Racket Assignment #3: Recursions in Racket

What's It All About?

This assignment affords you an opportunity to do some relatively simple recursive programming. **Constraint: No nonrecursive iterative constructs are allowed.**

Task 0: Template for Your Solution Document

Craft a template for your solution document for this programming assignment, one that is structured like the one that I am providing along with this assignment.

Task 1: Counting Down, Counting Up

Simply perform the following tasks:

1. Define a recursive function called `count-down` taking one nonnegative integer parameter which prints the integers from the given value down to 1, one number per line.
2. Define a recursive function called `count-up` taking one nonnegative integer parameter which prints the integers from 1 up to the given value, one number per line.
3. Craft a demo that is identical to the given demo.
4. Add the functions to the appropriate location of your solution document. Add the demo to the appropriate location of your solution document.

The Demo

```
> ( count-down 5 )
5
4
3
2
1
> ( count-down 10 )
10
9
8
7
6
```

```
5
4
3
2
1
> ( count-down 20 )
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
> ( count-up 5 )
1
2
3
4
5
> ( count-up 10 )
1
2
3
4
5
6
7
8
9
10
> ( count-up 20 )
1
2
3
4
5
6
7
8
9
10
```

```
11
12
13
14
15
16
17
18
19
20
>
```

Task 2: Triangle of Stars

Simply perform the following tasks:

1. Define a recursive function called `triangle-of-stars` taking one nonnetagive integer parameter which displays a triangle of stars, in a manner consistent with the accompanying demo.
2. Mimic the accompanying demo.
3. Post the code and the demo in the designated locations of the accompanying template.

The Demo

```
> ( triangle-of-stars 5 )
*
* *
* * *
* * * *
* * * * *
> ( triangle-of-stars 0 )
> ( triangle-of-stars 15 )
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
>
```

Task 3: Flipping a Coin

Simply perform the following tasks:

1. Define a recursive function called `flip-for-difference` taking one nonnetagive integer parameter which flips a virtual coin until the number of heads and the number of tails differ by the given value, displaying the outcomes of each flip along the way, all on one line.
2. Mimic the accompanying demo, in terms of forms typed in at the prompt, knowing that the details of the output will vary from those appearing in the demo. Note that the function is called 4 times with the goal of 1, 6 times with the goal of 2, 6 times with the goal of 3, and 8 times with the goal of 4.
3. Add the required function, and any auxiliary functions that you write to support it, to the appropriate location of your solution document. Add the demo to the appropriate location of your solution document.

Suggestion: It is sometimes the case that you want to write a "helper" function from within a function that you intend to define, simply to change your problem into one of defining a function with a different configuration of parameters. In the situation at hand, you might want to introduce a helper function of two parameters, one of which summarizes the difference in heads and tails "so far" (perhaps encoding that difference by means of positive numbers for more heads and negative numbers for more tails).

The Demo

```
> ( flip-for-difference 1 )
t
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
h
> ( flip-for-difference 1 )
h
> ( flip-for-difference 2 )
hh
> ( flip-for-difference 2 )
thhthhthhtthhh
> ( flip-for-difference 2 )
hthh
> ( flip-for-difference 2 )
thtt
> ( flip-for-difference 2 )
tt
> ( flip-for-difference 2 )
ththtt
> ( flip-for-difference 3 )
hthhthtthhhh
> ( flip-for-difference 3 )
hthhhh
> ( flip-for-difference 3 )
hhh
> ( flip-for-difference 3 )
ttt
```

```
> ( flip-for-difference 3 )
thtththhthhtttt
> ( flip-for-difference 3 )
hhththttttt
> ( flip-for-difference 4 )
tttt
> ( flip-for-difference 4 )
hhhtthttthttthhthhtthhhhthhh
> ( flip-for-difference 4 )
hthhhththh
> ( flip-for-difference 4 )
hthtthhhthhthh
> ( flip-for-difference 4 )
httttt
> ( flip-for-difference 4 )
hhtthttttthtt
> ( flip-for-difference 4 )
hhthhh
> ( flip-for-difference 4 )
tthhhthhthttththhthhhtthttthhtthtthtttt
>
```

Task 4: Laying Down Colorful Concentric Disks

This problem features one of the great ideas in art, variations on a theme. In the hands of Mozart, this was, indeed, an art. What we will be doing is just a bit of play. Still, the idea of variations on a theme will be present in this problem.

You are asked to write three recursive functions, each with the same essential structure.

In terms of contributions to your solution document, simply add code and a demo for each of the three functions, as called for in the template.

CCR: Laying Down Disks of Random Colors

Simply perform the following tasks:

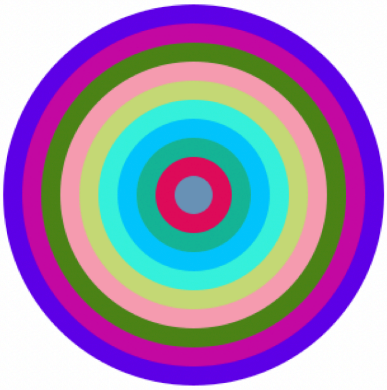
1. Write a function called `ccr` which:
   (a) Takes two numeric arguments, the radius of a bounding circle, and the difference in radius between one concentric circle and the next.
   (b) Returns an image consisting of concentric disks of random colors, in a manner consistent with the accompanying demo.
2. Mimic the accompanying demo, in terms of forms typed at the prompts.

Demo

```
> ( ccr 100 50 )
```

```
> ( ccr 50 10 )
```

```
> ( ccr 150 15 )
```



```
>
```

CCA: Laying Down Disks of Alternating Colors

Simply perform the following tasks:

1. Write a function called `cca` which:
   (a) Takes two numeric arguments, the radius of a bounding circle, and the difference in radius between one concentric circle and the next, and two color arguments.
   (b) Returns an image of concentric disks of alternating colors in a manner consistent with the accompanying demo.
2. Mimic the accompanying demo, in terms of forms typed at the prompts.

Demo

```
> ( cca 160 10 'black 'white )
```

```
> ( cca 150 25 'red 'orange )
```

```
>
```

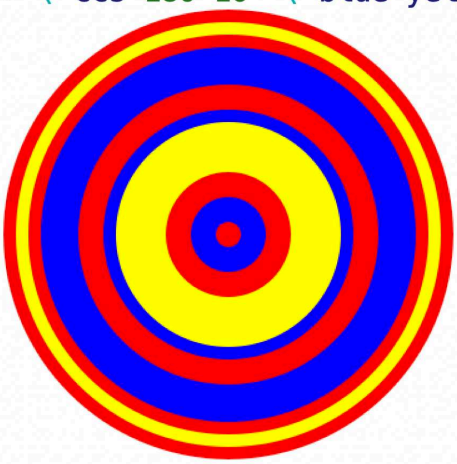
CCS: Laying Down Disks of Sampled Colors

Simply perform the following tasks:

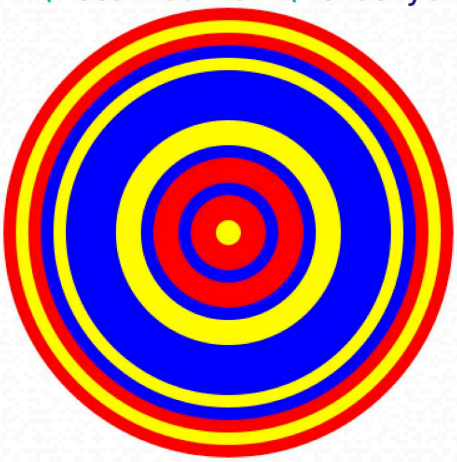
1. Write a function called `ccs` which:
   (a) Takes two numeric arguments, the radius of a bounding circle, and the difference in radius between one concentric circle and the next, and a third argument denoting a list of colors.
   (b) Returns an image of concentric disks of randomly sampled colors in a manner consistent with the accompanying demo.
2. Mimic the accompanying demos, in terms of forms typed at the prompts.

Demo 1

```
> ( ccs 180 10 '( blue yellow red ) )
```



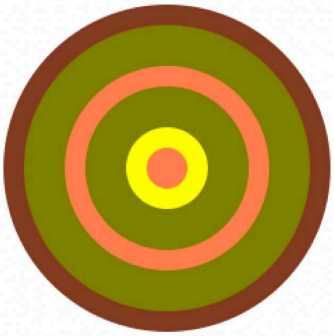
```
> ( ccs 180 10 '( blue yellow red ) )
```



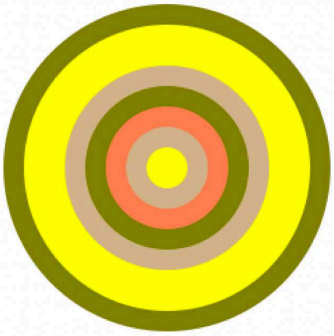
```
>
```

Demo 2

```
> ( ccs 120 15 '( brown coral goldenrod yellow olive tan ) )
```



```
> ( ccs 120 15 '( brown coral goldenrod yellow olive tan ) )
```



```
>
```

Task 5: Variations on Hirst Dots

For this problem, you will be asked to channel some of Racket Lesson #2, the part pertaining to rendering grids of "tiles". And you will want to lift bits your work from the previous problem on this programming assignment.

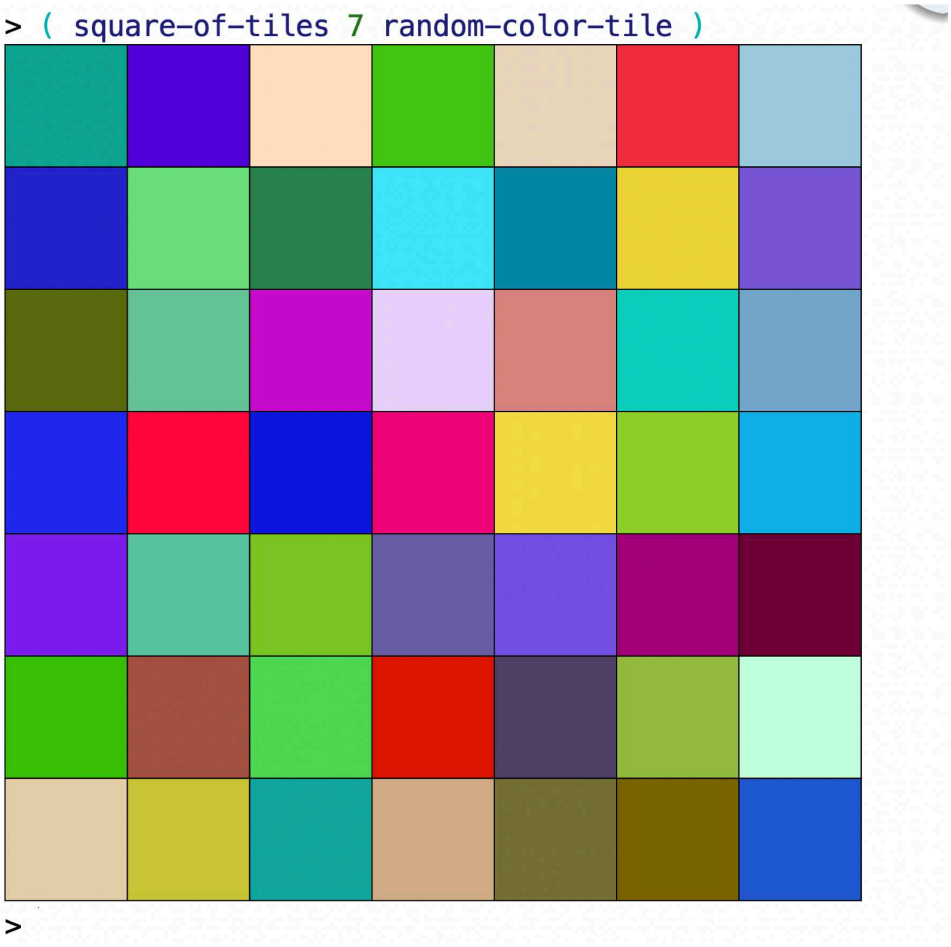
Throughout, please remember the chief constraint: **Recursion is the only form of iteration allowed.** One way to get good a thing is to ban the use of other things.

Please start by replicating the development of the program called `square-of-tiles` that was presented in Racket Lesson #2. Also, please be sure to implement the `random-color-tile` from that lesson.

As you complete each of the five tasks for this part of your programming assignment, please add the required demo to your solution document in appropriate locations. After you have complete the five tasks, please add the code used to generate the five demos to the appropriate location in your solution document.

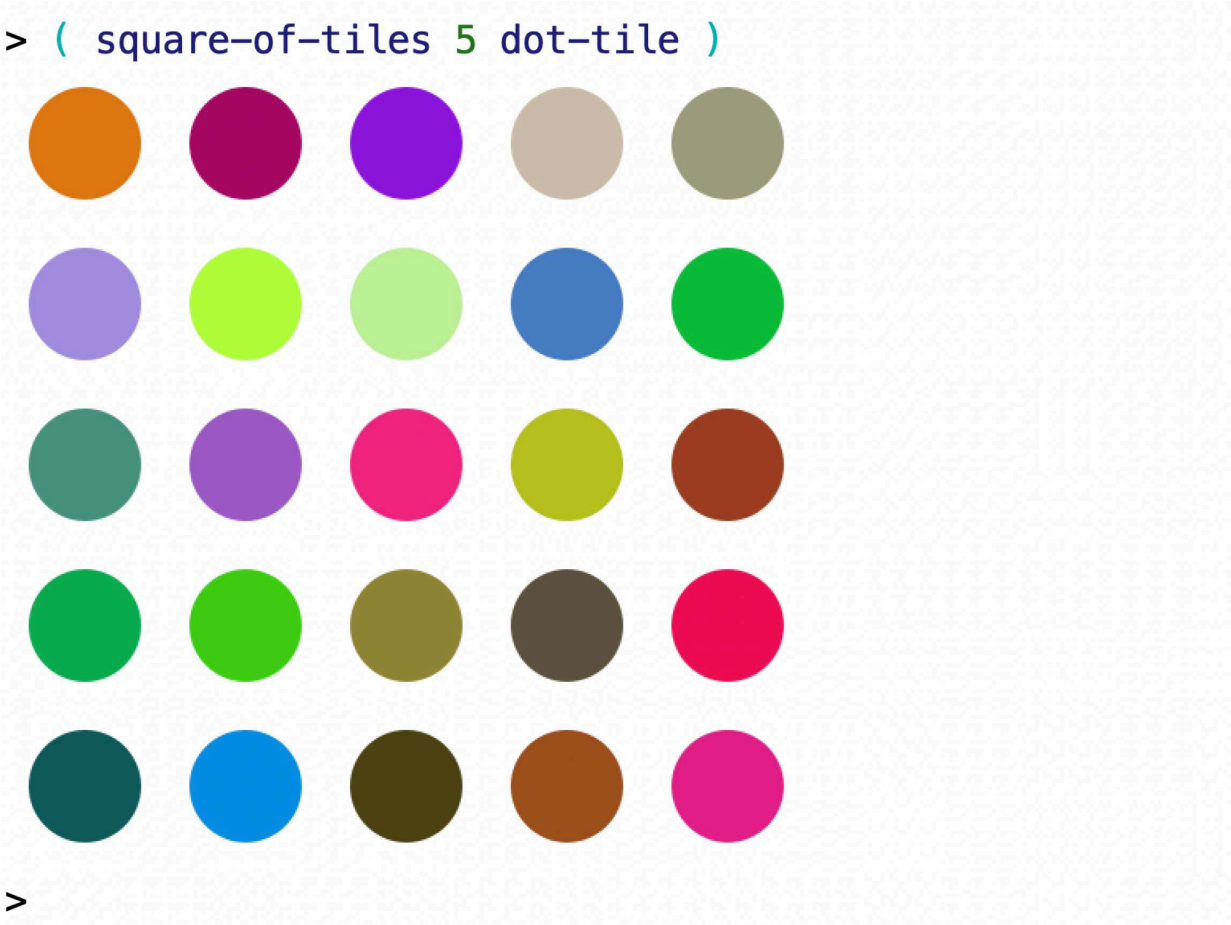
Solid Randomly Color Tiles with Borders

Mimic the following demo, in terms of the form presented to the prompt:



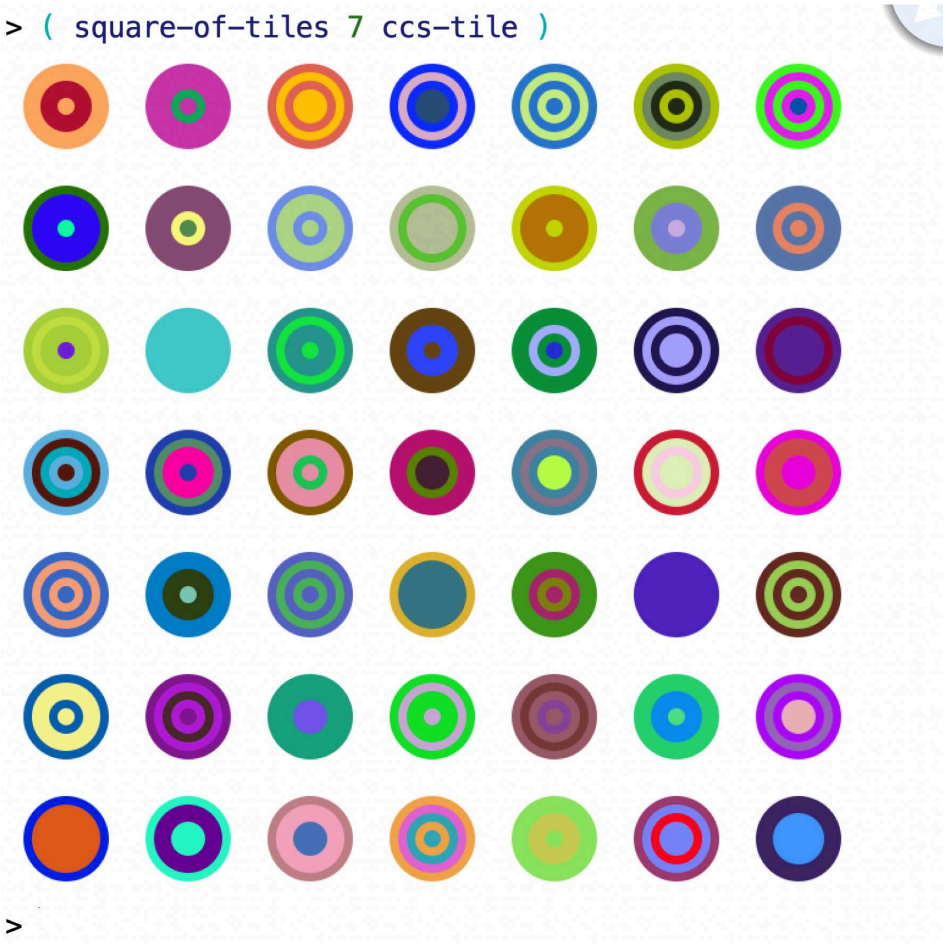
Hirst Dots

Write a function called “dot-tile” to generate a 100x100 tile with a randomly colored dot of diameter 70 on a borderless white background. Tile generator in hand, mimic the following demo, in terms of the form presented to the prompt:



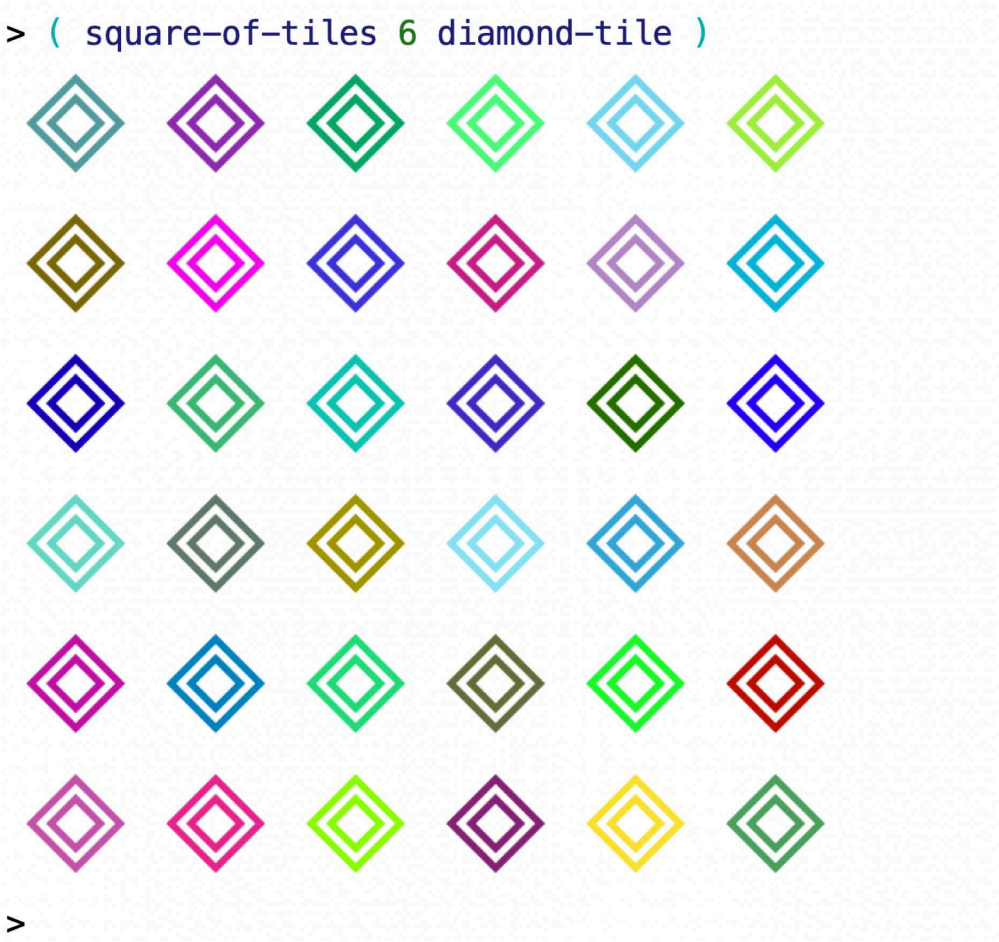
CCS Dots

Write a function called `ccs-tile` to generate a 100x100 tile containing an image consisting of concentric circles sampled from three randomly generated colors on a white background, where the bordering circle is of radius 35 and consecutive circles differ in radius by 7. Tile generator in hand, mimic the following demo, in terms of the form presented to the prompt:



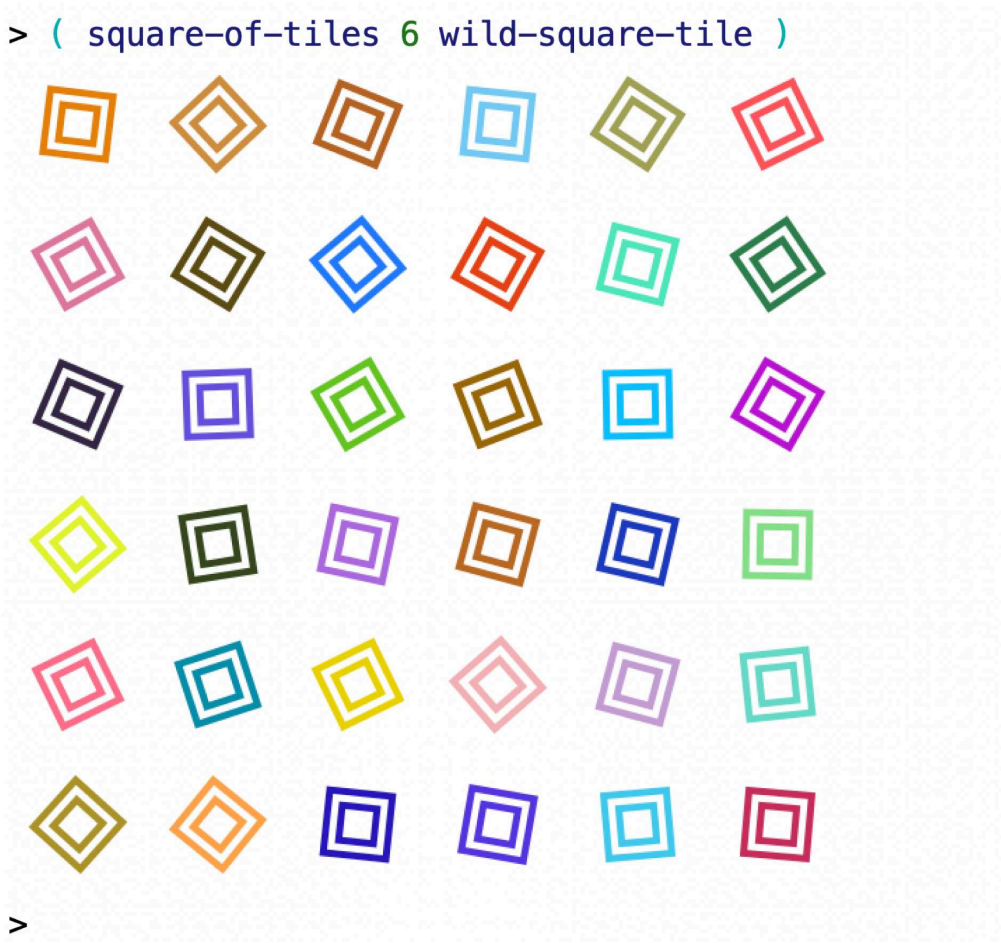
Nested Diamonds

Write a function called `diamond-tile` to generate a 100x100 tile containing an image consisting of two nested diamonds of the same randomly generated color on a white background, Rather than specifying tile sizes/thicknesses of the diamonds, I will leave it to you to experimentally determine some reasonable sizes. Tile generator in hand, mimic the following demo, in terms of the form presented to the prompt, and also in terms of the nature of the diamond tiles:



Unruly Squares

Write a function called `wild-square-tile` to generate a 100x100 tile containing an image consisting of two nested squares of the same randomly generated color on a white background, subject to the constraint that the nested squares are randomly rotated to some degree. Rather than specifying tile sizes/thicknesses of the squares, I will leave it to you to experimentally determine some reasonable sizes. Tile generator in hand, mimic the following demo, in terms of the form presented to the prompt, and also in terms of the nature of the diamond tiles:



Task 6: Posting Your Solution Document

Please post your solution document to your web work site by **Tuesday, February 28, 2023**.