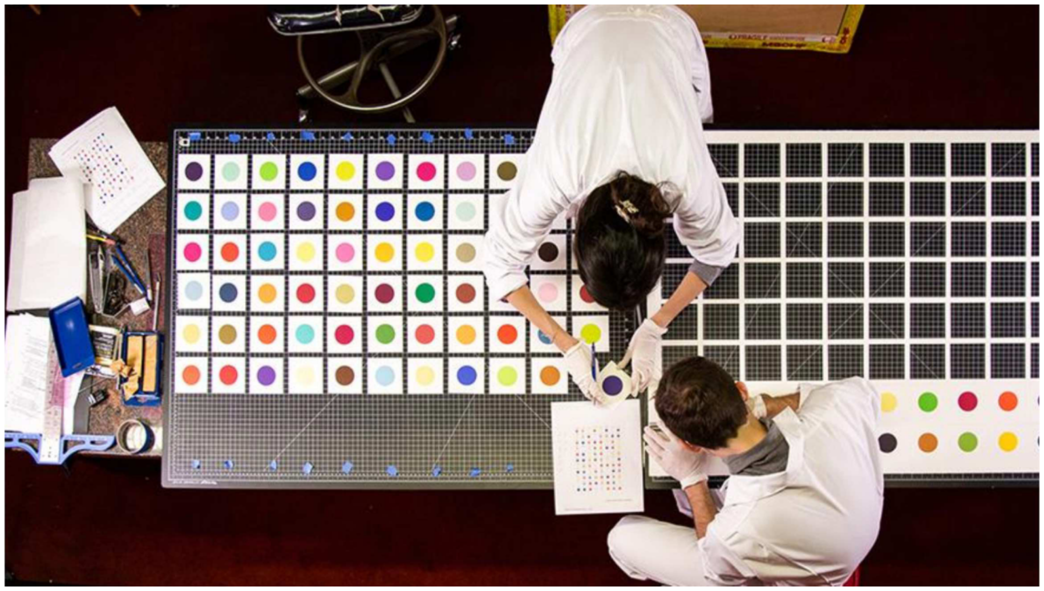
Racket Programming Assignment #2: Racket Functions and Recursion



What's It All About?

This is all about affording you the opportunity to write some racket programs, including a number of recursive functions. More precisely, you will be asked to:

1. Write a program to display images of colorful tract houses that are grouped by the concept of permutation.
2. Write some programs to do things with virtual dice.
3. Write a couple of programs pertaining to classical number sequences.
4. Write a program to display a grid of Hirst dots.
5. Write a program to display images which channel Frank Stella.
6. Complete a program to display a set of dominos.
7. Programmably create an image that uses bits of functionality from the `2htdp/image` library which was featured in class.

Overall Charge

Working within the DrRacket PDE, please do each of the numbered programming tasks. Then please do the document compilation/posting task.

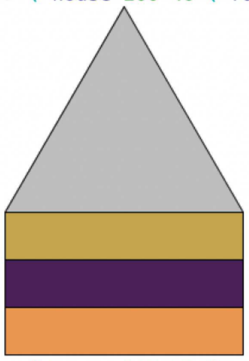
Task 1: Colorful Permutations of Tract Houses

Programming constraint: For this part of your assignment, your are not permitted to use any form of repetition (recursion/iteration) or any form of conditional statement (e.g., if, cond).

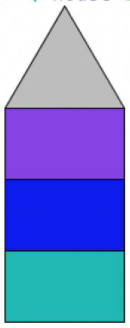
1. Please study the "randomly colored checkerboard" program that was featured in Lesson 2, and the way it was developed. That program is intended to serve as a resource for you to consult as you engage in this programming exercise.
2. Write a function called `house` that creates an image of a house consisting of 3 floors and a roof. All three floors are of the same size, given by a width and a height. Moreover, all three floors are of a different color. The `house` function is to take five parameters, the width of a floor, the height of a floor, the color if the first floor, the color of the second floor, and the color of the third floor. The roof will always be a gray equilateral triangle of just the right size. If the words are overwhelming, just look at the examples presented in the accompanying `house` demo.
3. Write a program called `tract` which creates an image of 6 houses, side by side, separated by a space of 10 pixels, subject to the constraint that the floors of the six houses represent the set of all permutations on 3 random colors. The `tract` function is to take two parameters, the width of the track, and the height of the three floors of the house (this height does not include the roof). Please look to the accompanying `tract` demo for clarification.

The `house` Demo

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

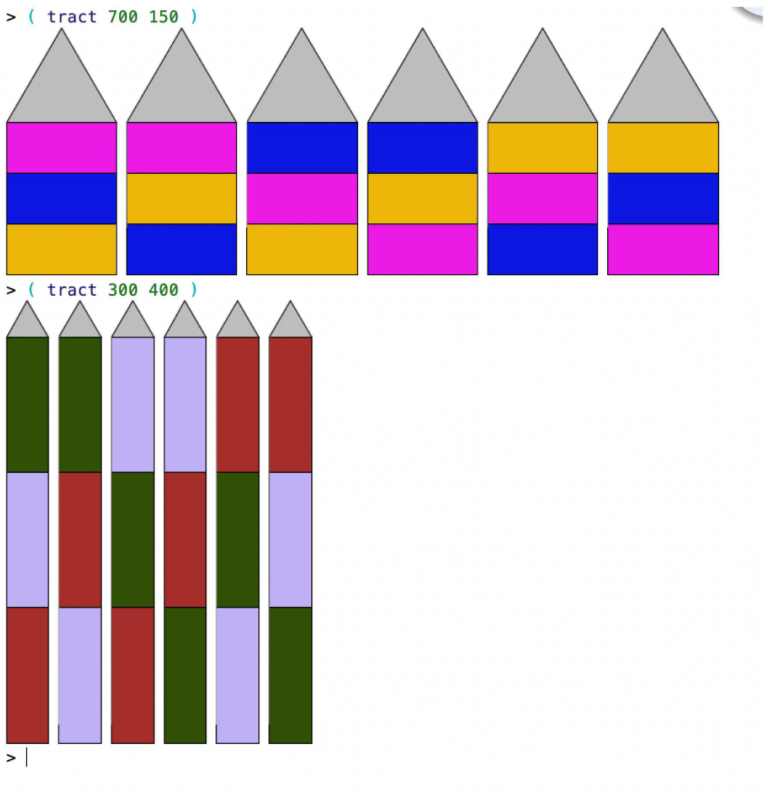


```
> ( house 100 60 ( random-color ) ( random-color ) ( random-color ) )
```



```
>
```

The tract Demo



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

Contribution to the solution document

For the section of your solution document that corresponds to this part of your assignment, please include:

1. A snapshot of a demo for the `house` function that is just like that provided, except for the colors.
2. A snapshot of a demo for the `tract` function that is just like that provided, except for the colors.
3. The code for the `house` function, for the `tract` function, and for all of the code that your wrote in support of these two functions.

Task 2: Dice

Programming constraint: For this part of your assignment, your are not permitted to use any form of iterative construct. Rather, you are required to use recursion for any repetition that you would like to accomplish.

---

1. Please study the accompanying demo, which you will be expected to replicate in terms of function calls. The demo illustrates the behavior of the functions that you are asked to write for this part of the assignment.
2. Write a function called `roll-die` that simulates the roll of a standard die in that it returns an random integer between 1 and 6.
3. Write a function called `roll-for-1` that simulates the roll of a standard die until a 1 turns up, tracing the outcomes of the rolls along the way.
4. Write a function called `roll-for-11` that simulates the roll of a standard die until two consecutive 1s turn up, tracing the outcomes of the rolls along the way.
5. Write a function called `roll-for-odd-even-odd` that simulates the roll of a standard die until consecutive values which are odd then even then odd turn up, tracing the outcomes of the rolls along the way.
6. Write a function called `roll-two-dice-for-a-lucky-pair` that simulates the roll of two standard dice until either the sum of seven, the sum of eleven, or a double turns up, tracing the outcomes of the rolls along the way.

The Demo

```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( roll-die )
5
> ( roll-die )
3
> ( roll-die )
2
> ( roll-die )
5
> ( roll-die )
6
> ( roll-for-1)
1
> ( roll-for-1 )
3 4 5 5 5 4 1
> ( roll-for-1 )
1
> ( roll-for-1 )
6 2 1
> ( roll-for-1 )
2 4 1
> ( roll-for-11 )
4 6 6 6 1 5 4 6 4 4 1 3 3 5 6 4 2 1 6 6 4 1 1
> ( roll-for-11 )
6 1 2 4 6 2 6 4 1 4 1 2 1 3 4 6 2 2 4 5 6 2 5 2 5 1 2 4 6 1 2 4 2 1 2 3 5 4 5 6 5 6 5 6 3 4 1 1
```

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

Contribution to the solution document

For the section of your solution document that corresponds to this part of your assignment, please include:

1. A demo in which `roll-die` is run 5 times, `roll-for-1` is run 5 times, `roll-for-11` is run 5 times, `roll-for-odd-even-odd` is run 5 times, and `roll-two-dice-for-a-lucky-pair` is run 10 times.
2. Your code for the five featured functions, and for any other functions that you write in support of the featured functions.

Task 3: Number Sequences

Programming constraint: For this part of your assignment, your are not permitted to use any form of of iterative construct. Rather, you are required to use recursion.

1. Please study the “Preliminary Code” that consists of three functions: (1) a function to compute the square of a number, (2) a function to compute the cube of a number, and (3) a higher order function to compute the first n elements in a number sequence, assuming that functions exist to compute the nth element of the sequence. Then type these functions into a Racket file, and get them to work properly.
2. Create a demo that mimics the given “Preliminary Demo”.
3. Within the same Racket file, please write a function called `triangular` taking one positive integer as its sole parameter which returns the triangular number corresponding to the given value. Notes: (1) While this one could be written with a simple closed expression, it is nevertheless required that you write it recursively. (2) The nth triangular number is the sum of the numbers from 1 to n.
4. Once your function is working properly, mimic the given “Triangular” demo.
5. Within the same Racket file, please write a function called `sigma` taking one positive integer as its sole parameter which returns the sum of the divisors of the given number. Hint: The `sigma` function should probably call a function of two parameters, which will need to be written in a recursive faction due to the recursive programming constraint.
6. Once your function is working properly, mimic the given “Sigma” demo.

Preliminary Code

```
( 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 " " )
    )
  )
)
```

Preliminary Demo

```
> ( 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

```
> ( 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

```
> ( 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
>
```

Contribution to the solution document

For the section of your solution document that corresponds to this part of your assignment, please include:

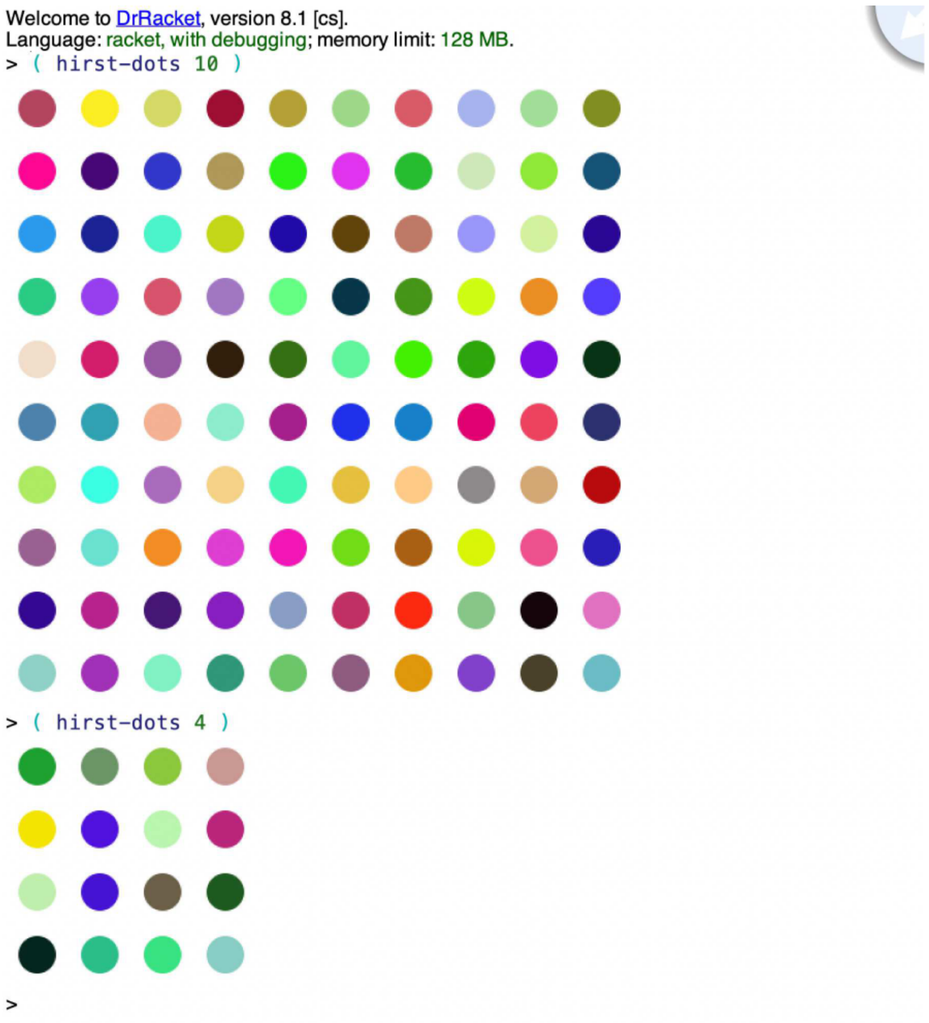
1. Three demos: Your re-creation of the “preliminary demo”. Your re-creation of the “Triangular” demo. Your re-creation of the “Sigma” demo.
2. A listing of the code that was involved in generating the three demos (my preliminary code, your code for triangular numbers and the sigma (sum of divisors) numbers.

Task 4: Hirst Dots

Programming constraint: For this part of your assignment, your are not permitted to use any form of of iterative construct. Rather, you are required to use recursion.

1. Please study the "Grid of Squares" portion of Lesson 3. The programs featured in that section of the lesson are intended to serve as a resource for you to consult as you engage in this programming activity. Furthermore, you might find it beneficial to keep the first problem of this assignment in mind as you think about how to accomplish what you need to do for this problem.
2. Write a function called `hirst-dots` to draw square arrangements of Hirst dots that are consistent with the accompanying demo. Please arrange for the diameter of the dots to be 30 pixels, and for each dot to be 20 pixels from its nearest dots. Please note that the the parameter is presumed to be a non-negative integer, and that the number of dots in an image is the square of the value of the parameter.

The Demo



```
Welcome to DrRacket, version 8.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( hirst-dots 10 )
> ( hirst-dots 4 )
>
```

Contribution to the solution document

For the section of your solution document that corresponds to this part of your assignment, please include:

1. A demo that displays a 10x10 grid of Hirst dots, and a 4x4 grid of Hirst dots.
2. Your code for the `hirst-dots` function, and for any functions that you write in support of it.

Task 5: Chanelling Frank Stella

Programming constraint: For this part of your assignment, your are not permitted to use any form of of iterative construct. Rather, you are required to use recursion.

1. Please study the "Channeling Frank Stella" portion of Lesson 3. The programs featured in that section of the lesson are intended to serve as a resource for you to consult as you engage in the programming for this problem.
2. Write a function called `stella` to display graphical images in the spirit of Frank Stella subject to the following constraints:
   (a) Your program must be based on a shape other than either of those that I used in the Stella section Lesson 3. Thus, you must not use either a square or a star for it. Perhaps you would like to use a circle, or an ellipse, or a wedge, or a triangle, or rhombus, or a regular polygon, or a star-polygon. Maybe something else! Please find your way to the documentation for the `2htdp/image` library to select some functionality that resonates with you. (Recall that the "2htdp Image Documentation" section of Racket Lesson #3 references the main functionality associated with this library.)
   (b) Your program must be consistent with one of the two varieties of Stella images that were presented in Lesson 3. That is, speaking in the terminology dropped in the lesson, you must do a monochromatic image, or a two-tone image.

Contribution to the solution document

For the section of your solution document that corresponds to this part of your assignment, please include:

1. A demo that displays at least two different images in the family of images that your program produces.
2. Your code for your "Stella variation" function, and for any functions that you write in support of it.

Task 6: Dominos

This part of the assignment invites you to complete a partially written program, not an uncommon activity in the world of computer programming. The typical incremental approach to programming will be adopted for this problem. I will provide you with the start of a program to render images of dominos. I will then ask you to complete the program by scaling up my contribution.

More precisely, I will provide you with code to render dominos with tiles of 0, 1, 2, or 3 pips. I will then ask you to carefully place this code into a Racket file and test the code. After that, are asked to scale the program up so that it will render dominos with tiles containing pips numbering 0 through 6, and to test the final domino rendering program.

Step by step, here is your charge:

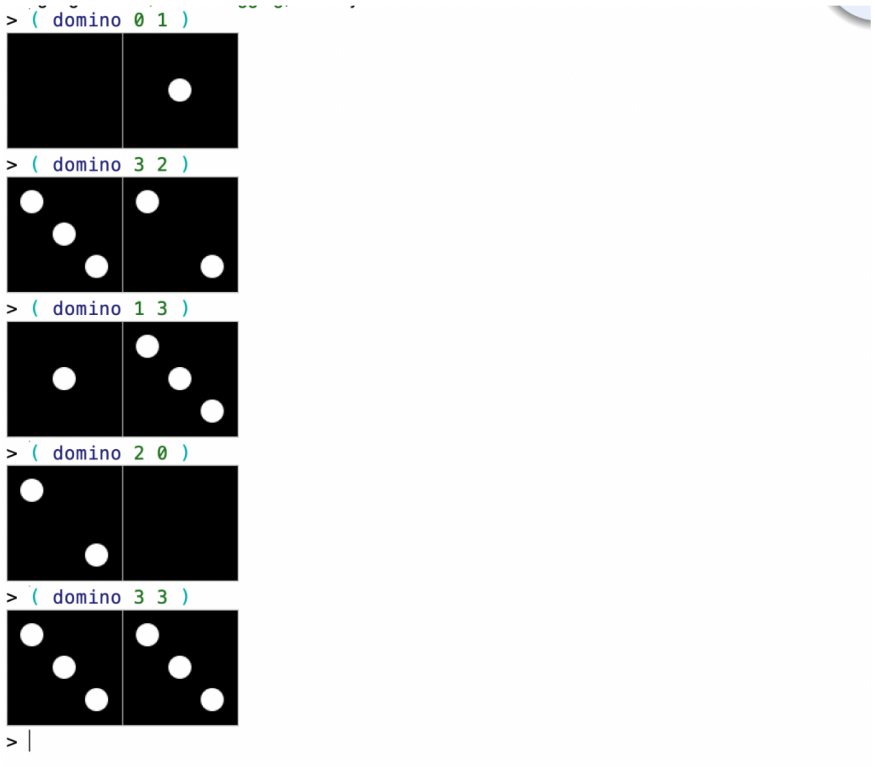
1. Please study the given domino rendering code, which renders dominos having tiles of 0, 1, 2, or 3 pips. Then, enter the code into a Racket file, and re-create the given demo for dominos having tiles containing 0, 1, 2, or 3 pips.
2. Extend the program so that it can render dominoes having tiles with pips numbering 0 through 4, and perform an appropriate test.
3. Extend the program so that it can render dominoes having tiles with pips numbering 0 through 5, and perform an appropriate test.
4. Extend the program so that it can render dominoes having tiles with pips numbering 0 through 6, and perform an appropriate test.
5. Re-create the final demo (see below).

Code for tiles with 0, 1, 2, and 3 pip dominos

```
#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
;
```

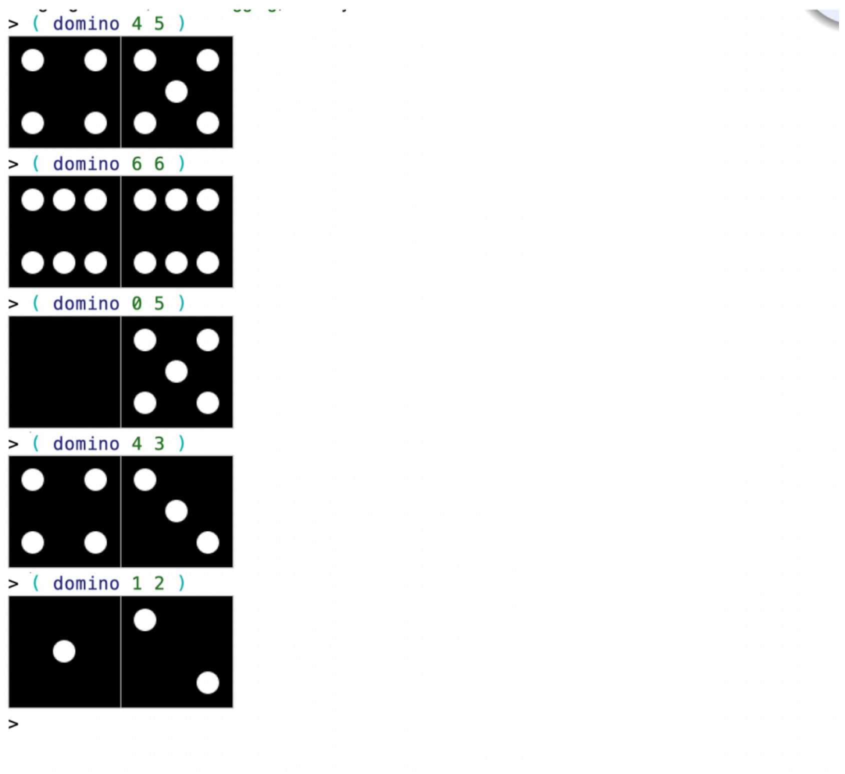
```
( define d ( * diameter-of-pip 1.4 ) )
( define nd ( * -1 d ) )

;------------------------------------------------------------------------------
; 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 ) )

;------------------------------------------------------------------------------
; 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 ) )

;------------------------------------------------------------------------------
; 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 )
  )
)
```

Demo for 0, 1, 2, and 3 pip dominos



```
> ( domino 0 1 )
> ( domino 3 2 )
> ( domino 1 3 )
> ( domino 2 0 )
> ( domino 3 3 )
> |
```

Final domino rendering demo



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

Contribution to the solution document

For the section of your solution document that corresponds to this part of your assignment, please include:

1. Your re-creation of the final domino rendering demo.
2. The code for your complete domino rendering program (the code that you were given to work with plus that which you contributed.)

Task 7: Creation

Programming constraint: For this part of your assignment, your are not permitted to use any form of of iterative construct. Rather, you are required to use recursion for any repetition that you would like to accomplish.

1. Please spend some time browsing the functionality for creating and manipulating images in the `2htdp/image` library.
2. Define a function called `my-creation` to produce an interesting image using some of the more involved functionality in the `2htdp/image` library. Perhaps use some of the more elaborate `overlay` commands (maybe `overlay/offset` or `overlay/xy`), or maybe some of the rotating, scaling, flipping, cropping, and framing functionality. (Recall that the "2htdp Image Documentation" section of Racket Lesson #3 references the main functionality associated with this library.)

Contribution to the solution document

For the section of your solution document that corresponds to this part of your assignment, please include:

1. An image of a demo that features you creation.
2. The code for your creation.

Additional Notes on your Solution Document

Craft a nicely structured document that contains:

1. A nice title, indicating that this is your second Racket assignment.
2. A nice learning abstract, which artfully says something about the 2htdp/image library, the generation of visual permutations on a set of three discs, recursive programming, number sequences, renderings that channel a couple of famous modern artists, and a visual creation of your own.
3. A section for Task 1, the task involving colorful permutation tracts of houses.
4. A section for Task 2, the task involving virtual dice.
5. A section for Task 3, the task involving number sequences.
6. A section for Task 4, the task involving Hirst dots.
7. A section for Task 5, the task involving your Stella inspired images.
8. A section for Task 6, the task involving dominos.
9. A section for Task 7, the task featuring your own visual creation.

Then post your document, in **pdf** format, to you web work site.

Due Date

Friday, September 16, 2022.