 )
```



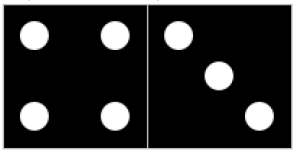
```
> ( domino 6 6 )
```



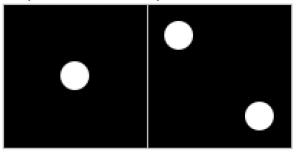
```
> ( domino 0 5 )
```



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



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

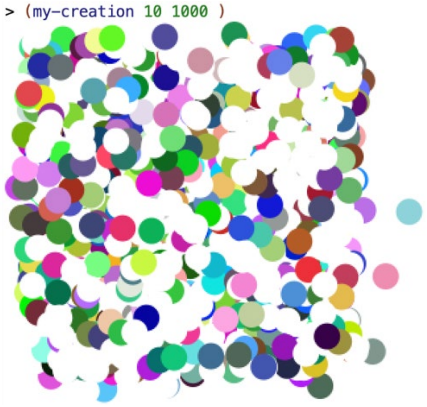


```
>
```

```
#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 )  nd d
            ( overlay/offset ( pip ) d nd basic-tile2 )
   )
)

( 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 6: My Creation

```
> (my-creation 10 1000 )
```



```
#lang racket

( require 2htdp/image )

( define ( random-color )
   ( color ( rand ) ( rand ) ( rand ) )
)
( define ( rand ) ( random 255 ) )

( define ( my-creation radius count )
   ( paint-my-creation 1 count radius )
)

( define ( paint-my-creation from to radius )
   ( cond
      ( ( = from to )
         ( circle radius "solid" ( random-color ) ) )
      )
      ( ( < from to )
         ( if ( even? from )
              ( overlay/offset ( circle radius "solid" ( random-color ) ) ( rand ) ( rand )
                               ( paint-my-creation ( + from 1 ) to radius )
              )
              ( overlay/offset ( circle radius "solid" "white" ) ( rand ) ( rand )
                               ( paint-my-creation ( + from 1 ) to radius )
              )
         )
      )
   )
)
```