15-110: Principles of Computing, Spring 2018

Lab 9 – Thursday, March 29

Goals

This lab is intended to develop your understanding of doing simple graphics with Python. We will be using the module tkinter to create simple computer graphics on the screen. We will also be using the module random to generate random colors for the shapes we draw. After we learn how to draw rectangles, we introduce the concept of a fractal in the last question and make you observe the recursive nature of fractals. That part is mainly intended to make you think recursively and enjoy a beautiful application of recursion.

When you are done with this lab, you should be able to do the following:

1. Initialize a canvas of a certain size
2. Explain how the coordinate system works
3. Draw rectangles on the canvas, possibly with borders, and possibly filled with color
4. Use the module random to generate random integers
5. Use the module random to choose random elements from a given list
6. Explain the effect of using a seed for a pseudorandom number generator

Part 1: TA Demonstrations (Graphics in Python)

- Review the graphics coordinate system
- Review graphicsdemo.py
- Review how to use randint

NOTE: If the graphics module is not installed on the Andrew servers (try to import tkinter in interactive mode), then allow students to use their own Python 3 installation on their laptop if they used the default installation. Students may partner together in this case. Additionally, if the TA has a laptop, he or she can lead the class together to work on the problems on the screen.

Self-Directed Activities

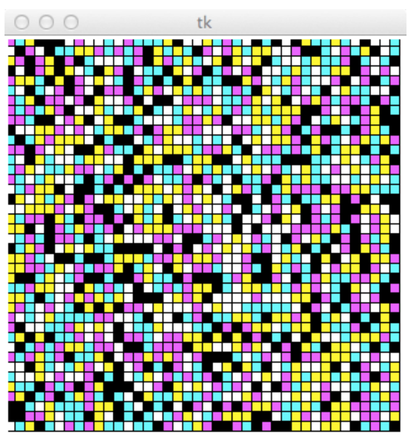
1. Random colorful boxes

   Write a Python function `random_boxes()` in the file `random_boxes.py` that creates a window of size 320 by 320 and draws a 40 by 40 grid of 8 by 8 squares, each colored a random color from black, white, yellow, cyan, and magenta. Be sure to include the correct `import` statements before your function in the file. Look at `graphicsdemo.py` from class: all you need to do is edit it a little!

   Your exact color results will vary since we are using a random number generator and we do not expect you to seed the random number generator to fix the sequence of numbers that it will generate.

   Sample usage:

   ```
   > python3 –i random_boxes.py
   >>> random_boxes()
   ```

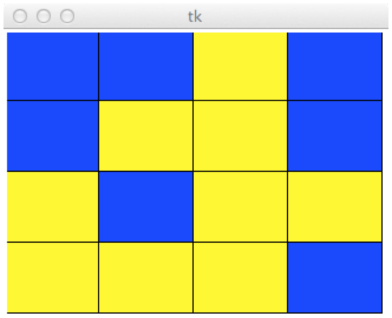
   (Note how the left column and row get cut off a little due to the graphics module. Can you tweak the code so you can see the whole picture? Hint: Make the window a little bigger, and add a small offset to each coordinate to shift the picture to the right and down.)

2. Matrix colors

   Write a function `data_display(matrix)` that takes a 4 by 4 matrix (list of lists) as its parameter and displays a 4 by 4 grid of 80 by 60 rectangles in a window of size 320 (width) by 240 (height) based on the data in the input matrix . If the value in a matrix cell is odd, draw a yellow rectangle in its grid location in the window and if the value in a matrix cell is even, draw a blue rectangle in its grid location in the window.

Sample usage:

```
> python3 –i data_display.py
>>> matrix = [[0,2,1,4], [4,5,3,8], [9,4,7,1], [5,1,7,0]]
>>> data_display(matrix)
```



3. Fractals!

The image on the next page is a Sierpinski's Triangle (albeit, rotated and with a different angle than the canonical form). It is a fractal; that is, it is "a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole." In this case, you will notice that, if you look at the lower-right corner, it looks the same (down to the level of one pixel) at different levels of magnification.

Starting with a completely filled-in square, this Sierpinski's Triangle can be produced with the following procedure:

I. Return None if the image is too small to be visibly subdivided.
II. Otherwise, consider the square being divided up into four quadrants (squares).
III. Draw a square (rectangle with equal sides) in the upper-left quadrant.
IV. Recursively draw a Sierpinski triangle in each of the other three quadrants.

Finish the following incomplete Python implementation of this algorithm and save it as `triangle.py`. (Cut and paste the code into this file.) **You only need to supply the parameters to the recursive procedure calls. You should not add any other additional code.** If you do this correctly, you will be able to reproduce the image shown above, by calling start_triangle(256, "blue") after loading your source file. You should note that when you specify drawing coordinates for the canvas, the top left corner is the origin (0,0), x increases as you go from left to right, but y increases as you go from top to bottom.

HINT: You need to figure out the top-left corner and size for each of the other three quadrants and fill them in for the parameters for the three recursive calls.

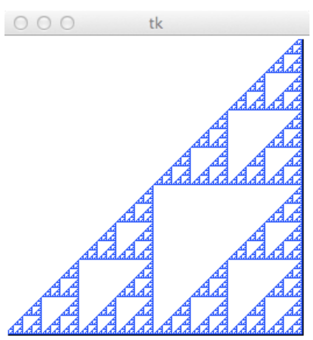
```
from tkinter import *

def triangle(c, x, y, size):
    # create a triangle at top left (x,y) of given size on canvas c
    # by creating 3 subtriangles recursively
    if size < 2:     # size = 1 means 1 pixel width, stop there
        return None
    half = size // 2
    # draw white square starting at top left (x,y) of half size on canvas c
    c.create_rectangle(x, y, x+half, y+half, width=0, fill="white")
    # recursively repeat the triangle
    # for each of the other three quadrants on canvas c
    triangle(c,      ,       ,       ) # Fill in the missing parameters
    triangle(c,      ,       ,       ) # Fill in the missing parameters
    triangle(c,      ,       ,       ) # Fill in the missing parameters

# This function initializes a drawing canvas,
# then calls the recursive function triangle to do all the work
def start_triangle(size, color):
    window = Tk()
    c = Canvas(window, width=size, height=size)
    c.pack()
    c.create_rectangle(0, 0, size, size, fill=color)
    triangle(c, 0, 0, size)  # draw a Sierpinski triangle top left at (x,y)
                             # and with height and width of size on canvas c
```

Usage:

```
> python3 –i triangle.py
>>> start_triangle(256, "blue")
```



Submission

When you finish the lab, you should be inside the `lab9` folder, which is inside the `private/15110` directory. When you type `ls` and press the Enter key, you should see the following files: `random_boxes.py`, `data_display.py`, and `triangle.py`. Once you see all files, please type `cd ..` to move up one folder and press the Enter key. Then, zip your `lab9` folder by typing `zip –r lab9.zip lab9` and you should see a `lab9.zip` file in the current folder if you type `ls`. Please submit the zipped file `lab9.zip` on Autolab under 'lab 9'.