LABS #3 & 4 – Graphics

This assignment is in two related parts, the first will be counted as Lab #3 and the second part as Lab #4 (both are 10 points). The first part is to build and test a basic graphics system based on the code in the book, and the second part is to write a special program that can only generate graphics for its output.

PART #1 – Lab #3

You will need to create a library file called **`Graphics.py`** that contains the following code from the Computer Science Companion (4TH edition):

0. **PUT YOUR NAME IN A COMMENT AT THE VERY TOP OF THE FILE!!!**
1. The `Distance` function from page 382 (don't forget to import the math library)
2. All of the graphics code from pages 404-407:
   - The color definitions (`black`, `blue`, etc.)
   - `getRed`, `getGreen`, `getBlue`
   - `makeEmptyPicture`
   - `getWidth`, `getHeight`
   - `PixelIndex`
   - `setPixel`, `getPixel`
   - `HorizontalLine`, `VerticalLine`
   - `WriteString`, `WriteBytes` (for Python 3) , `WriteBMP`
3. The `addLine` function from the ***top*** of page 408
4. The circle functions from page 411:
   - `BresenhamCircle`, `BresenhamFilledCircle`

Save this file as **`Graphics.py`** in your Python directory. This is a lot of code to type in, and it is likely that you will make mistakes. To quickly test if the functions basically work, "run" the file (it won't do anything), then follow the following script by typing in the commands in black. If everything works, you should get the responses from Python as shown in blue:

```
>>> Canvas = makeEmptyPicture(320, 200, gray)
>>> Canvas
{'Width': 320, 'Height': 200, 'Default': [128, 128, 128]}
>>> setPixel(Canvas, 100, 50, green)
>>> Canvas
{'Width': 320, 'Height': 200, 'Default': [128, 128, 128], 5000100: [0, 255, 0]}
>>> getPixel(Canvas, 100, 50)
[0, 255, 0]
>>> setPixel(Canvas, 100, 50, gray)
>>> getPixel(Canvas, 100, 50)
[128, 128, 128]
>>> Canvas
{'Width': 320, 'Height': 200, 'Default': [128, 128, 128]}
>>>
```

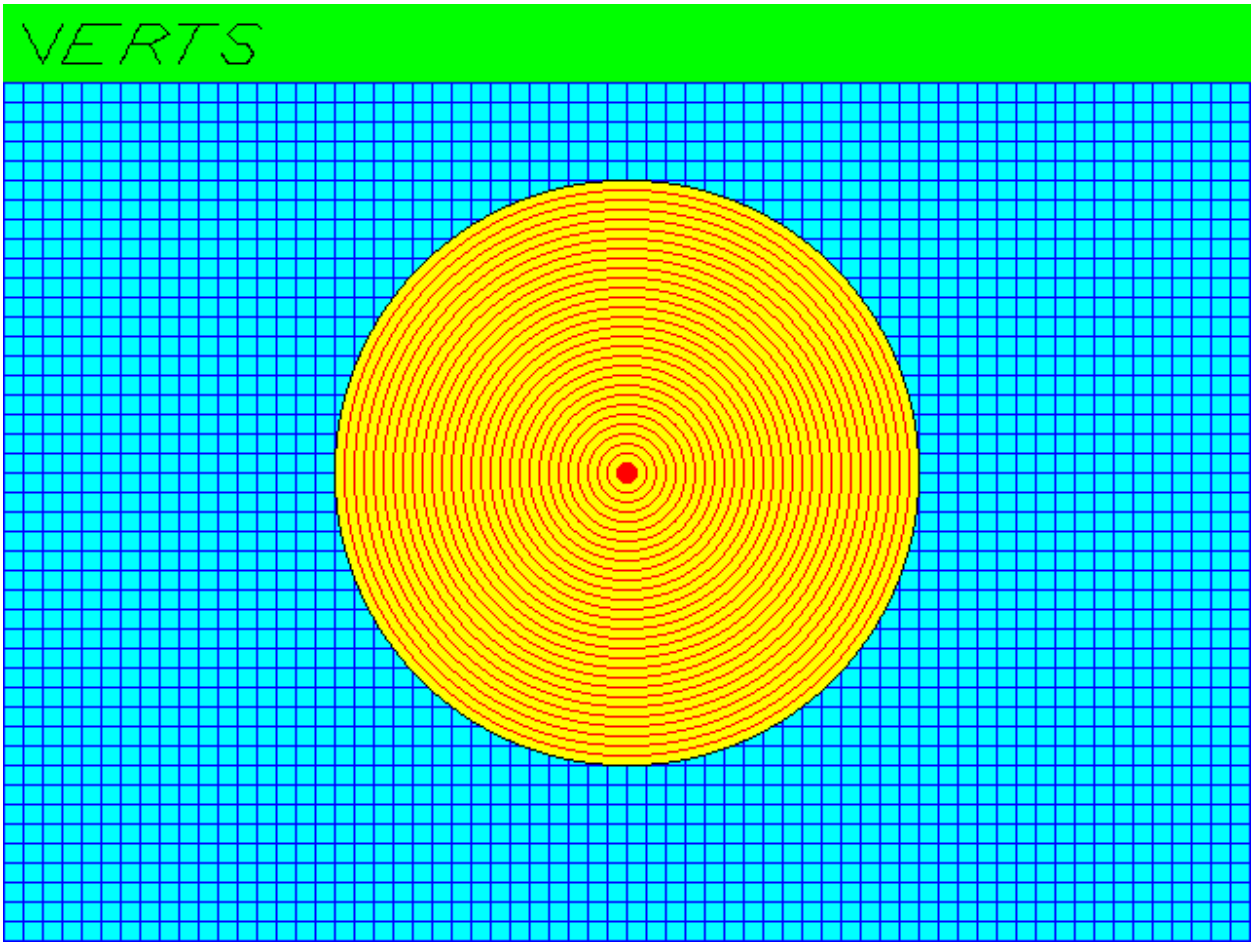
This is a very basic test of the routines, and does not exercise all of the functions. For a much more rigorous test, download the `TestGraphics.py` program from the class site. Store that file in the ***same folder*** as your `Graphics.py` file.

I ***strongly*** recommend that you ***download*** the `TestGraphics.py` program from the class site. However, if you cannot do so, here and on the next page is the source code for that file:

```
#-------------------------------------------------------------
# TEST PROGRAM FOR GRAPHICS LIBRARY
#
# Copyright (C) December 1, 2019 -- Dr. William T. Verts
#-------------------------------------------------------------

from Graphics import *

#-------------------------------------------------------------
# Build the canvas and define variables for later use.
#-------------------------------------------------------------

Canvas = makeEmptyPicture(641, 481, cyan)
XMid = getWidth(Canvas)  // 2
YMid = getHeight(Canvas) // 2
Radius = 150

#-------------------------------------------------------------
# Draw horizontal green lines from screen left to screen
# right for the first 40 lines of the image.
#-------------------------------------------------------------

for Y in range(40):
    HorizontalLine(Canvas, 0, getWidth(Canvas)-1, Y, green)

#-------------------------------------------------------------
# Draw horizontal blue lines from screen left to screen
# right, starting at line 40 and stepping down 10 rows at
# a time.
#-------------------------------------------------------------

for Y in range(40, getHeight(Canvas), 10):
    HorizontalLine(Canvas, 0, getWidth(Canvas)-1, Y, blue)

#-------------------------------------------------------------
# Draw vertical blue lines from line 40 to screen bottom,
# starting at screen left and stepping right 10 columns at
# a time.
#-------------------------------------------------------------

for X in range(0, getWidth(Canvas), 10):
    VerticalLine(Canvas, X, 40, getHeight(Canvas)-1, blue)

#-------------------------------------------------------------
# Draw a big yellow filled circle at screen center, with
```

```
# a bunch of red circle outlines starting at radius 10 and
# stepping by 5 per circle.  Finally, draw a red spot at
# the center, and draw a black outline.
#-------------------------------------------------------------

BresenhamFilledCircle(Canvas, XMid, YMid, Radius, yellow)
for R in range(10, Radius, 5):
    BresenhamCircle(Canvas, XMid, YMid, R, red)
BresenhamFilledCircle(Canvas, XMid, YMid, 5, red)
BresenhamCircle(Canvas, XMid, YMid, Radius, black)

#-------------------------------------------------------------
# Draw my signature in the green area.
#-------------------------------------------------------------

addLine(Canvas, 10, 10, 20, 30)     # V
addLine(Canvas, 20, 30, 30, 10)
addLine(Canvas, 40, 10, 30, 30)     # E
addLine(Canvas, 40, 10, 60, 10)
addLine(Canvas, 35, 20, 45, 20)
addLine(Canvas, 30, 30, 50, 30)
addLine(Canvas, 70, 10, 60, 30)     # R
addLine(Canvas, 70, 10, 80, 10)
addLine(Canvas, 65, 20, 75, 20)
addLine(Canvas, 80, 10, 85, 15)
addLine(Canvas, 85, 15, 75, 20)
addLine(Canvas, 75, 20, 80, 30)
addLine(Canvas, 90, 10,110, 10)     # T
addLine(Canvas,100, 10, 90, 30)
addLine(Canvas,120, 10,130, 10)     # S
addLine(Canvas,130, 10,132, 12)
addLine(Canvas,120, 10,115, 15)
addLine(Canvas,115, 15,120, 20)
addLine(Canvas,120, 20,125, 20)
addLine(Canvas,125, 20,128, 25)
addLine(Canvas,128, 25,125, 30)
addLine(Canvas,125, 30,115, 30)
addLine(Canvas,115, 30,112, 25)

#-------------------------------------------------------------
# Save the image to file.
#-------------------------------------------------------------

print ("Pixels in Canvas = ", getWidth(Canvas)*getHeight(Canvas))
print ("Items in Canvas = ",len(Canvas))

WriteBMP('TestGraphics.bmp', Canvas)
print ("All Done")
```

Run the `TestGraphics.py` program that I provide; it should create a bitmap graphics file in the same folder as `TestGraphics.py` and `Graphics.py`, called `TestGraphics.bmp` (about 900K in size). When you load that bitmap into a standard graphics program (whatever you've got on your computer), you should see the following image:



As you notice, the `TestGraphics.py` program calls nearly all of the functions in `Graphics.py`, so if there are any bugs (either syntax or logic) then either (1) `TestGraphics.py` will crash, or (2) the image will be different from what you see here. **Fix your bugs until you get the correct test image.**

When you are done, submit your `Graphics.py` program as **Lab #3**. We will test your submission by running `TestGraphics.py` using your version of `Graphics.py`, and will grade as follows:

A: -10 No student name in a comment at the top of the `Graphics.py` file.
B: -8 `TestGraphics.py` crashes inside your `Graphics.py` file.
C: -5 `TestGraphics.py` runs to completion but `TestGraphics.bmp` contains little if anything that is recognizable.
D: -3 The `TestGraphics.bmp` image is created but the geometry of something (lines, circles, colors, etc.) is wrong

Since you are basically typing in someone else's code (mine!) there is really no reason for there to be *any* errors, so the grading is very simple (and harsh).

PART #2 – Lab #4

This assignment will use the (now debugged) `Graphics.py` library to create an interesting image called a fractal. In particular, the fractal is called the "Sierpinski Gasket" by mathematicians. To create one in Python, you will need to build a graphic as follows:

0: Create a new program called `Sierpinski.py` and **put your name in a comment at the top of the file**. You will lose all credit if you don't. Save this program file in the same directory as the `Graphics.py` library you created in part #1.

1: Make sure to import everything from your `Graphics.py` library. Also, import the random number library as well.

2: Create a graphics canvas that is 640×480 pixels, with a white background.

3: Define three 2D points, `P0`, `P1`, and `P2`. Each one can either be represented as individual variables for X and Y (e.g., `P0X` and `P0Y`) or as a list of two numbers (e.g., `P0` is `[__, __]` where X is `P0[0]` and Y is `P0[1]`). How you decide this question will control how you write all of the remaining code in this program.

`P0` is defined as the coordinates of the **middle of the top row** of the image.

`P1` is defined as the coordinates of the middle of the **lower-left corner** of the image.

`P2` is defined as the coordinates of the middle of the **lower right corner** of the image.

4: Define a new 2D point `PP` which has the same initial X and Y values as `P0`. Note that if you choose to use the list model for points, that you should NOT do this as `PP = P0` as this does not make a copy of the list (it makes both variables point to the same list).

5: Create a `for`-loop that runs for 1000000 iterations (that's right: a million times). Use essentially any loop variable you want, as this variable is not important to the contents of the loop.

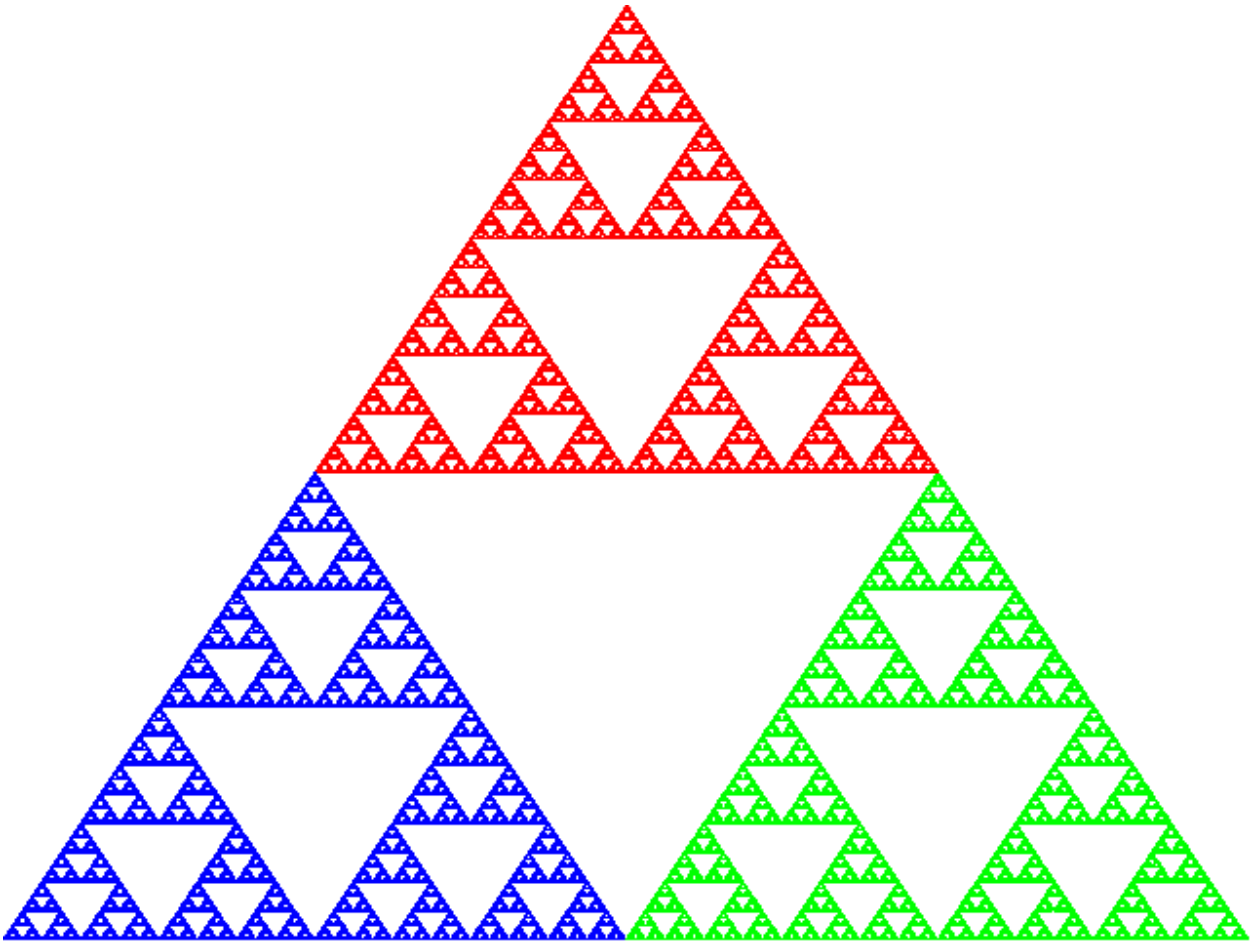
6: Inside the loop, pick a random integer `N` that is either 0, 1, or 2, but no other values.

7. Still inside the loop, if `N` is 0 then change `PP` to be the average of `PP` and `P0` (that is, the X value of `PP` will be the integer average of the X value of `PP` and the X value of `P0`, and similarly for Y). If `N` is 1 then `PP` will become the integer average of `PP` and `P1`, and if `N` is 2 then `PP` will become the integer average of `PP` and `P2`.

8. Still inside the loop, if `N` is 0 then set pixel `PP` to be red. If `N` is 1 set pixel `PP` to be green, and if `N` is 2 set pixel `PP` to blue.

9. Once the loop is complete, write the image to `Sierpinski.bmp`, and you are done. The bitmap should be in the same directory as your program.

Check to make sure that the `Sierpinski.bmp` file contains the fractal (as shown here). When your program generates the correct image, submit your source code as **Lab #4**.



We will run your `Sierpinski.py` program using a known-to-be-correct version of the `Graphics.py` library (<u>not</u> your `Graphics.py` that you submitted in Lab #3!). Scoring will be as follows:

A: -10 No student name in a comment at the top of the `Sierpinski.py` file.
B: -10 Program crashes before completion.
C: -8 `Sierpinski.py` runs to completion but `Sierpinksi.bmp` is not created.
D: -5 `Sierpinski.py` runs to completion and `Sierpinski.bmp` is created, but contains little if anything that is recognizable.
E: -3 `Sierpinski.py` runs to completion and `Sierpinski.bmp` is created, but the points are not in the correct places or colors are not correct.