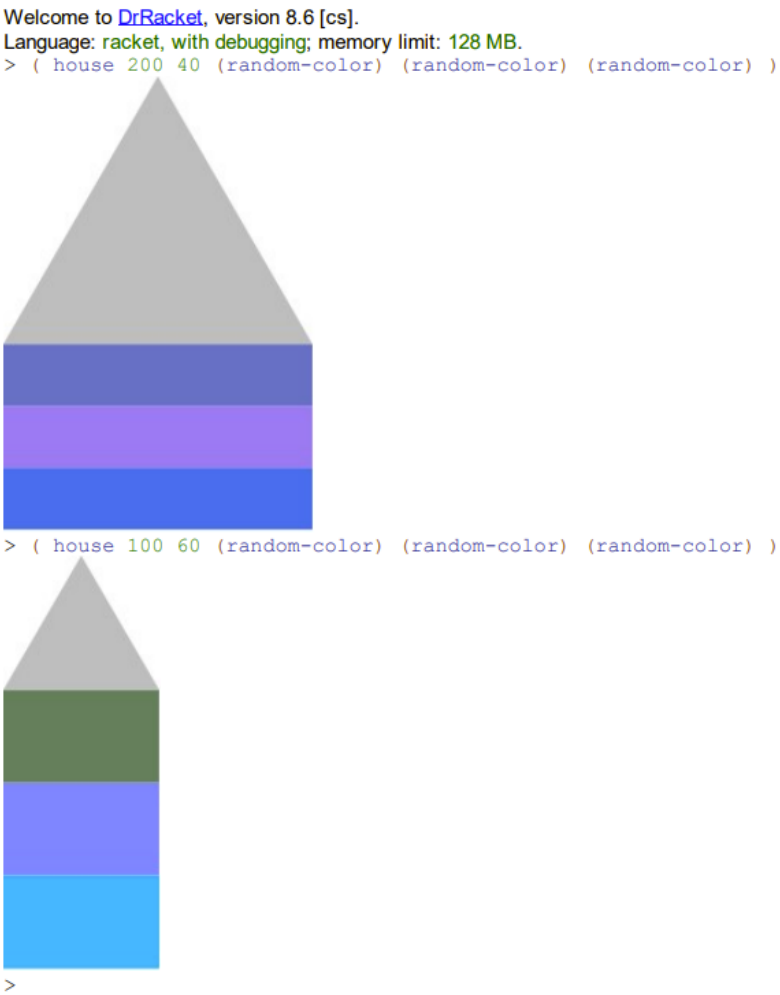
Racket Programming Assignment #2: Racket Functions and Recursion

Learning Abstract

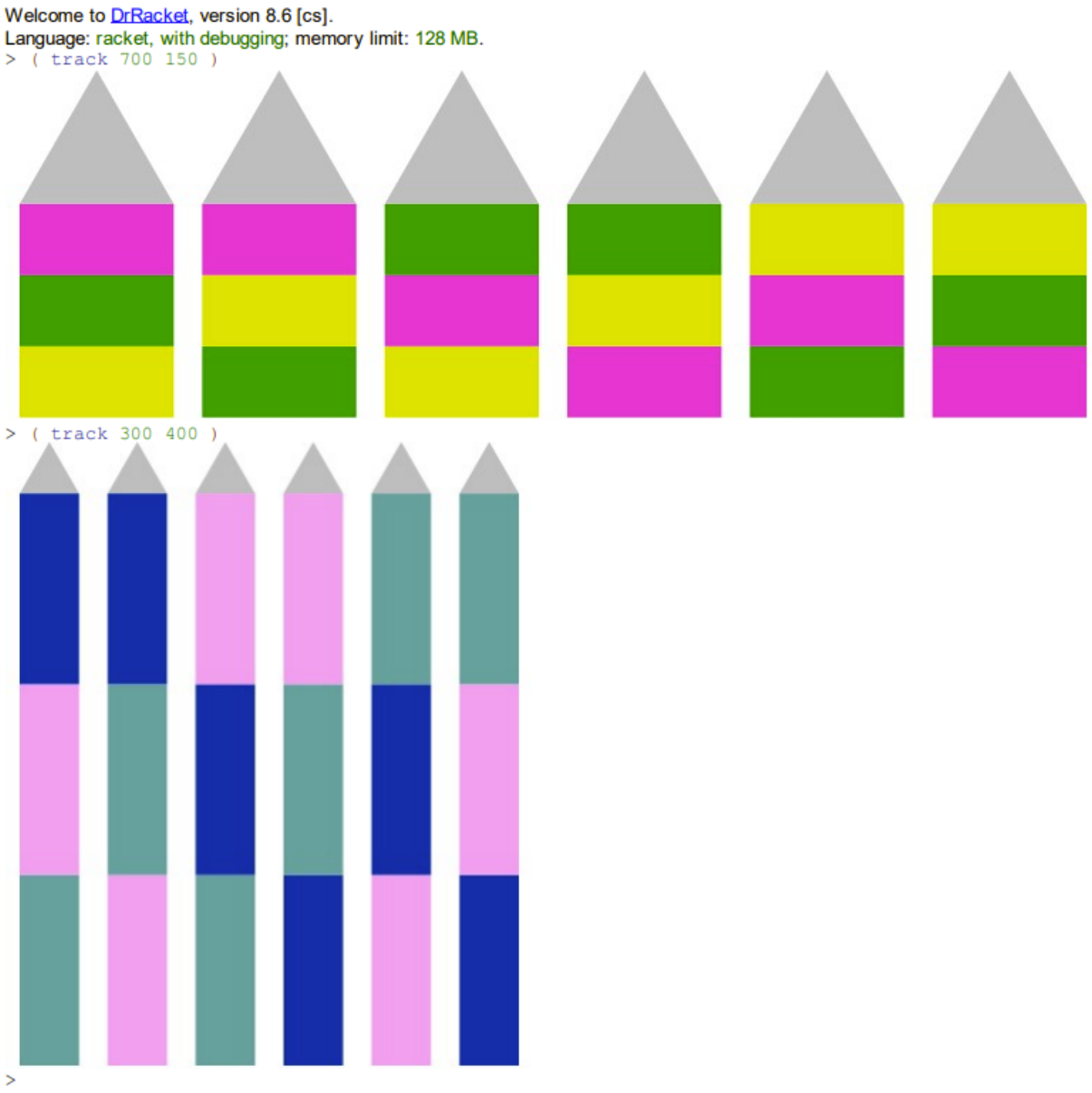
The assignment is designed to reinforce how to create functions and solve problems in Racket through recursion.

Task 1: Colorful Permutations of Track House

Demo for House



Demo for tract



Code

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

```
; creating a house
; -------------------------------------------------------------------------

( define ( house width height color1 color2 color3 )
   ( above ( triangle width "solid" "grey")
         ( rectangle width height "solid" color1 )
         ( rectangle width height "solid" color2 )
         ( rectangle width height "solid" color3 )
   )
)

; -------------------------------------------------------------------------
; Create track
; -------------------------------------------------------------------------

( define ( the-track width height color1 color2 color3 )
   (overlay ( house width height color1 color2 color3 ) ( rectangle ( + width 20 ) height
"solid" "white" ))
)

( define ( track width height )
   ( define width-of-house ( / ( - width ( * 10 5 ) ) 6) )
   ( define floor-height ( / height 3 ) )
   ( define color1 ( random-color ) )
   ( define color2 ( random-color ) )
   ( define color3 ( random-color ) )
   ( define roof ( triangle width-of-house "solid" "grey" ) )
   ( beside ( the-track width-of-house floor-height color1 color2 color3 )
         ( the-track width-of-house floor-height color1 color3 color2 )
         ( the-track width-of-house floor-height color2 color1 color3 )
         ( the-track width-of-house floor-height color2 color3 color1 )
         ( the-track width-of-house floor-height color3 color1 color2 )
         ( the-track width-of-house floor-height color3 color2 color1 )
   )
)
```

Task 2: The Dice

The Demo

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( roll-die )
2
> ( roll-die )
6
> ( roll-die )
3
> ( roll-die )
2
> ( roll-die )
3
> ( roll-for-1 )
5 6 4 5 4 5 1
> ( roll-for-1 )
1
> ( roll-for-1 )
3 1
> ( roll-for-1 )
5 1
> ( roll-for-1 )
4 6 3 3 5 1
> ( roll-forr-11 )
roll-forr-11: undefined;
 cannot reference an identifier before its definition
> ( roll-for-11 )
4 5 5 4 2 4 2 6 2 3 5 6 1 1
> ( roll-for-11 )
6 1 1
> ( roll-for-11 )
2 1 1
> ( roll-for-11 )
6 6 1 6 5 2 3 3 6 1 2 2 1 5 1 4 4 2 1 5 4 3 2 3 3 5 5 3 1 4 5 4 2 6 3 1 6 4 3 1 2 6 6 3 5 6 4 3
6 5 1 5 1 1
```

```
> ( roll-for-odd-even-odd )
4 6 1 1 4 1
> ( roll-for-odd-even-odd )
2 1 2 4 4 1 2 6 6 1 5 6 3
> ( roll-for-odd-even-odd )
4 6 3 6 4 5 1 4 3 2 5
> ( roll-for-odd-even-odd )
5 2 1
> ( roll-for-odd-even-odd )
2 4 1 5 3 2 1 5 1 2 1 3 6 3
> ( roll-for-dice-for-a-lucky-pair )
roll-for-dice-for-a-lucky-pair: undefined;
 cannot reference an identifier before its definition
> ( roll-two-dice-for-a-lucky-pair )
(2 2)
> ( roll-two-dice-for-a-lucky-pair )
(5 6)
> ( roll-two-dice-for-a-lucky-pair )
(6 3) (5 1) (6 5)
> ( roll-two-dice-for-a-lucky-pair )
(5 5)
> ( roll-two-dice-for-a-lucky-pair )
(1 6)
> ( roll-two-dice-for-a-lucky-pair )
(2 6) (1 5) (5 4) (6 6)
> ( roll-two-dice-for-a-lucky-pair )
(6 2) (1 6)
> ( roll-two-dice-for-a-lucky-pair )
(6 3) (4 3)
> ( roll-two-dice-for-a-lucky-pair )
(4 2) (5 2)
```

```
> ( roll-two-dice-for-a-lucky-pair )
 (6 2) (6 5)
>
```

Code

#lang racket

```
;-------------------------------------------------------------------------------
; Roll die
;-------------------------------------------------------------------------------
( define ( roll-die )
   ( + ( random 6 ) 1 ))
;-------------------------------------------------------------------------------
; Roll for one
;-------------------------------------------------------------------------------
( define ( roll-for-1)
   ( define roll ( roll-die) )
   ( display roll )( display " ")
   (cond ( ( not ( eq? roll 1 ) )
          ( roll-for-1)
          )
   )
)
;-------------------------------------------------------------------------------
; Roll for two consecutive ones
;-------------------------------------------------------------------------------
( define ( roll-for-11)
   ( roll-for-1 )
   ( define roll ( roll-die))
   ( display roll )( display " ")
   (cond ( ( not ( eq? roll 1 ) )
                ( roll-for-11)
           )
   )
)

;-------------------------------------------------------------------------------
; Roll for odd then even then odd
```

```
;-------------------------------------------------------------------------------


(define ( roll-for-odd-even-odd )
; first roll
  ( define 1roll ( roll-die ) )
  ( display 1roll ) ( display " " )
  ; see if roll 1 is odd
  ( cond
    ( ( odd? 1roll )
      ;see if roll 2 is even
      ( define 2roll ( roll-die ) )
      (cond
        ( ( even? 2roll )
          ( display 2roll ) ( display " " )
          ;see if roll 3 is odd
          ( define 3roll ( roll-die ) )
          ( cond
            ( ( odd? 3roll )
              ( display 3roll )( display "\n" ) )
            ; see if roll 3 even
            ( ( even? 3roll)
              ( roll-for-odd-even-odd )
              )
            )
          )
          ; see if roll 2 odd
        ( ( odd? 2roll )
          (roll-for-odd-even-odd)
          )
        )
      )
      ; see if roll 1 even
      ( ( even? 1roll )
        ( roll-for-odd-even-odd)
        )
      )
    )
```

```
;-------------------------------------------------------------------------
; Roll for two dice for  sum of 7, 11 or a pair.
;-------------------------------------------------------------------------

( define ( roll-two-dice-for-a-lucky-pair )
  ( define die1 ( roll-die))
  ( display " (" ) ( display die1 ) (display " " )
  (define die2 ( roll-die ) )
  ( display die2) (display")" )
  (cond
   ( ( = ( + die1 die2 ) 7 )
     ( display "\n" )
     )
   ( ( = ( + die1 die2 ) 11 )
     ( display "\n" )
     )
   ( ( = die1 die2 )
     ( display "\n" )
     )
   ( else ( roll-two-dice-for-a-lucky-pair ) )
   )
  )
```

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
>
```

Sigma Demo

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

Code

```
#lang racket
( 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 " " )
    )
  )
)


;-----------------------------------------------------
; Triangular
;-----------------------------------------------------

( define ( triangular n )
   ( cond
      ( ( = n 1 ) 1 )
      ( ( > n 1 )
        ( + n ( triangular ( - n 1 ) ) )
        )
      )
   )
;-----------------------------------------------------
; Sigma
;-----------------------------------------------------

( define ( sigma n )
   ( calc n n )
)
( define ( calc n 1n)
   ( cond
      ( ( = n 1)  1 )
      ( ( = ( modulo 1 n n ) 0 )
       ( + n ( calc ( - n 1 ) 1n ) )
       )
      ( else
        (calc ( - n 1 ) 1n )
      )
   )
)
```

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

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

;-------------------------------------------------------------
;create dot and space
;-------------------------------------------------------------

(define ( dot ) ( overlay ( circle 15 "solid" (random-color) )
                          ( square 40 "solid" "white" ) )
  )

;-----------------------------------------------------
; row of dots
;-----------------------------------------------------

(define ( row-of-dots n dot )
  ( cond
     ( ( = n 0 )
       empty-image
     )
     ( ( > n 0 )
       ( beside ( row-of-dots ( - n 1 ) dot ) ( dot ) )
     )
  )
)
;-----------------------------------------------------
; rectangle of dots
;-----------------------------------------------------
( define ( rectangle-of-dots row column dot )
   ( cond
      ; row empty
      ( ( = row 0 )empty-image )
      ; row not empty
      ( ( > row 0)
```

```
        ( above ( rectangle-of-dots ( - row 1 ) column dot ) ( row-of-dots column dot ) )
        )
      )
   )

;-------------------------------------------------------
; square of dots
;-------------------------------------------------------

(define ( square-of-dots n dot )
  ( rectangle-of-dots n n dot )
)


(define ( hirst-dots n )
  (square-of-dots n dot ) )
```

Task 5: Channelling Frank Stella

Demo

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( stella 300 10 "yellow" "red" )
```

```
> ( stella 200 5 "green" "blue" )
```

```
>
```

Code

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


;-----------------------------------------------
; Create nested triangles ( for monochromatic )
;-----------------------------------------------
;( define ( stella side count color )
;( define unit ( / side count ) )
;( paint-nested-triangles-one 1 count unit color )
;)
;-----------------------------------------------
; Paint nested triangles
;-----------------------------------------------

( define ( paint-nested-triangles-one from to unit color)
  ( define side-length ( * from unit ) )
  ( cond
    ( ( = from to )
      ( framed-triangle side-length color )
      )
    ( ( < from to )
      ( overlay
        ( framed-triangle side-length color )
        ( paint-nested-triangles-one ( + from 1 ) to unit color )
        )
      )
    )
  )

;-----------------------------------------------
; Frame triangle
;-----------------------------------------------

( define ( framed-triangle side-length color )
  ( overlay
    ( triangle side-length "outline" "black" )
    ( triangle side-length "solid" color )
    )
  )
```

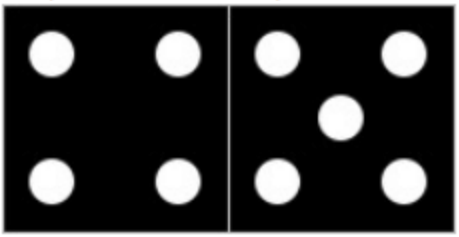
```
;-------------------------------------------------
; Two tone stella traigle
;-------------------------------------------------
( define ( stella side count color1 color2 )
  ( define delta ( / side count ) )
  ( paint-nested-triangles-two 1 count delta color1 color2 )
  )


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

<u>Task 6: Dominos</u>

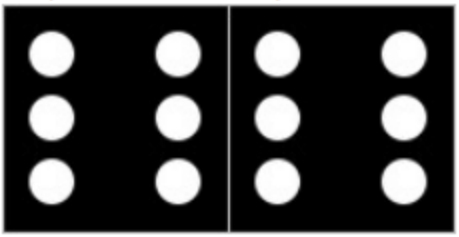
Final demo

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

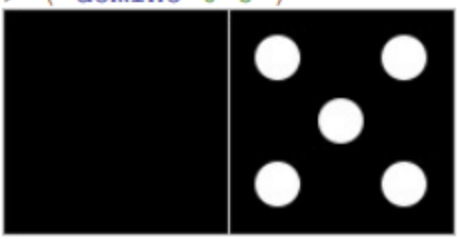
```
> ( domino 4 5 )
```



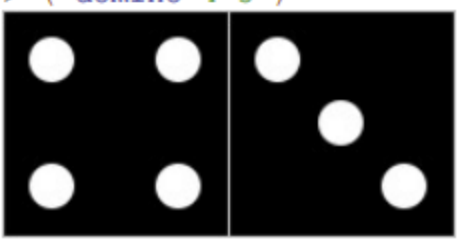
```
> ( domino 6 6 )
```



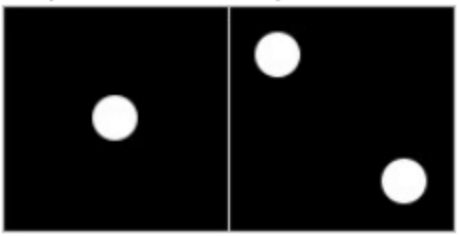
```
> ( domino 0 5 )
```



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



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



```
>
```

Collected code

#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 ) d 0
                               ( overlay/offset ( pip ) nd 0 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

Creation (image)

Code

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

;-------------------------------------------------------
; create foliage
;-------------------------------------------------------

( define dot ( circle 30 "solid" "dark green" ) )

( define bush ( overlay/offset
                ( overlay/offset
                  (overlay/offset
```

```
                (overlay/offset
                 (overlay/offset
                  dot -10 10 dot  )
                 20 10 dot  )
                30 10 dot )
               -40 10 dot )
              10 -10 dot )
  )

( define more-bush ( overlay/offset (overlay/offset bush -50 40 bush ) 50 30 bush ) )

;-------------------------------------------------------
; create bark
;-------------------------------------------------------

( define branch ( rectangle 20 70 "solid" "dark brown" ) )

( define trunk ( rectangle 50 200 "solid" "dark brown" ) )

( define tree ( overlay/offset ( overlay/offset ( rotate 45 branch ) 40 20 trunk ) 60 -10 ( rotate 130
branch ) ) )

;-------------------------------------------------------
; create tree
;-------------------------------------------------------

( define my-creation ( overlay/offset more-bush 0 80 tree ) )
```