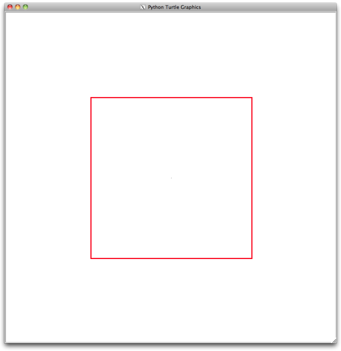
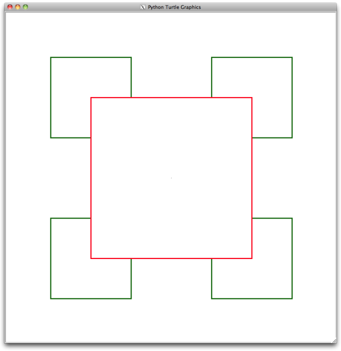
Problem-Based Intro. to Computer Science
Squares (Lab)

Problem

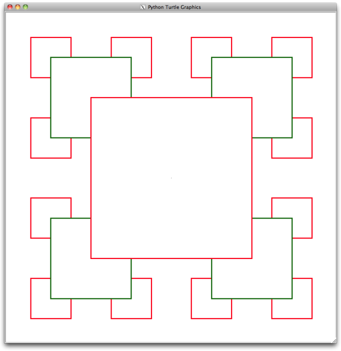
The problem this week is to develop a program that generates a recursive pattern of squares. When run, the program must *prompt* for the recursion `depth`. If the `depth` is 0, then the program should draw one red square (see picture below). If the `depth` is 1, then the program should draw the same red square and also, at each of the four corners, it should draw half-sized dark-green 3/4 squares that omit the overlapping corner (see picture below). If the `depth` is 2, then the program should draw the same figure as when `depth` is 1 and also at each of the 12 corners of the half-sized 3/4 squares, it should draw quarter-sized red 3/4 squares that omit the overlapping corner (see picture below). As the `depth` increases, the colors of the squares should toggle between red and dark-green. The program should draw the figure in the standard canvas window. You may assume that `depth` will always be non-negative.



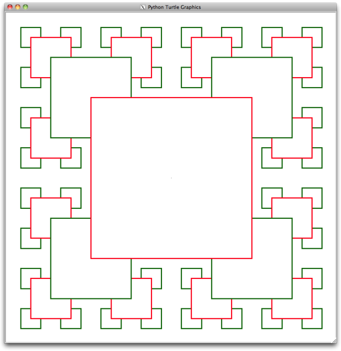
`depth == 0`



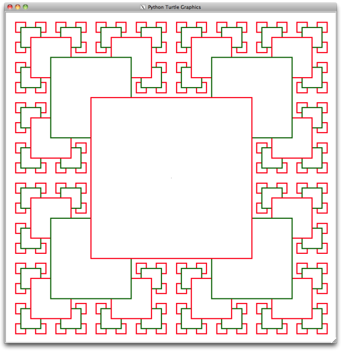
`depth == 1`



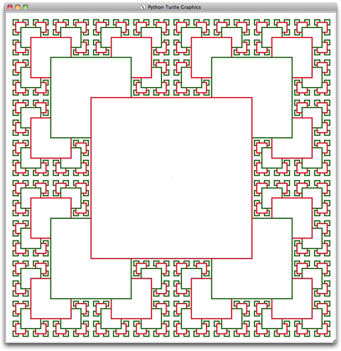
`depth == 2`



`depth == 3`



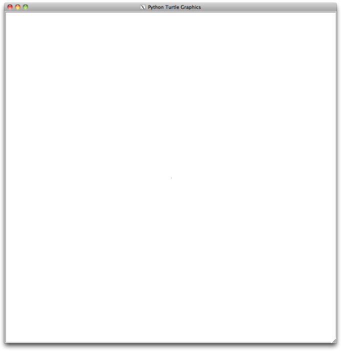
`depth == 4`



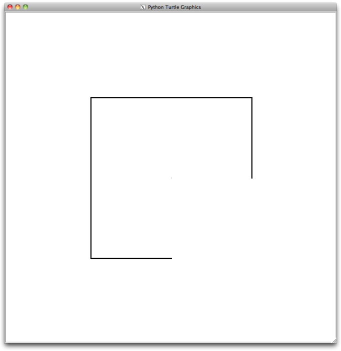
`depth == 5`

Problem-Solving Session (20%)

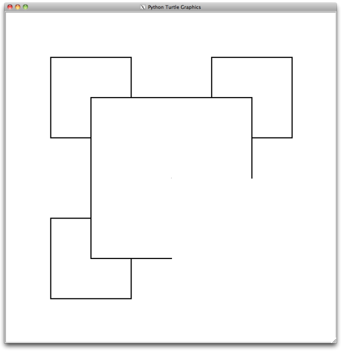
For the problem-solving session, we consider a simpler problem: we draw 3/4 squares at all depths (and draw nothing at `depth == 0`) and do not change the drawing color at each depth. (You will need to implement the proper full and 3/4 squares and color changes in your individual lab solution.) Compare the pictures below to the pictures on the first page to understand the difference.



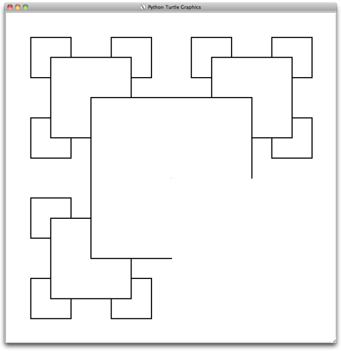
`depth == 0`



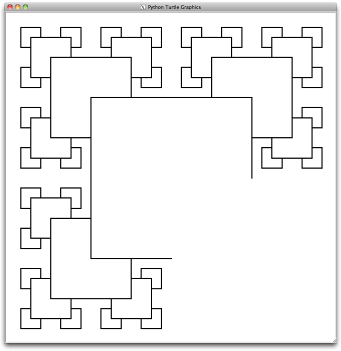
`depth == 1`



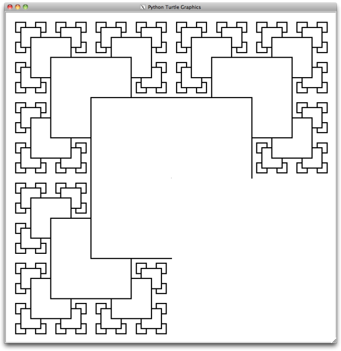
`depth == 2`



`depth == 3`



`depth == 4`



`depth == 5`

You will work in teams of four to five students (as determined by your instructor) to accomplish the following tasks for the simplified problem:

1. Describe, in English, how the figures on this page differ when `depth == 0`, `depth == 1`, and `depth == 2`.
2. Write pseudocode for a function producing the drawing for `depth == 0`.
3. Write pseudocode for a function producing the drawing for `depth == 1`.
4. Write pseudocode for a function producing the drawing for `depth == 2`.
5. Write pseudocode for a recursive function producing the drawing for any `depth`.
6. Show the behavior of your pseudocode from part 4 for `depth == 2` using an execution diagram.
7. Explain, in English, how you can use the solution for the simplified problem to solve the original problem.

At the end of problem-solving session, hand in your team's work. Be sure that all team member's names are legible and that all task items are numbered.

Implementation (80%)

You will work individually to accomplish the following tasks:

1. 25%: Write a program that draws the *simplified-squares* figure (from the problem-solving session). Your program should be written in a file called `squares_simple.py` that is executed by `python3 squares_simple.py`.
2. 40%: Write a program that draws the *squares* figure (from page 1). Your program should be written in a file called `squares.py` that is executed by `python3 squares.py`.
3. 15%: In a plain-text file called `squares.txt`, answer the following:
   - Explain how the *simplified-squares* solution is used to implement the *squares* solution.
   - What values of `depth` should you test and why?
   - Run your `squares.py` program with `depth == 4`. Report the amount of time (in seconds) it takes your program to run.

Constraints Be sure that your submissions satisfy the following constraints:

- Both programs must *prompt* for the recursion `depth`, draw the appropriate figure so that it fits in the canvas window (see below), and *pause* until the user presses the `ENTER` key. Furthermore, the turtle should be hidden and not visible in the figure when waiting for the user to press the `ENTER` key. It is recommended that both programs initialize the canvas with the following, in a dedicated initialization function:

  ```
  turtle.setup( 600, 600 )
  turtle.setworldcoordinates( -202, -202, 202, 202 )
  turtle.speed( 'fastest' )
  ```

  This creates a 600 × 600 window with lower-left coordinate $(-202, -202)$ and upper-right coordinate $(202, 202)$ and draws as fast as possible. Drawing the figures such that the largest square is centered with side length 200 will fit well in this canvas window.
- Both programs must use recursion to draw the figures.
- Both programs may not use global variables.
- There is a `"""docstring"""` for each function. Each function's `"""docstring"""` must include a one-sentence English description of the function and must include a description of the *pre-conditions* assumed and the *post-conditions* guaranteed by the function. For example, if a function leaves the turtle pen-up and facing North, then the function's `"""docstring"""` should include a comment like the following:

  ```
  post-conditions: turtle is pen-up and facing North, ...
  ```
- There is a `"""docstring"""` for each program. Each program's `"""docstring"""` must include your full name and a one-sentence English description of the program.

Submission

Zip the program files (`squares_simple.py` and `squares.py`) and the design file (`squares.txt`) into a file called `lab03.zip`; to do so, use the following command:

```
zip lab03.zip squares_simple.py squares.py square.txt
```

Submit the `lab03.zip` file to the MyCourses dropbox for this week's lab assignment.