

Racket Programing Assignment #2: Racket Functions and Recursion

Learning Abstract

In this assignment, mostly the image creating library (2htdp/image) of racket is used to produce the creation of all the images. In addition, the recursive function of racket is also used to make the recursion in the assignment.

For Task.1, we were asked to create house images using the image function of the racket and the tract of houses.

For Task.2, we learnt about the basic concept of making a die rolling program and made the interesting plays with the die

For Task.3, using the recursion function the square, the cube, the triangular and the sigma of the numbers were calculated. The function to produce the sequence of each function were also included in the task.

For Task.4, the random colored hirst dots were created by using the recursion and image library of the racket.

For Task.5, the interesting stella of triangle shapes were created by using the image library and recursive methods.

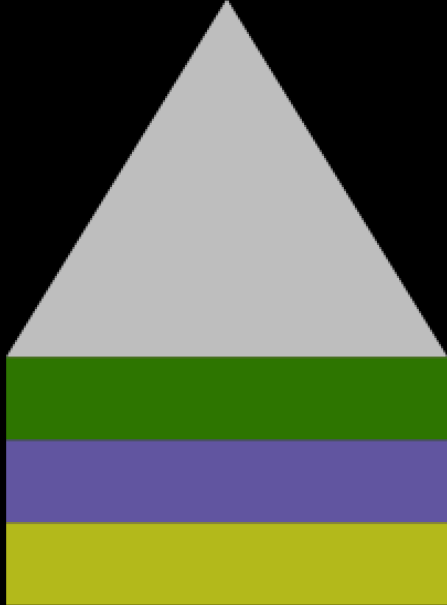
For Task.6, the dominos were produced by extending the preliminary codes that the instructor provided.

For Task.7, using the image library of the racket, I created the petals of the flowers using the eclipse shapes and the core using the overlaying the circles.

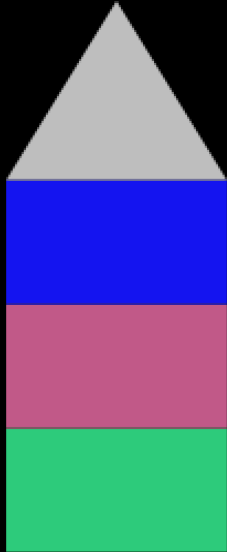
Task.1: Colorful Permutations of Tract Houses

Demo for house

```
Welcome to DrRacket, version 8.6 [cs].  
Language: racket, with debugging; memory limit: 128 MB.  
> (house 200 40)
```



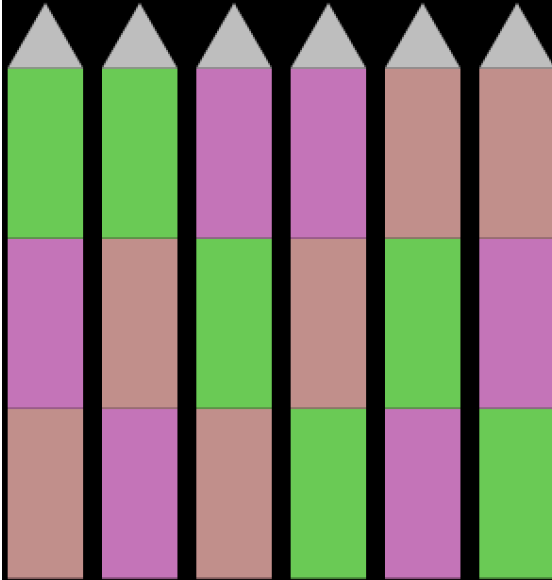
```
> (house 100 60)
```



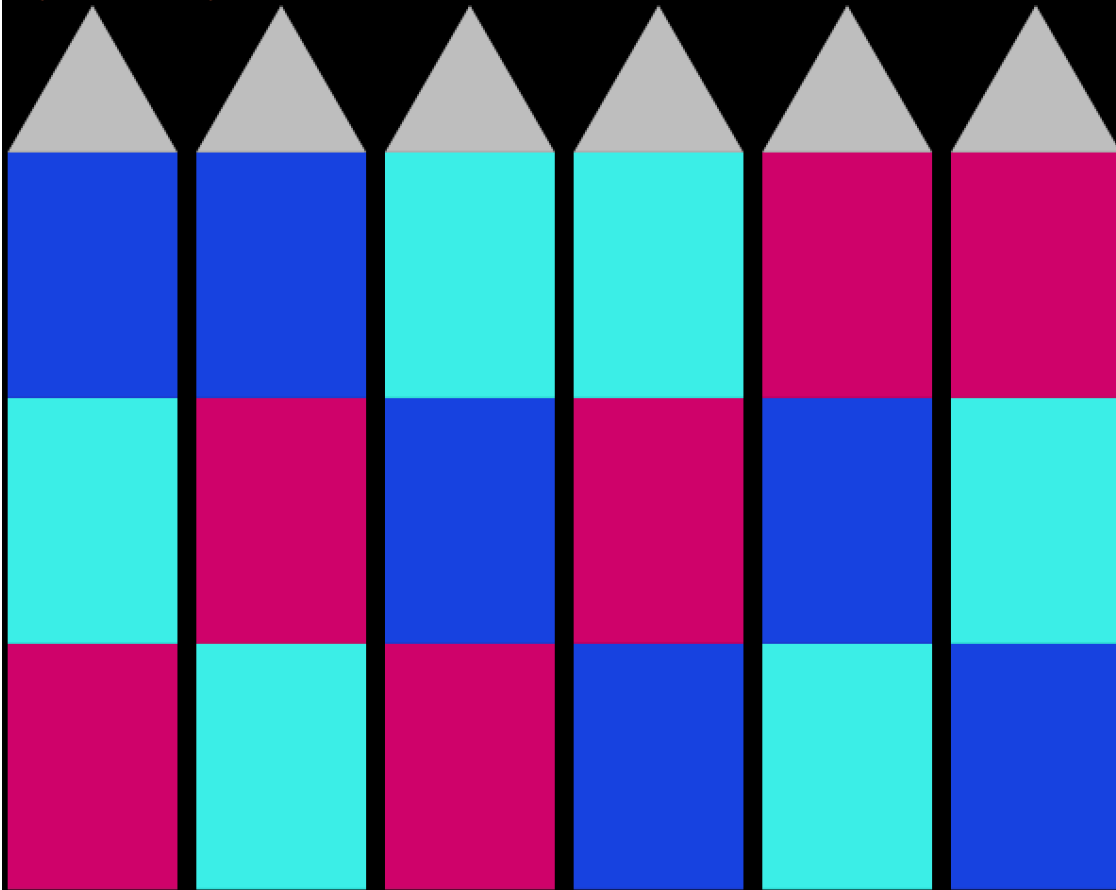
```
>
```

Demo for tract

```
> (tract 40 90)
```



```
> (tract 90 130)
```



The code for Task.1

```
1 #lang racket
2
3 (require 2htdp/image)
4
5
6 (define (random-color) (color(rgb-value)(rgb-value)(rgb-value)))
7 (define (rgb-value) (random 256))
8
9 (define (house-floor width height)
10   (rectangle width height "solid" (random-color)))
11 )
12
13
14 (define (house-roof side)
15   (triangle side "solid" "gray"))
16 )
17
18
19 (define (house width-of-house height-of-house)
20   (define first-floor (house-floor width-of-house height-of-house))
21   (define second-floor (house-floor width-of-house height-of-house))
22   (define third-floor (house-floor width-of-house height-of-house))
23   (define actual-house-roof (house-roof width-of-house))
24   (define house-layout (above actual-house-roof first-floor second-floor third-floor))
25   house-layout
26 )
27
28
29 (define (first-floor width height) (rectangle width height "solid" (random-color)))
30 (define (second-floor width height) (rectangle width height "solid" (random-color)))
31 (define (third-floor width height) (rectangle width height "solid" (random-color)))
32 (define gap (square 10 "solid" "black"))
33
34 (define (tract width height)
35   (define tract-first-floor (first-floor width height))
36   (define tract-second-floor (second-floor width height))
37   (define tract-third-floor (third-floor width height))
38   (define roof-of-house (house-roof width))
39   (define house1 (above roof-of-house tract-first-floor tract-second-floor tract-third-floor))
40   (define house2 (above roof-of-house tract-first-floor tract-third-floor tract-second-floor))
41   (define house3 (above roof-of-house tract-second-floor tract-first-floor tract-third-floor))
42   (define house4 (above roof-of-house tract-second-floor tract-third-floor tract-first-floor))
43   (define house5 (above roof-of-house tract-third-floor tract-first-floor tract-second-floor))
44   (define house6 (above roof-of-house tract-third-floor tract-second-floor tract-first-floor))
45   (define tract-layout (beside house1 gap house2 gap house3 gap house4 gap house5 gap house6))
46   tract-layout
47 )
```

Task.2: Dice











Demo

```

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

```

```

Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (roll-for-odd-even-odd)
2 5 3 4 4 4 1   application: not a procedure;
expected a procedure that can be applied to arguments
given: #<void>
> (roll-for-odd-even-odd)
1 2 2 5   application: not a procedure;
expected a procedure that can be applied to arguments
given: #<void>
> (roll-for-odd-even-odd)
5 5 5 5 5 2 5   application: not a procedure;
expected a procedure that can be applied to arguments
given: #<void>
> (roll-for-odd-even-odd)
4 6 1 6 3   application: not a procedure;
expected a procedure that can be applied to arguments
given: #<void>
> (roll-for-odd-even-odd)
1 6 1   application: not a procedure;
expected a procedure that can be applied to arguments
given: #<void>

```

```

Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (roll-for-the-lucky-pair)
3 6 3 3
> (roll-for-the-lucky-pair)
1 6
> (roll-for-the-lucky-pair)
6 4 3 4
> (roll-for-the-lucky-pair)
5 5
> (roll-for-the-lucky-pair)
1 4 1 4 5 6
> (roll-for-the-lucky-pair)
3 1 3 2 4 2 2 3 2 3 1 6
> (roll-for-the-lucky-pair)
1 6
> (roll-for-the-lucky-pair)
6 2 5 4 5 6
> (roll-for-the-lucky-pair)
3 6 3 4
> (roll-for-the-lucky-pair)
2 6 3 6 6 2 1 4 4 6 1 2 2 6 4 3

```

The code for Task.2

```

1 #lang racket
2
3
4 ;Function to roll a die
5 (define (roll-die)
6   (+ (random 6) 1)
7 )
8
9
10 ;Function to roll for 1
11 (define (roll-for-1)
12   (define outcomes (roll-die))
13   (display outcomes) (display " ")
14
15   (cond
16     ((not (eq? outcomes 1))
17      (roll-for-1))
18   )
19 )
20
21
22
23 ;Function to roll for 11
24 (define (roll-for-11)
25   (roll-for-1)
26   (define outcomes (roll-die))
27   (display outcomes) (display " ")
28
29   (cond
30     ((not (eq? outcomes 1))
31      (roll-for-11))
32   )
33 )
34
35

```

```

1  #lang racket
36
37 ;Function to roll for even numbers
38
39 (define (roll-for-even)
40   (define outcomes (roll-die))
41   (display outcomes) (display " ")
42   (cond
43     ((not(even? outcomes))
44      (roll-for-even)
45     ))
46   )
47
48 ;Function to roll for odd numbers
49
50 (define (roll-for-odd)
51   (define outcomes (roll-die))
52   (display outcomes) (display " ")
53   (cond
54     ((not(odd? outcomes))
55      (roll-for-odd)
56     ))
57   )
58
59
60
61 ;Function to roll for odd even odd
62 (define (roll-for-odd-even-odd)
63   ((roll-for-odd)(roll-for-even)(roll-for-odd))
64 )
65
66
67
68
69 ;Function to roll two dice for a lucky pair
70
71 (define (roll-for-the-lucky-pair)
72   (define outcomes (roll-die))
73   (define outcomes0 (roll-die))
74   (display outcomes) (display " ")
75   (display outcomes0) (display " ")
76
77   (define numbers (+ outcomes outcomes0))
78   (define lucky-number-7 (= numbers 7))
79   (define lucky-number-11 (= numbers 11))
80   (define same-number (= outcomes outcomes0))
81
82   (cond
83     ((not (or lucky-number-7 lucky-number-11 same-number))
84      (roll-for-the-lucky-pair))
85   )
86 )
87

```

Task.3: Number Sequences

Preliminary 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
>
```

Triangular demo

```
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
>
```

The code for Task.3

```

1  #lang racket
2
3  ;; Square numbers
4  (define (square n)
5    (* n n)
6  )
7
8  ;; Cube numbers
9  (define (cube n)
10   (* n n n)
11 )
12
13 ;; Square Sequences
14 (define (sequence square n)
15   (cond

```

```

15     ((= n 1)
16      (display (square 1)) (display " "))
17   )
18   (else
19    (sequence square (- n 1))
20    (display (square n)) (display " "))
21   )
22 )
23
24
25
26 ;; Cube Sequences
27 (define (sequence1 cube n)
28   (cond
29     ((= n 1)
30      (display (cube 1)) (display " "))
31   )
32   (else
33    (sequence1 cube (- n 1))
34    (display (cube n)) (display " "))
35   )
36 )
37
38
39 ;; Triangular numbers
40
41 (define (triangular n)
42   (cond
43     ((= n 1) 1)
44     ((= n 2) 3)
45     ((> n 2)
46      (+ (triangular (- n 1)) n)
47     )
48   )
49 )
50
51 ;; Triangular sequences
52
53 (define (triangular-sequence triangular n)
54   (cond
55     ((= n 1)
56      (display (triangular 1)) (display " "))
57   )
58   (else
59    (triangular-sequence triangular (- n 1))
60    (display (triangular n)) (display " ")
61   )
62 )
63 )
64

```

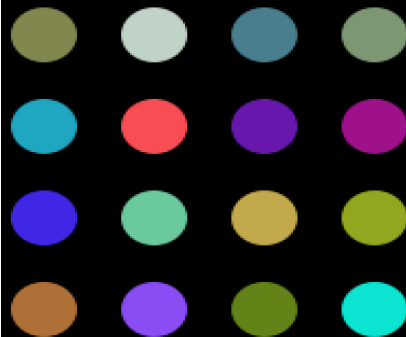
Task.4: Hirst Dots

Demo

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



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



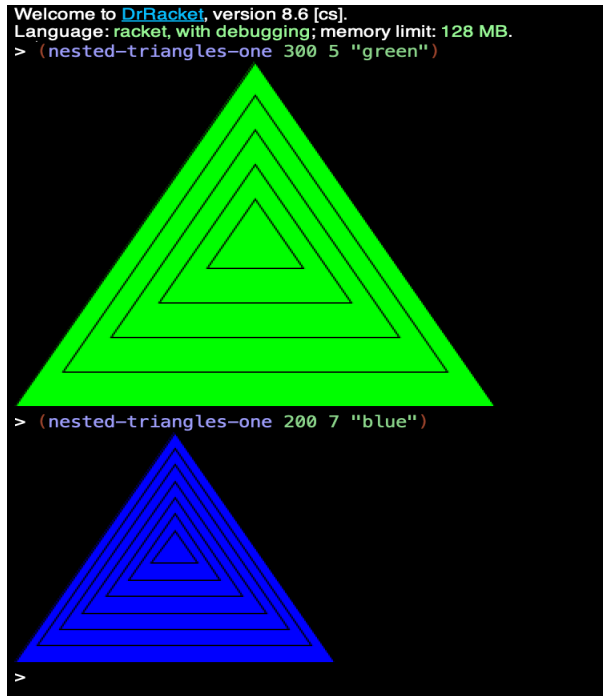
```
>
```

Code for Task.4

```
1  #lang racket
2
3  (require 2htdp/image)
4
5  ;; Generate random colors
6
7  (define (random-color) (color(rgb-value)(rgb-value)(rgb-value)))
8  (define (rgb-value) (random 256))
9
10 ;; Create the dots
11 (define diameter 30)
12 (define radius (/ diameter 2))
13 (define gap (square 20 "solid" "black"))
14 (define (dot) (circle radius "solid" (random-color)))
15
16 ;; Generate the row-of-dots
17 (define (row-of-dots n)
18   (cond
19     ((= n 0)
20      empty-image)
21     ((> n 0)
22      (beside (row-of-dots (- n 1)) (dot) gap))
23     )
24   )
25
26 ;; Generate the rectangle-of-dots
27 (define (rectangle-of-dots r r1)
28   (cond
29     ((= r 0)
30      empty-image)
31     ((> r 0)
32      (above (rectangle-of-dots (- r 1) r1) gap (row-of-dots r1)))
33     )
34   )
35
36 ;; Create the function to produce hirst dots
37 (define (hirst-dots n)
38   (rectangle-of-dots n n)
39 )
```

Task.5: Channeling Frank Stella

Demo


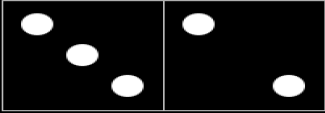
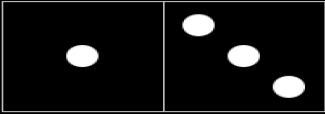
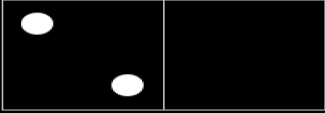
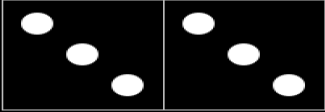
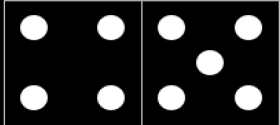
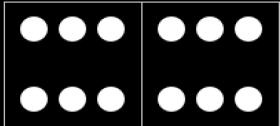
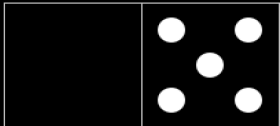
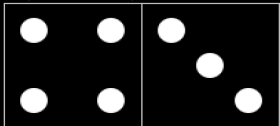
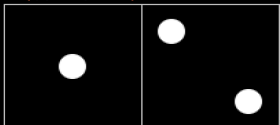


Code for Task.5

```
1 #lang racket
2
3 ( require 2htdp/image )
4
5 ( define ( nested-triangles-one side count color )
6   ( define unit ( / side count ) )
7   ( paint-nested-triangles-one 1 count unit color )
8 )
9
10 ( define ( paint-nested-triangles-one from to unit color)
11   ( define side-length ( * from unit ) )
12   ( cond
13     ( ( = from to )
14       ( framed-triangle side-length color )
15     )
16     ( ( < from to )
17       ( overlay
18         ( framed-triangle side-length color )
19         ( paint-nested-triangles-one ( + from 1 ) to unit color )
20       )
21     )
22   )
23 )
24 ( define ( framed-triangle side-length color )
25   ( overlay
26     ( triangle side-length "outline" "black" )
27     ( triangle side-length "solid" color )
28   )
29 )
```

Task.6: Dominos

Final Demo

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

> (domino 3 2)

> (domino 1 3)

> (domino 2 0)

> (domino 3 3)

> |
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)

> |
```

Collected Code for Task.6

```
1 #lang racket
2
3 ;-----
4 ; Requirements
5 ;
6 ; - Just the image library from Version 2 of "How to Design Programs"
7 ;
8 ( require 2htdp/image )
9 ;-----
10 ; Problem parameters
11 ;
12 ; - Variables to denote the side of a tile and the dimensions of a pip
13 ;
14 ( define side-of-tile 100 )
15 ( define diameter-of-pip ( * side-of-tile 0.2 ) )
16 ( define radius-of-pip ( / diameter-of-pip 2 ) )
17 ;-----
18 ; Numbers used for offsetting pips from the center of a tile
19 ;
20 ; - d and nd are used as offsets in the overlay/offset function applications
21 ;
22 ( define d ( * diameter-of-pip 1.4 ) )
23 ( define nd ( * -1 d ) )
24 ;-----
25 ; The blank tile and the pip generator
26 ;
27 ; - Bind one variable to a blank tile and another to a pip
28 ;
29 ( define blank-tile ( square side-of-tile "solid" "black" ) )
30 ( define ( pip ) ( circle radius-of-pip "solid" "white" ) )
31 ;-----
32 ; The basic tiles
33 ;
34 ; - Bind one variable to each of the basic tiles
35 ;
36 ( define basic-tile1 ( overlay ( pip ) blank-tile ) )
37 ( define basic-tile2
38   ( overlay/offset ( pip ) d d
39     ( overlay/offset ( pip ) nd nd blank-tile)
40   )
41 )
42
43 (define basic-tile3 (overlay (pip) basic-tile2))
44 (define basic-tile4
45   (overlay/offset ( pip ) d d
46     (overlay/offset ( pip ) d nd
47       (overlay/offset ( pip ) nd d
48         (overlay/offset ( pip ) nd nd blank-tile)
49       )
50     )
51   )
52 )
```

```

49 )
50 )
51 )
52 )
53
54 (define basic-tile5 (overlay ( pip ) basic-tile4))
55 (define basic-tile6
56   (overlay/offset ( pip ) d d
57     (overlay/offset ( pip ) nd nd
58       (overlay/offset ( pip ) d nd
59         (overlay/offset ( pip ) nd d
60           (overlay/offset ( pip ) 0 d
61             (overlay/offset ( pip ) 0 nd blank-tile)
62           ))))
63
64 ;-----
65 ; The framed framed tiles
66 ;
67 ; - Bind one variable to each of the six framed tiles
68 ;
69 ( define frame ( square side-of-tile "outline" "gray" ) )
70 ( define tile0 ( overlay frame blank-tile ) )
71 ( define tile1 ( overlay frame basic-tile1 ) )
72 ( define tile2 ( overlay frame basic-tile2 ) )
73 ( define tile3 ( overlay frame basic-tile3 ) )
74 ( define tile4 ( overlay frame basic-tile4 ) )
75 ( define tile5 ( overlay frame basic-tile5 ) )
76 ( define tile6 ( overlay frame basic-tile6 ) )
77 ;-----
78 ; Domino generator
79 ;
80 ; - Funtion to generate a domino
81 ;
82 ( define ( domino a b )
83   ( beside ( tile a ) ( tile b ) )
84 )
85 ( define ( tile x )
86   ( cond
87     ( ( = x 0 ) tile0 )
88     ( ( = x 1 ) tile1 )
89     ( ( = x 2 ) tile2 )
90     ( ( = x 3 ) tile3 )
91     ( ( = x 4 ) tile4 )
92     ( ( = x 5 ) tile5 )
93     ( ( = x 6 ) tile6 )
94   )
95 )

```

Task.7: Creation

Creation (image)



Code for Task.7

```
1 #lang racket
2
3 (require 2htdp/image)
4
5
6 (let ([petal (put-pinhole
7           30 30
8           (ellipse 175 60 "solid" "yellow"))])
9   (clear-pinhole
10    (overlay/pinhole
11      (overlay
12        (circle 20 "solid" (make-color 90 90 255))
13        (circle 26 "solid" (make-color 100 100 255))
14        (circle 32 "solid" (make-color 150 150 255))
15        (circle 38 "solid" (make-color 200 200 255))
16        (circle 44 "solid" (make-color 250 250 255)))
17      (circle 30 "solid" "purple")
18      (rotate (* 60 0) petal)
19      (rotate (* 60 1) petal)
20      (rotate (* 60 2) petal)
21      (rotate (* 60 3) petal)
22      (rotate (* 60 4) petal)
23      (rotate (* 60 5) petal)
24    )))
25
```