

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

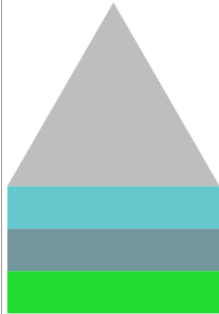
Learning Abstract:

In this assignment, I made use of recursive programming principles to solve multiple tasks many of which in this case made further use of the Racket 2htdp/image library. In addition to honing my ability to deduce how to solve problems recursively, this assignment was also a lesson in gaining familiarity on how to navigate programming language documentation. For the last part of the assignment, I chose the radial-star shape which had interesting parameters to make use of such as points, outer and inner radius to further manipulate the shape of the star.

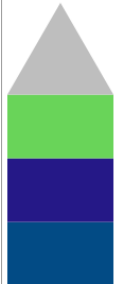
Question 1

Part 1.1

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



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



```
>
```

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

Part 1.2

```
> (tract 700 150)
> (tract 300 400)
> |
```

The image shows a sequence of visual elements representing recursive steps. The first set consists of six triangles, each with a grey top triangle and a bottom rectangle. The bottom rectangles are colored in a sequence: dark green, olive green, pink, dark green, pink, and olive green. The second set consists of six rectangles, each with a grey top triangle and a bottom rectangle. The bottom rectangles are colored in a sequence: purple, magenta, red, purple, red, and magenta. The third set consists of a single vertical line.

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

Part 1.3

```
1 #lang racket
2 (require 2htdp/image)
3
4 (define (random-color) (color (rgb-value) (rgb-value) (rgb-value) (rgb-value)))
5 (define (rgb-value) (random 256))
6
7
8 (define (floor width height color)
9   (rectangle width height "solid" color)
10  )
11
12 (define (roof width)
13   (triangle width "solid" "grey"))
14
15 (define (house width height color1 color2 color3)
16   (above (roof width) (floor width height color1)(floor width height color2)(floor width height color3)))
17
18 (define (divSix num)(/ num 6))
19
20 (define color1 (random-color))
21 (define color2 (random-color))
22 (define color3 (random-color))
23
24 (define (tract width height)
25   (beside (house (divSix width) (divSix height) color1 color2 color3)
26           (house (divSix width) (divSix height) color2 color1 color3)
27           (house (divSix width) (divSix height) color3 color1 color2)
28           (house (divSix width) (divSix height) color1 color3 color2)
29           (house (divSix width) (divSix height) color2 color3 color1)
30           (house (divSix width) (divSix height) color3 color2 color1)
31   )
32 | )
```

Question 2

Part 2.1

```
> (roll-die)
2
> (roll-die)
1
> (roll-die)
4
> (roll-die)
1
> (roll-die)
4
> (roll-for-1)
5 1
> (roll-for-1)
2 2 5 4 5 2 4 1
> (roll-for-1)
1
> (roll-for-1)
3 2 4 5 1
> (roll-for-1)
4 1
> (roll-for-11)
1 5 1 3 2 4 5 2 1 1
> (roll-for-11)
1 5 3 1 4 5 1 2 2 2 3 2 2 1 5 2 1 3 1 5 2 4 1 1
> (roll-for-11)
4 5 4 3 2 4 2 4 1 5 3 1 4 5 2 5 3 2 5 2 3 4 3 4 3 2 2 4 2 5 5 2 2 5 5 3 1 5 5 2 5 2 3 5 1 5 2 1 1
> (roll-for-11)
1 2 3 5 1 4 4 2 3 2 3 1 2 5 5 5 4 4 5 4 5 5 1 1
> (roll-for-11)
5 1 5 3 1 4 5 3 5 4 2 3 3 5 1 4 1 4 3 2 2 3 1 4 1 3 2 2 5 5 1 1
> (roll-for-11)
3 4 2 3 1 4 5 1 2 4 2 3 4 4 5 2 2 2 4 1 5 1 4 1 1
```

Nathaniel Wolf
Racket Programming Assignment #2: Racket Functions and Recursion
2/23/2022
CSC 344

Part 2.1 cont.

```
> (roll-for-odd-even-odd)
5 2 1
> (roll-for-odd-even-odd)
5 3 5 3 5 3 2 4 5 2 4 4 2 5 3 1 5 5 2 5
> (roll-for-odd-even-odd)
1 4 5
> (roll-for-odd-even-odd)
1 4 4 1 3 1 1 4 1 2 4 1 5 2 3 3 3 5 2 3 4 4 4 2 1 4 4 5 4 1
> (roll-for-odd-even-odd)
4 2 5 4 2 4 5 2 4 1 3 4 3 3 1 3 5 3 4 3 4 4 3 4 3
> (roll-for-odd-even-odd)
1 1 1 2 4 5 3 3 3 3 3 1 5 4 1 5 4 2 4 1 1 1 2 5
> (roll-two-dice-for-a-lucky-pair)
(5,3) (1,2) (3,5) (3,1) (1,1) Rolled a Double!
> (roll-two-dice-for-a-lucky-pair)
(2,5) Rolled a lucky 7!
> (roll-two-dice-for-a-lucky-pair)
(4,1) (5,3) (2,3) (1,3) (2,5) Rolled a lucky 7!
> (roll-two-dice-for-a-lucky-pair)
(1,1) Rolled a Double!
> (roll-two-dice-for-a-lucky-pair)
(5,2) Rolled a lucky 7!
> (roll-two-dice-for-a-lucky-pair)
(3,4) Rolled a lucky 7!
> (roll-two-dice-for-a-lucky-pair)
(5,4) (4,3) Rolled a lucky 7!
> (roll-two-dice-for-a-lucky-pair)
(3,3) Rolled a Double!
> (roll-two-dice-for-a-lucky-pair)
(1,2) (5,2) Rolled a lucky 7!
> (roll-two-dice-for-a-lucky-pair)
(2,1) (3,5) (1,3) (4,5) (1,2) (4,2) (2,3) (4,2) (2,5) Rolled a lucky 7!
> |
```

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

Part 2.2

```
1 #lang racket
2 (define (roll-die) (random 1 6))
3
4 (define (roll-for-1)
5   (define outcome (roll-die) )
6   (cond
7     ((= outcome 1) (display outcome) (display " " ) )
8     ((display outcome) (display " ") (roll-for-1))))
9
10 (define (roll-for-11)
11   (roll-for-1 )
12   (define outcome (roll-die) )
13   (display outcome ) (display " " )
14   (cond
15     ((not (eq? outcome 1) ) (roll-for-11 ))))
16
17 (define (roll-for-odd-even-odd)
18   (define outcome (roll-die))
19   (display (format "~a " outcome) )
20   (cond
21     [(even? outcome)(roll-for-odd-even-odd)]
22     [(odd? outcome) ; else is odd
23      (define outcome (roll-die))
24      (display (format "~a " outcome) )
25      (cond
26        [(odd? outcome)(roll-for-odd-even-odd)]
27        [(even? outcome) ;; else is even
28         (define outcome (roll-die))
29         (display (format "~a " outcome) )
30         (cond
31           [(even? outcome)(roll-for-odd-even-odd)])))])))]))
32
33 (define (roll-two-dice-for-a-lucky-pair)
34   (define roll-1 (roll-die) )
35   (define roll-2 (roll-die) )
36   (define sum (+ roll-1 roll-2) )
37   (display (format "(~a,~a)" roll-1 roll-2))
38   (cond
39     [(= sum 7) (display " Rolled a lucky 7! ")]
40     [(= sum 11) (display " Rolled a Lucky 11! ")]
41     [(= roll-1 roll-2)(display " Rolled a Double! ")]
42     [else (display " ")(roll-two-dice-for-a-lucky-pair)]))
43
44 (define (roll-for-odd)
45   (define outcome (roll-die) )
46   (cond
47     (( odd? outcome) (display outcome) (display " " ) )
48     ((display outcome) (display " ") (roll-for-odd))))
49
50 (define (roll-for-even)
51   (define outcome (roll-die) )
52   (cond
53     (( even? outcome) (display outcome) (display " " ) )
54     ((display outcome) (display " ") (roll-for-even))))
```

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

Question 3

Part 3.1

```
> ( 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
>
```

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

Part 3.2

```
1 #lang racket
2
3 (define (square n)(* n n))
4
5 (define (cube n)(* n n n))
6
7 (define (sequence name n)
8   (cond
9     ((= n 1)
10      (display (name 1)) (display " ")))
11     (else
12      (sequence name (- n 1))
13      (display (name n)) (display " "))))
14
15 (define (triangular n)
16   (cond
17     ((= n 1)1)
18     (else (+ (triangular (- n 1)) n))))
19
20 (define (sigma n)
21   (sigrun n n))
22
23 (define (sigrun m n)
24   (cond
25     ((= n 1) 1)
26     (else
27      (cond
28        ((= (remainder m n) 0)
29         (+ (sigrun m (- n 1)) n))
30        (else(sigrun m (- n 1)))))))
31
```

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

Question 4

Part 4.1

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



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



```
>
```


Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

Part 4.2

```
1 #lang racket
2 (require 2htdp/image)
3
4 (define (random-color) (color (rgb-value) (rgb-value) (rgb-value)))
5 (define (rgb-value) (random 256))
6
7 (define (dot)
8   (define frame (square 50 0 "white"))
9   (define dot-shape (circle 15 "solid" (random-color)))
10  (overlay dot-shape frame))
11
12 (define (hirst-square r c)
13   (cond
14     [(= c 1) (row-of-dots r)]
15     [else (above (row-of-dots r) (hirst-square r (- c 1)))]))
16
17
18 (define (row-of-dots n)
19   (cond
20     [(= n 1) (dot)]
21     [else (beside (dot) (row-of-dots (- n 1)))]))
22
23 (define (hirst-dots n)
24   (hirst-square n n))
```

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

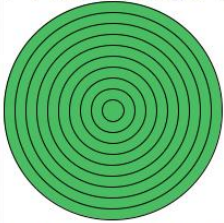
2/23/2022

CSC 344

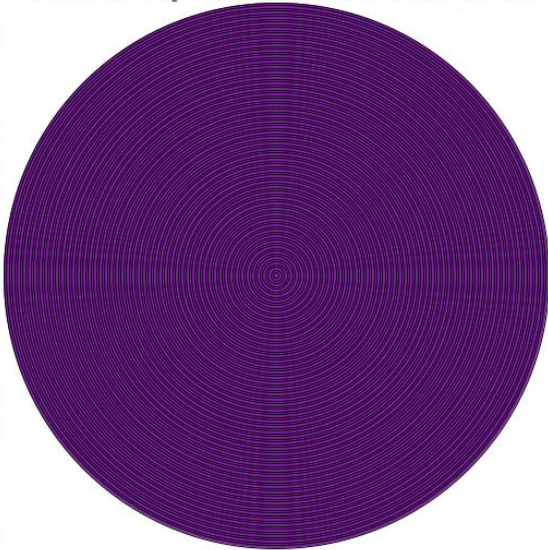
Question 5

Part 4.1

```
> ( nested-shapes-one 100 10 ( random-color) )
```



```
> ( nested-shapes-one 250 100 ( random-color) )
```



```
>
```

Part 5.2

```
1 #lang racket
2
3 ( require 2htdp/image )
4
5 ( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ))
6 ( define ( rgb-value ) ( random 256 ) )
7
8 ( define ( framed-shape side-length color )
9   ( overlay
10     ( circle side-length "outline" "black" )
11     ( circle side-length "solid" color )))
12
13 ( define ( nested-shapes-one side count color )
14   ( define unit ( / side count ) )
15   ( paint-nested-shapes-one 1 count unit color ) )
16
17 ( define ( paint-nested-shapes-one from to unit color )
18   ( define side-length ( * from unit ) )
19   ( cond
20     [( = from to ) ( framed-shape side-length color)]
21     [( < from to ) ( overlay ( framed-shape side-length color)
22                               ( paint-nested-shapes-one ( + from 1 ) to unit color))]))
```

Nathaniel Wolf

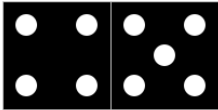
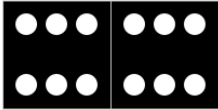
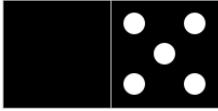
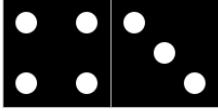
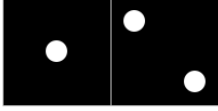
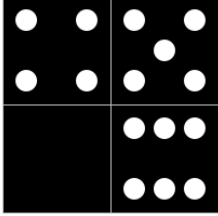
Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

Question 5

Part 5.1

```
> ( domino 4 5 )  
  
> ( domino 6 6 )  
  
> ( domino 0 5 )  
  
> ( domino 4 3 )  
  
> ( domino 1 2 )  
  
> ( above ( domino 4 5 ) ( domino 0 6 ))  
  
> |
```

Nathaniel Wolf

Racket Programming Assignment #2: Racket Functions and Recursion

2/23/2022

CSC 344

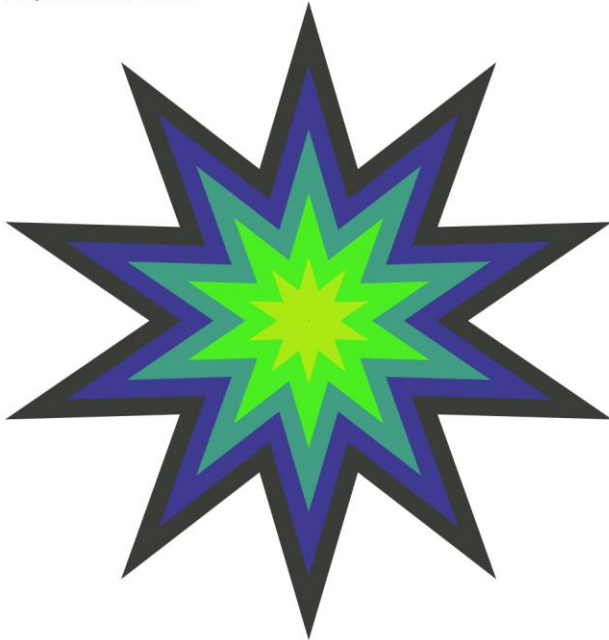
Part 5.2

```
1 #lang racket
2
3 ( require 2htdp/image )
4 ( define side-of-tile 100 )
5 ( define diameter-of-pip ( * side-of-tile 0.2 ) )
6 ( define radius-of-pip ( / diameter-of-pip 2 ) )
7 ;-----
8 ; Numbers used for offsetting pips from the center of a tile
9 ; - d and nd are used as offsets in the overlay/offset function applications
10 ;
11 ( define d ( * diameter-of-pip 1.4 ) )
12 ( define nd ( * -1 d ) )
13 ;-----
14 ; The blank tile and the pip generator
15 ; - Bind one variable to a blank tile and another to a pip
16 ;
17 ( define blank-tile ( square side-of-tile "solid" "black" ) )
18 ( define ( pip ) ( circle radius-of-pip "solid" "white" ) )
19 ;-----
20 ; The basic tiles
21 ; - Bind one variable to each of the basic tiles
22 ;
23 ( define basic-tile1 ( overlay ( pip ) blank-tile ) )
24 ( define basic-tile2 ( overlay/offset ( pip ) d d ( overlay/offset ( pip ) nd nd blank-tile ) ) )
25 ( define basic-tile3 ( overlay ( pip ) basic-tile2 ) )
26 ( define basic-tile4 ( overlay/offset ( pip ) nd d ( overlay/offset ( pip ) d nd basic-tile2 ) ) )
27 ( define basic-tile5 ( overlay ( pip ) basic-tile4 ) )
28 ( define basic-tile6 ( overlay/offset ( pip ) 0 d ( overlay/offset ( pip ) 0 nd basic-tile4 ) ) )
29 ;-----
30 ; The framed framed tiles
31 ; - Bind one variable to each of the six framed tiles
32 ;
33 ( define frame ( square side-of-tile "outline" "gray" ) )
34 ( define tile0 ( overlay frame blank-tile ) )
35 ( define tile1 ( overlay frame basic-tile1 ) )
36 ( define tile2 ( overlay frame basic-tile2 ) )
37 ( define tile3 ( overlay frame basic-tile3 ) )
38 ( define tile4 ( overlay frame basic-tile4 ) )
39 ( define tile5 ( overlay frame basic-tile5 ) )
40 ( define tile6 ( overlay frame basic-tile6 ) )
41
42 ;-----
43 ; Domino generator
44 ; - Funtion to generate a domino
45 ;
46 ( define ( domino a b )
47   ( beside ( tile a ) ( tile b ) ) )
48 ( define ( tile x )
49   ( cond
50     ( ( = x 0 ) tile0 )
51     ( ( = x 1 ) tile1 )
52     ( ( = x 2 ) tile2 )
53     ( ( = x 3 ) tile3 )
54     ( ( = x 4 ) tile4 )
55     ( ( = x 5 ) tile5 )))
```

Question 7

Part 7.1

```
> (generate-stars 10 500)
```



Part 7.2

```
1 #lang racket
2 (require 2htdp/image)
3 (define (random-color) (color (rgb-value) (rgb-value) (rgb-value)))
4 (define (rgb-value) (random 256))
5
6 (define (create-star points radius)
7   (radial-star points (/ radius 2) radius "solid" (random-color)))
8
9 (define (generate-stars points radius)
10  (cond
11   [(eq? radius 0) (create-star points radius)]
12   [else (overlay
13         (generate-stars points (- radius 100)) (create-star points radius))]))
```