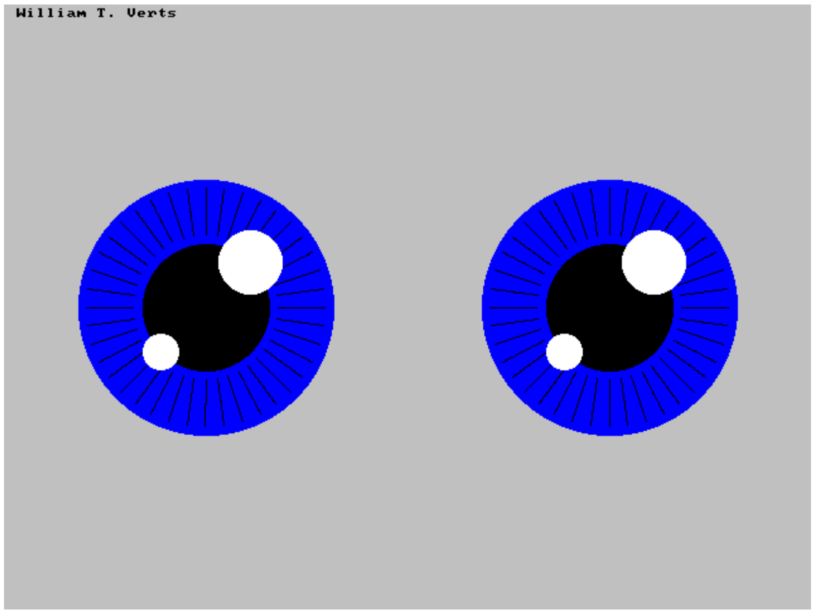
CMPSCI 119
LAB #4 – Anime Eyes
Professor William T. Verts

The goal of this Python programming assignment is to write your own code inside a provided program framework, with some new graphical and mathematical considerations. You must create a pair of staring Anime Eyes, which have an iris, a pupil, a starburst ring of dashes in the iris, and a pair of highlights. Your code must adapt to the width and height of the canvas, so that the eyes perfectly fit the screen and do not overlap any edges, for any arbitrary canvas size. You must also paint your own name in the upper left corner of the image. This is shown as follows:



In the Python environment, type in the following program code *exactly as you see it on the next two pages, including the comments*, and save the program as `Lab4.py` in your Python folder (make sure that `Graphics.py` and `Font8x8.py` are in the same folder). Alternatively, you can download the framework code from the class site.

Wherever you see my name in red in the code, replace it with *your own name*. As always, be very careful about indentation and capitalization. The blank space in each function is where you will be writing your own code later.

```
# William T. Verts - Lab #4 - Anime Eyes

from Graphics import *
from Font8x8 import *
import math

#---------------------------------------------------------------
# This function paints a bunch of black lines radially around
# <Xc,Yc> where R1 and R2 are the inner and outer radii and
# Segments indicates the number of lines.  Xc, Yc, R1, and R2
# are all allowed to be floats.
#---------------------------------------------------------------

def addStarburst (Canvas,Xc,Yc,R1,R2,Segments):


    return

#---------------------------------------------------------------
# This function paints ONE anime eyeball on the Canvas,
# centered at <Xc,Yc>.  The color of the iris is NewColor, the
# pupil is black, and the highlights are white.  The sizes and
# positions of the iris, pupil, starburst, and highlights are
# derived from center point <Xc,Yc> and the iris radius R.
#---------------------------------------------------------------

def Eyeball (Canvas,Xc,Yc,R,NewColor):


    return

#---------------------------------------------------------------
# This function CALLS Eyeball TWICE (once for each eye) to
# place the pair of eyeballs on the screen.  Stare must first
# determine the correct radius and center positions for each
# eye before calling Eyeball at all.
#---------------------------------------------------------------

def Stare (Canvas,NewColor):


    return

#---------------------------------------------------------------
# Plot a string S on the canvas where the lower left coordinate
# is at <X,Y> (X pixels from the left, Y pixels from the top).
#---------------------------------------------------------------

def PlotText (Canvas,X,Y,S,NewColor=black):


    return
```

```
#---------------------------------------------------------------
# Prompt the user for a color.  You MAY NOT change ANY code
# in the requestColor function!
#---------------------------------------------------------------

def requestColor(Message):
    while True:
        S = input(Message)
        S = S.lower()
        if (S == "red"):     return red
        if (S == "green"):   return green
        if (S == "blue"):    return blue
        if (S == "cyan"):    return cyan
        if (S == "magenta"): return magenta
        if (S == "yellow"):  return yellow

#---------------------------------------------------------------
# This function establishes the size of the canvas and the
# color of the anime eyes.  You MAY NOT change ANY code in
# the Main function except for your name instead of mine!
#---------------------------------------------------------------

def Main():
    W = int(input("Enter Width --- "))
    H = int(input("Enter Height --- "))
    C = requestColor("Enter Iris Color --- ")
    Canvas = makeEmptyPicture(W,H,[192,192,192])
    Stare(Canvas,C)
    PlotText (Canvas,10,10,"William T. Verts")
    WriteBMP("Eyeballs.bmp", Canvas)
    return
```

In Python, at the `>>>` prompt, type `Main()` with the parentheses and press Enter↵. The program should run as it is shown here, but should not do anything except save a light-gray but otherwise blank canvas to `Eyeballs.bmp` with dimensions selected by the user. Fix any syntax errors or other mistakes. Do not change `Main` or `requestColor`, but we will have you fill in code for the other four functions.

On the next few pages are the detailed descriptions of how to write the code in each of the the four functions, `addStarburst`, `addEyball`, `Stare`, and `PlotText`. I recommend that you leave `addStarburst` to last, and concentrate on getting `addEyeball` and `Stare` working first.

Function `addStarburst`

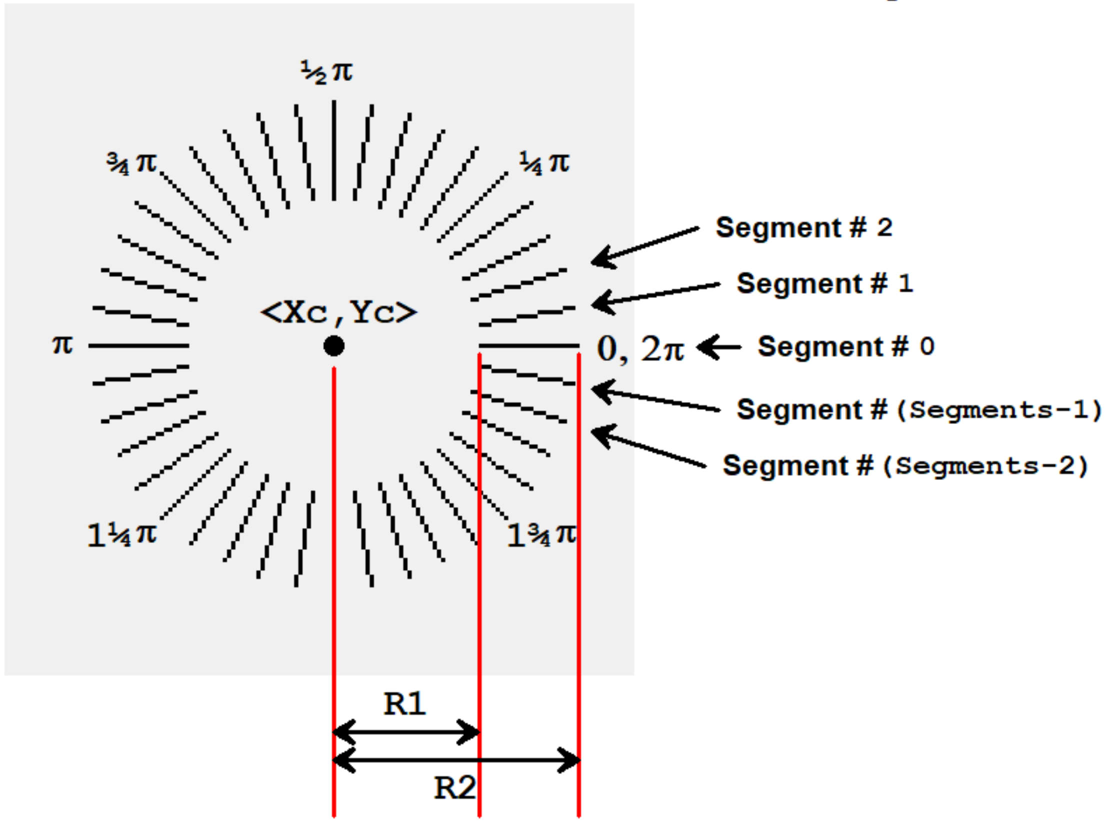
In the blank space in the `addStarburst` function, ***write new code*** to plot a bunch of radial lines around the center `<Xc,Yc>`. The number of lines is in parameter `Segments`, and is always a positive integer (that is, you must use `Segments` and not a constant for the number of lines to draw).

For each line, you need to compute its angle as a fraction of the distance around a circle, which is 2π radians. That is, the increment (how much the angle changes between any two lines) is computed by dividing 2π by the number of segments. Start with the angle equal to 0, and for each new line add the increment to the old angle to compute the new angle. Once you know the angle for a particular line, its endpoints can be computed with a little trigonometry:

```
X1 = Xc + R1 × cos(angle)
Y1 = Yc − R1 × sin(angle)
X2 = Xc + R2 × cos(angle)
Y2 = Yc − R2 × sin(angle)
```

Use the `addLine` function to draw a line between `<X1,Y1>` and `<X2,Y2>`. Be careful about the data types of your numbers: the coordinates are floats, but `addLine` expects integers.

```
def addStarburst (Canvas,Xc,Yc,R1,R2,Segments):
```



Function `Eyeball`

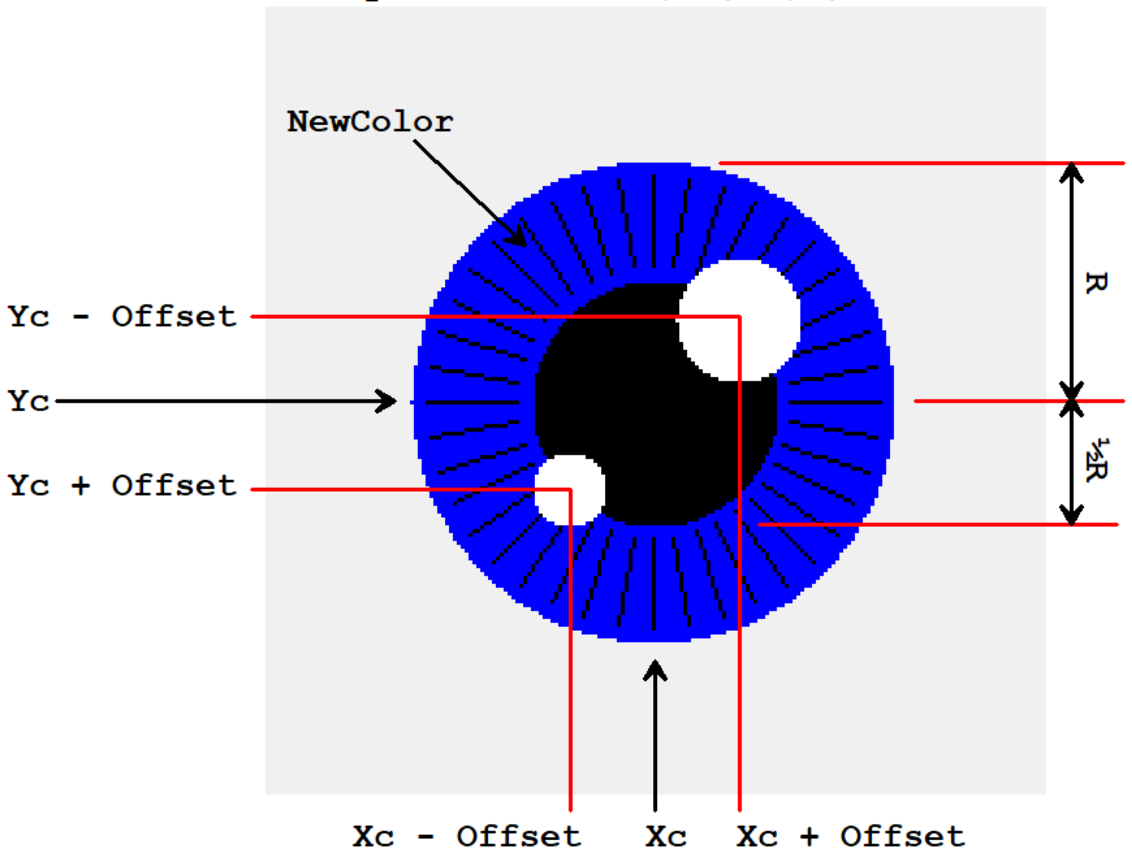
In the `Eyeball` function, *write new code* to paint a single anime eyeball on the canvas, given the center of the circle at location <`Xc`,`Yc`>, the iris radius `R` and iris color `NewColor` (all of this is passed in through the parameter list). In `Eyeball`, you will need to call the `BresenhamFilledCircle` function four times, once for the iris, once for the pupil, and once for each of the two highlights.

The radius of the iris (the colored area) comes directly from parameter `R`. The radius of the pupil (the black center of the eye) is half the radius of the iris, and has the same center coordinates as the iris. The radius of the big highlight (the big white spot) is one-quarter that of the iris. The radius of the smaller highlight is one-seventh that of the iris. The centers of the highlights are located relative to the center of the eyeball at a distance offset in the X and Y directions by an amount calculated as follows:

$$\text{Offset} = \tfrac{1}{4} R \sqrt{2}$$

This will put the center of the highlights directly on the boundary between the pupil and the iris, at 45° angles relative to the center of the pupil.

```
def Eyeball (Canvas,Xc,Yc,R,NewColor):
```



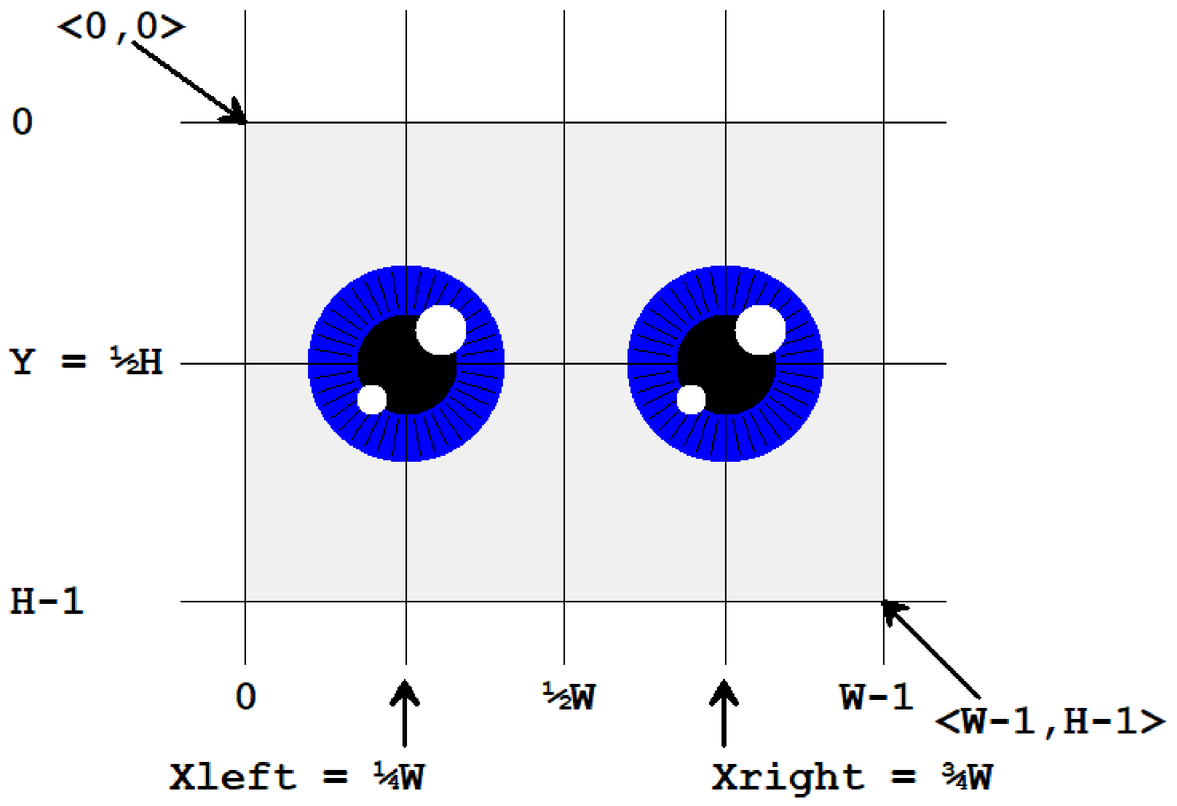
You will also have to call `addStarburst` at the appropriate time. The starburst has the same center coordinates as the pupil and the iris. The inner radius is ½R (the radius of the pupil) plus $^1/_{15}$TH of R, and the outer radius is R minus $^1/_{15}$TH of R. There are 40 segments in the starburst.

Function `Stare`

In the `Stare` function, ***write new code*** to call `Eyeball` twice, once for the left eye and once for the right eye, with the correct radius and coordinates for each eye computed from the size of the `Canvas`. If the width and height of the Canvas are extracted into variables `W` and `H` as follows:

```
W = getWidth(Canvas)
H = getHeight(Canvas)
```

then all the other measurements needed to find the position of the eyeballs are computed as follows (I recommend creating separate variables for each named value below):



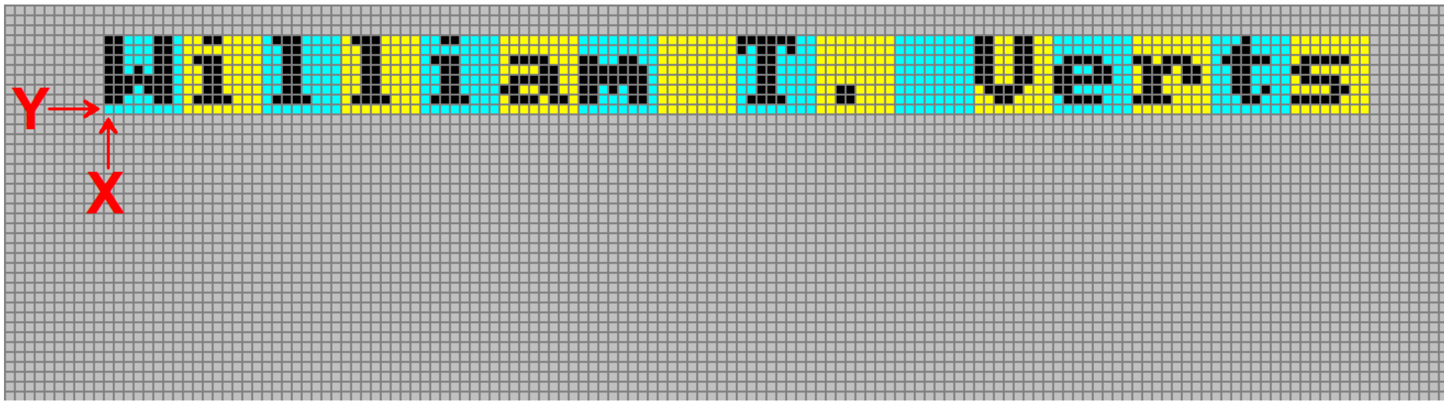
What remains for `Stare` to compute is the eyeball radius.

The radius `R` is computed from the minimum (using Python's `min(…,…)` function) of `Xleft` and `Y`, times ⅔, minus 5 pixels. This may seem overly complicated, but doing it this way guarantees that the eyes will always fit onto the screen.

Function `PlotText`

The purpose of `PlotText` is to put a string of characters onto the canvas at a specified point and with a specified color. Ultimately, the `PlotText` function will need to end up inside `Graphics.py`, but for now we will have you build it in here so it can be graded along with your other code.

For example, the call `PlotText(Canvas,10,10,"William T. Verts",black)` will result in the following placement of text onto the canvas. The X and Y coordinates are to the lower left pixel of the region where the text will be placed, in this case at X=10 and Y=10. The cyan and yellow boxes will NOT appear on screen, but are drawn in here to show where each 8×8 block of pixels is placed.



Inside `PlotText`, your code must compute the placement position of each character, extract the pixel definition of each character from the `Font8x8` list, and plot those pixels on the canvas at the appropriate coordinates.

Remember that the `Font8x8` table (available on the class site) looks like this, as a list containing 256 sub-lists, one for each character, each sub-list containing 8 integers:

```
Font8x8 =                [[  0,   0,   0,   0,   0,   0,   0,   0], \
                          [126, 129, 165, 129, 189, 153, 129, 126], \
                          [126, 255, 219, 255, 195, 231, 255, 126], \
                          [108, 254, 254, 254, 124,  56,  16,   0], \
…
                          [112,  24,  48,  96, 120,   0,   0,   0], \
                          [  0,   0,  60,  60,  60,  60,   0,   0], \
                          [  0,   0,   0,   0,   0,   0,   0,   0]]
```

The index into the list is determined by which character you want to paint. For example, `ord("A")` is 65, so `Font8x8[65]` contains in its sub-list `[48,120,204,204,252,204,204,0]` all the information needed in order to paint an `A`. The first number in a sub-list represents the top row of pixels, the second number the next row of pixels, etc.

Each integer in a sub-list is between 0 and 255 (which fits exactly into a single byte), and the 8-bit binary representation of those integers determines which pixels will be painted (1 bits) and which will not (0 bits). Your function will plot *only* those pixels corresponding to 1-bits; that way any background image will show through.

For example, the capital `A` painted as black pixels on a red background will appear as:

Decimal	Binary (Pixels)
48	00**11**0000
120	0**1111**000
204	**11**00**11**00
204	**11**00**11**00
252	**111111**00
204	**11**00**11**00
204	**11**00**11**00
0	00000000

Finishing Up

That is enough information for you to figure out how to fill out the four functions and run the program. Once complete, try using different sizes for the screen (the eyeball sizes should track this correctly).

When you are done, turn in your assignment through the on-line code submitter Web form.

Grading

Your program will be graded by running it to see if it create a bitmap with the appropriately sized eyeballs. In addition to the standard grading codes (it must contain your name at the top, run without crashing, etc., at:

`https://people.cs.umass.edu/~verts/cmpsci119/Grading%20Codes%20for%20COMPSCI%20119.pdf`

the following additional codes are in effect for this program:

H. -1: Eyeballs are in the wrong positions on the canvas or spill over the edges of the canvas.
I. -1: Eyeball iris size or color is wrong.
J. -1: Eyeball pupil size or color is wrong.
K. -1: Eyeballs highlights sizes or color are wrong, or are in the wrong places.
L. -1: Starburst is missing, the wrong size, or has the wrong number of segments (or uses constant 40).
M. -1: Starburst, highlights, pupil, and iris are in the wrong order.
N. -1: Text (name) is missing, in the wrong place, has scrambled pixels, or is the wrong color.