

JD Vargas

Racket Programming Assignment #2: Racket Functions and Recursion

Learning Abstract:

During this assignment I learned how to use recursion to generate different types of images from the 2htdp/image library and put them together.

Task 1: Colorful Permutations of Tract Houses



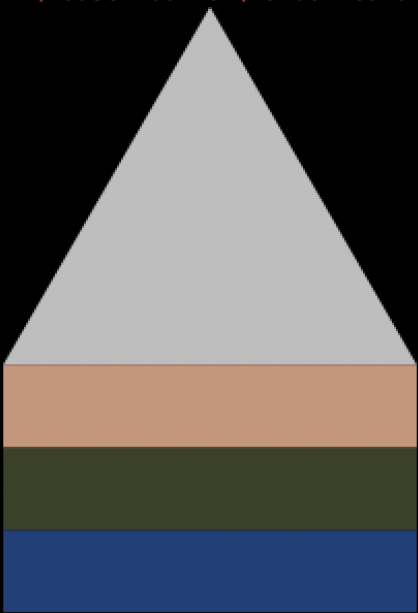
Demo/code for house:

```
Untitled (define ...) Macro Stepper # Run Stop

1 #lang racket
2 (require 2htdp/image)
3 (define (vari) (random 256))
4 (define (random-color) (color (vari) (vari) (vari)))
5 (define (rec1 width height c1) (rectangle width height 'solid c1))
6 (define (rec2 width height c2) (rectangle width height 'solid c2))
7 (define (rec3 width height c3) (rectangle width height 'solid c3))
8 (define (tri width) (triangle width "solid" "grey"))
9
10 (define (house width height c1 c2 c3)
11   (define actual-house
12     (above (tri width)
13             (rec1 width height c1)
14             (rec2 width height c2)
15             (rec3 width height c3)))
16   actual-house)
17
```

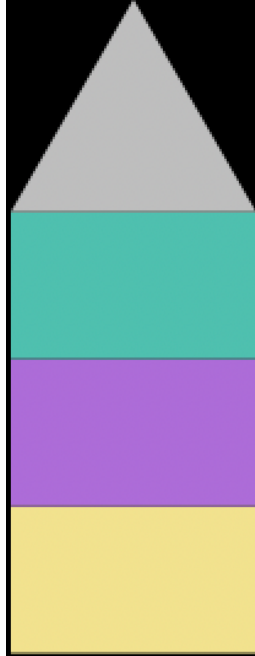
Welcome to [DrRacket](#), version 8.6 [cs].
Language: **racket**, with **debugging**; memory limit: 128 MB.

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



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

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



```
>
```

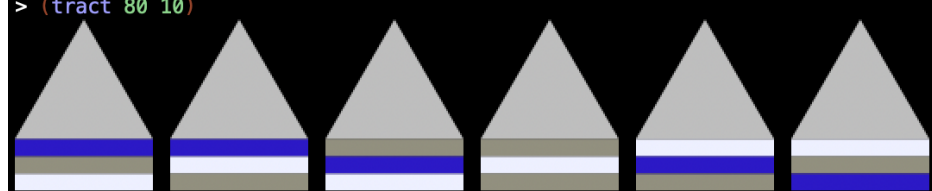
Demo/code for tract:

```
house demo.rkt (define ...) Debug Macro Stepper Run Stop

1 #lang racket
2 (require 2htdp/image)
3 (define (vari) (random 256))
4 (define (random-color) (color (vari) (vari) (vari)))
5 (define gap (square 10 "solid" "black"))
6 (define (1f width height) (rectangle width height 'solid (random-color)))
7 (define (2f width height) (rectangle width height 'solid (random-color)))
8 (define (3f width height) (rectangle width height 'solid (random-color)))
9 (define (top width) (triangle width "solid" "grey"))
10
11 (define (tract width height)
12   (define f1 (1f width height))
13   (define f2 (2f width height))
14   (define f3 (3f width height))
15   (define roof (top width))
16   (define house1 (above roof f1 f2 f3))
17   (define house2 (above roof f1 f3 f2))
18   (define house3 (above roof f2 f1 f3))
19   (define house4 (above roof f2 f3 f2))
20   (define house5 (above roof f3 f1 f2))
21   (define house6 (above roof f3 f2 f1))
22
23   (define tract-demo (beside house1 gap house2 gap house3 gap house4 gap
24                             house5 gap house6) ) tract-demo
25 )
```

Welcome to [DrRacket](#), version 8.6 [cs].
 Language: racket, with debugging; memory limit: 128 MB.

```
> (tract 80 10)
```



```
> |
```

Task 2: Dice

Code:

```
Untitled (define ...) Debug Macro Stepper Run Stop
1 #lang racket
2 (define (roll-die) ( + (random 6) 1) )
3
4 (define (roll-for-1) (define roll (roll-die) )
5   (display roll) (display " " )
6   (cond
7     ((not (eq? roll 1) ) (roll-for-1) ) ) )
8
9 (define (roll-for-11) (roll-for-1) (define roll (roll-die) )
10  (display roll)
11  (display " ")
12  (cond
13    ((not ( eq? roll 1 ) ) (roll-for-11) ) ) )
14
15 (define (roll-for-even) (roll-die) (define roll (roll-die) )
16  (display roll) (display " " )
17  (cond
18    ((not (eq? (remainder roll 2) 0) ) (roll-for-even) ) ) )
19
20 (define (roll-for-odd) (roll-die) (define roll (roll-die) )
21  (display roll) (display " " )
22  (cond
23    ((not (eq? (remainder roll 2) 1) ) (roll-for-odd) ) ) )
24
25 (define (roll-for-odd-even-odd) (roll-for-odd)(roll-for-even)(roll-for-odd))
26
27 (define (roll-two-dice-for-a-lucky-pair) (define r1 (roll-die))
28   (define r2 (roll-die))
29   (display "(") (display r1) (display " ") (display r2) (display ")")
30   (display " ")
31   (define lucky ( + r1 r2))
32   (define lucky7 (= lucky 7))
33   (define lucky11 (= lucky 11))
34   (define doubles (= r1 r2))
35   (cond
36     ((not(or lucky7 lucky11 doubles)) (roll-two-dice-for-a-lucky-pair))))
37
38
39
40
```

Demo:

```
Untitled ▾ (define ...) ▾ ➡ 📄 🔍 🏁 Debug 🐛 ▶ Macro Stepper # ▶ Run ▶ Stop 🛑

1 | #lang racket

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

Untitled (define ...) [Icons] Debug [Icons] Macro Stepper [Icons] Run [Icons] Stop [Icons]

1 | #lang racket

1 4 5 3 5 1 2 2 1 5 2 4 4 5 6 5 1 3 3 6 3 6 3 3 5 2 6 1 5 6 4 3 6 5 1 1
> (roll-for-11)
6 3 3 6 3 4 1 5 2 6 1 1
> (roll-for-11)
1 4 3 6 5 3 6 1 2 5 1 6 2 1 2 5 6 1 3 6 2 6 1 4 2 6 4 2 3 1 4 5 4 6 6 3 1 6 2 3 2
6 1 1
> (roll-for-11)
2 3 1 2 4 2 3 4 5 2 1 5 2 3 5 3 6 6 3 5 4 4 1 4 4 5 6 3 6 5 5 2 4 6 5 3 2 6 1 2 2
5 5 1 3 1 6 2 3 1 5 4 2 3 6 4 5 4 2 2 6 2 4 5 2 3 3 3 5 2 4 5 1 3 4 1 3 1 5 2 1 2
3 4 3 4 3 3 3 5 3 2 6 2 3 5 4 6 4 3 1 6 1 2 6 2 2 4 4 6 6 1 2 4 6 5 6 6 4 2 4 6 2
6 2 4 5 6 1 2 3 2 5 5 4 1 1
> (roll-for-11)
3 3 3 1 2 1 4 5 1 1
> (roll-for-odd-even-odd)
6 5 5 6 1
> (roll-for-odd-even-odd)
6 6 2 1 2 4 1
> (roll-for-odd-even-odd)
3 2 6 1
> (roll-for-odd-even-odd)
1 4 1
> (roll-for-odd-even-odd)
6 2 1 6 3
> (roll-two-dice-for-a-lucky-pair)
(4 4)
> (roll-two-dice-for-a-lucky-pair)
(6 6)
> (roll-two-dice-for-a-lucky-pair)
(5 6)
> (roll-two-dice-for-a-lucky-pair)
(2 4) (4 2) (2 4) (4 3)
> (roll-two-dice-for-a-lucky-pair)
(1 1)
> (roll-two-dice-for-a-lucky-pair)
(2 5)
> (roll-two-dice-for-a-lucky-pair)
(1 2) (5 4) (2 3) (6 6)
> (roll-two-dice-for-a-lucky-pair)
(1 6)
> (roll-two-dice-for-a-lucky-pair)
(3 1) (5 1) (4 6) (6 4) (4 2) (5 1) (1 5) (2 1) (2 1) (3 1) (1 2) (2 1) (3 3)
> (roll-two-dice-for-a-lucky-pair)
(1 5) (4 4)
>

Task 3: Number Sequences



Code:

```
Untitled (define ...) [Icons] Debug [Icons] Macro Stepper [Icons] Run [Icons] Stop [Icons]
1 #lang racket
2 (define (square n)
3   (* n n)
4 )
5 (define (cube n)
6   (* n n n)
7 )
8
9
10 (define (triangular n)
11   (cond
12     ((= n 1) 1)
13     ((= n 2) 3)
14     ((> n 2)
15      (+ (triangular (- n 1)) n)))
16 )
17 (define (sequence triangular n)
18   (cond
19     ((= n 1)
20      (display (triangular 1)) (display " "))
21     )
22     (else
23      (sequence triangular (- n 1))
24      (display (triangular n)) (display " "))
25     )
26   )
27 )
28
29 (define (sigma n) (math n n))
30 (define (math n orig)
31   (cond
32     ((= n 1) 1) ((= (modulo orig n) 0) (+ n (math (- n 1) orig)))
33     (else (math (- n 1) orig)))
34 )
35 (define (sequencee sigma n)
36   (cond
37     ((= n 1)
38      (display (sigma 1)) (display " "))
39     )
40     (else
41      (sequence sigma (- n 1))
42      (display (sigma n)) (display " ")))
43 )
```

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 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
> (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
> |
```

Task 4: Hirst Dots



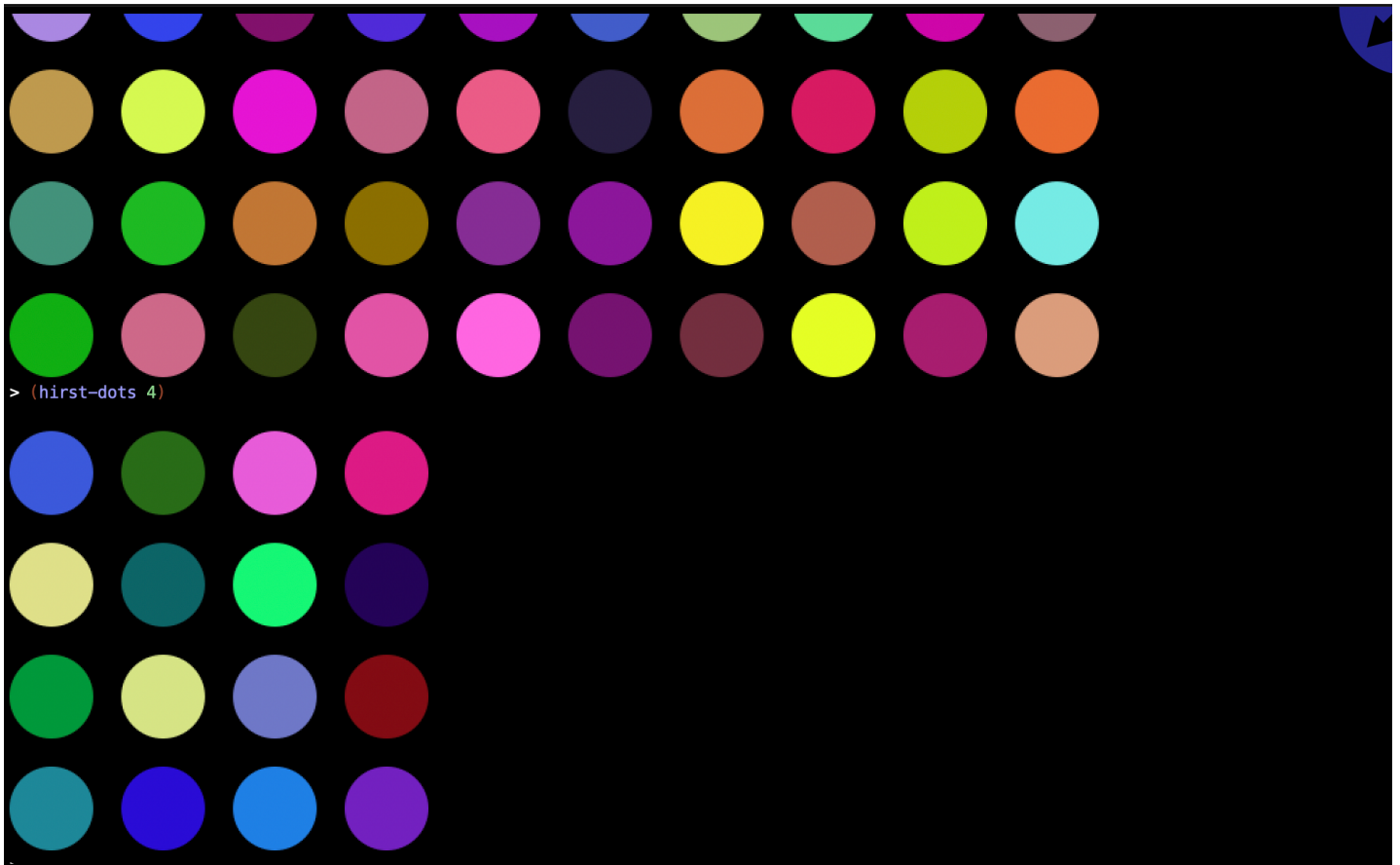
Code:

```
Untitled ▾ (define ...) ▾ ➡ 📄 Check Syntax 🔍 ✓ Debug 🐞 ▶ Macro Stepper # ▶ Run ▶ Stop 🛑

1 #lang racket
2 (require 2htdp/image)
3 (define (rgb-value) (random 256))
4 (define (random-color) (color (rgb-value) (rgb-value) (rgb-value)))
5 (define gap (square 20 "solid" "black"))
6 (define diameter 30)
7 (define (dot diameter color)(circle diameter 'solid (random-color)))
8
9 (define (row r)
10   (cond
11     ((= r 0) empty-image)((> r 0)(beside (row (- r 1))(dot diameter color)
12   gap))))
13
14 (define (column m n )
15   (cond
16     ((= m 0) empty-image)((> m 0)(above (column (- m 1) n) gap (row n))))
17
18 (define (hirst-dots r)(column r r))
19
20
21
22
```

Demo:





Task 5: Channelling Frank Stella



Code:

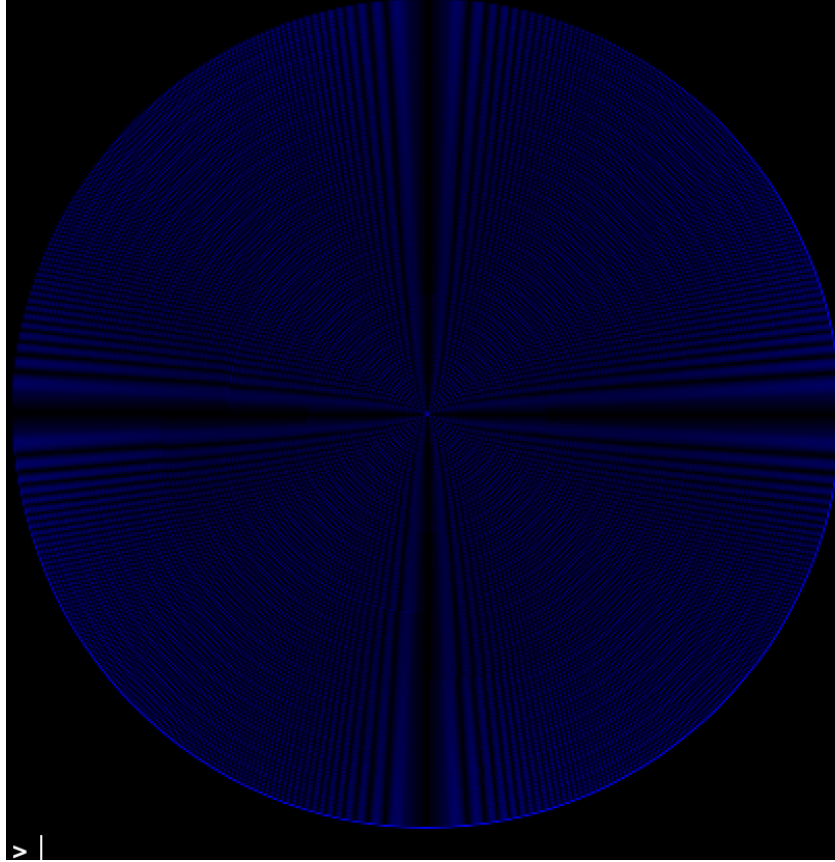
```

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

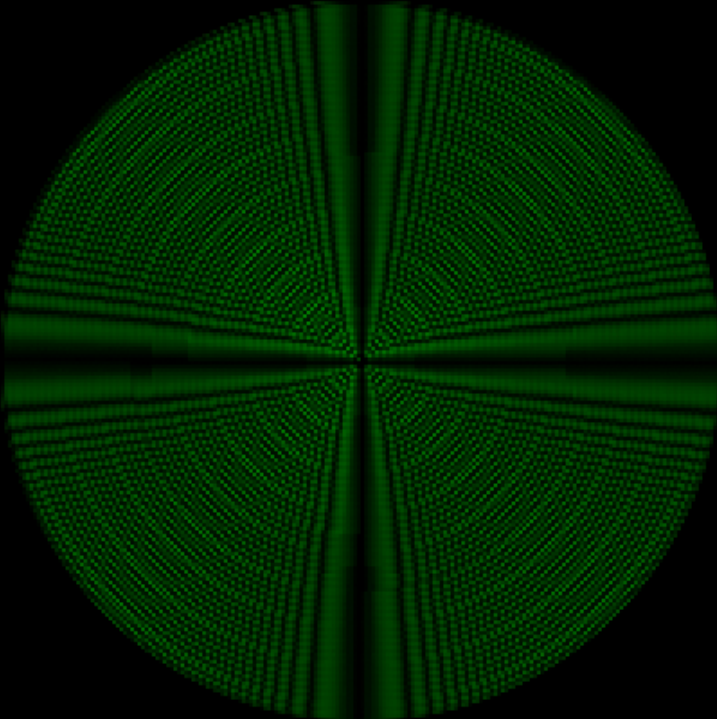
```

Demo:

```
welcome to DrRacket, version 8.6 [cs].  
Language: racket, with debugging; memory limit: 128 MB.  
> (nested-squares-one 200 199 "blue")
```



```
> (nested-squares-one 100 101 "green")
```



Task 6: Dominos



Code:

```
Untitled (define ...) [Icons] Debug [Icons] Macro Stepper [Icons] Run [Icons] Stop [Icons]
1 #lang racket
2 ( require 2htdp/image )
3 ;-----
4 ; Problem parameters
5 ;
6 ; - Variables to denote the side of a tile and the dimensions of a pip
7 ;
8 ( define side-of-tile 100 )
9 ( define diameter-of-pip ( * side-of-tile 0.2 ) )
10 ( define radius-of-pip ( / diameter-of-pip 2 ) )
11 ;-----
12 ; Numbers used for offsetting pips from the center of a tile
13 ;
14 ; - d and nd are used as offsets in the overlay/offset function applications
15 ;
16 ( define d ( * diameter-of-pip 1.4 ) )
17 ( define nd ( * -1 d ) )
18 ;-----
19 ; The blank tile and the pip generator
20 ;
21 ; - Bind one variable to a blank tile and another to a pip
22 ;
23 ( define blank-tile ( square side-of-tile "solid" "black" ) )
24 ( define ( pip ) ( circle radius-of-pip "solid" "white" ) )
25 ;-----
26 ; The basic tiles
27 ;
28 ; - Bind one variable to each of the basic tiles
29 ;
30 ( define basic-tile1 ( overlay ( pip ) blank-tile ) )
31
32 ( define basic-tile2
33 ( overlay/offset ( pip ) d d
34 ( overlay/offset ( pip ) nd nd blank-tile)
35 )
36 )
37
38 ( define basic-tile3 ( overlay ( pip ) basic-tile2 ) )
39
40 ( define basic-tile4
41 (overlay/offset ( pip ) d d
```

```
Untitled (define ...) [Icons] [Debug] [Macro Stepper] [Run] [Stop]
1 #lang racket
36 )
37
38 ( define basic-tile3 ( overlay ( pip ) basic-tile2 ) )
39
40 ( define basic-tile4
41   (overlay/offset ( pip ) d d
42     (overlay/offset ( pip ) nd nd
43       (overlay/offset (pip) d nd
44         (overlay/offset (pip) nd d blank-tile))))))
45
46 (define basic-tile5 (overlay (pip) basic-tile4))
47
48 (define basic-tile6
49   (overlay/offset (pip) d d
50     (overlay/offset (pip) nd nd
51       (overlay/offset (pip) d nd
52         (overlay/offset (pip) nd d
53           (overlay/offset (pip) 0 d
54             (overlay/offset (pip) 0 nd blank-tile)))))))
55
56
57
58 ;-----
59 ; The framed framed tiles
60 ;
61 ; - Bind one variable to each of the six framed tiles
62 ;
63 ( define frame ( square side-of-tile "outline" "gray" ) )
64 ( define tile0 ( overlay frame blank-tile ) )
65 ( define tile1 ( overlay frame basic-tile1 ) )
66 ( define tile2 ( overlay frame basic-tile2 ) )
67 ( define tile3 ( overlay frame basic-tile3 ) )
68 ( define tile4 ( overlay frame basic-tile4 ) )
69 ( define tile5 ( overlay frame basic-tile5 ) )
70 ( define tile6 ( overlay frame basic-tile6 ) )
71 ;-----
72 ; Domino generator
73 ;
74 ; - Funtion to generate a domino
75 ;
```

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

Demo:

Untitled (define ...) → [Icons] Debug [Icons] Macro Stepper [Icons] Run [Icons] Stop [Icons]

1 | #lang racket

Welcome to [DrRacket](#), version 8.6 [cs].
Language: **racket**, with **debugging**; memory limit: 128 MB.

> (domino 0 1)

	●
--	---

> (domino 3 2)

● ● ●	● ●
-------------	------------

> (domino 1 3)

● ●	● ● ●
------------	-------------

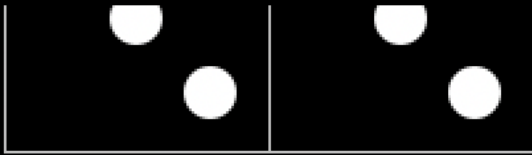
> (domino 2 0)

● ●	
------------	--

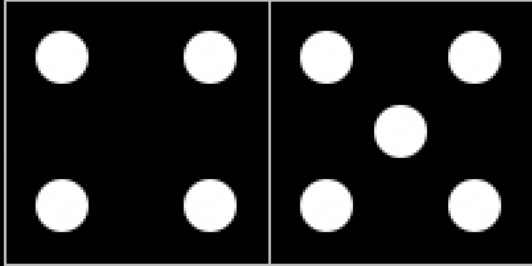
> (domino 3 3)

● ● ●	● ● ●
-------------	-------------

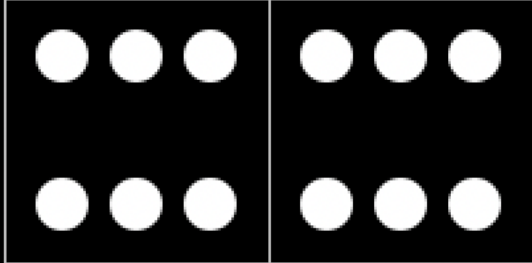
1 | #lang racket



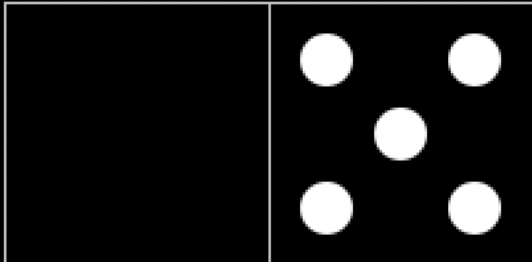
> (domino 4 5)



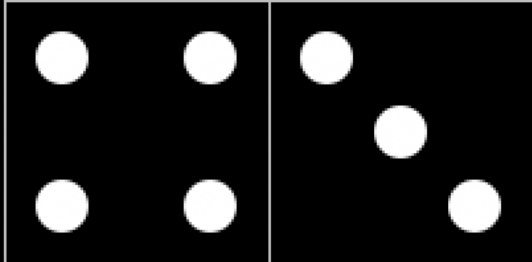
> (domino 6 6)



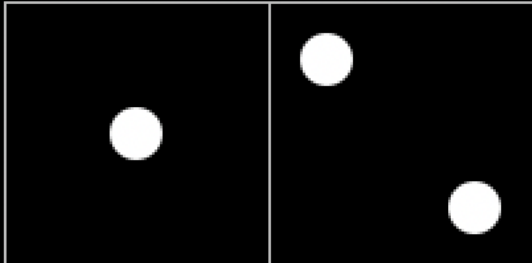
> (domino 0 5)



> (domino 4 3)



> (domino 1 2)



> |

Task 7: Creation

Creation/Code:

```
Untitled (define ...) Check Syntax Debug Macro Stepper Run Stop

1 #lang racket
2 (require 2htdp/image)
3 (define diameter 500)
4 (define dot (circle diameter "solid" "yellow"))
5 (define o (star 200 "outline" "white"))
6 (define rec (wedge 100 100 "outline" "black"))
7 (define l (line 200 200 "yellow"))
8 (define tri (triangle 60 "outline" "red"))
9 (define li (triangle 20 "solid" "blue"))
10
11 (define (spin-alot t)
12   (local [(define (spin-more i θ)
13             (cond
14               [(= θ 360) i]
15               [else
16                (spin-more (overlay i (rotate θ t))
17                           (+ θ 1))]))])
18   (spin-more t 0))

Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (overlay (spin-alot li) (overlay (spin-alot tri) (overlay (spin-alot rec) (overlay
(spin-alot l) (spin-alot o)))))
```

