Harry Netzer's fifth cs project

The purpose of this project was to design an algorithm for the computer to generate a collage of pictures from information stored in a list. A list is a type of variable python can use to store different pieces of data in a single structure. In this project we saw that all of the information for a collage can be stored in a single list.

To make python create a collage out of a list, we first had to decide what information defines a collage. We decided the images themselves, their positions, the filter we wanted to apply to each image, and how much each image should blend if it overlaps another were the fundamental characteristics of a collage. As such our list defining the collage would have to be a list of lists, with each inner list containing slots for each of the variables listed above.

Once we had such a list defining a collage, our first step in building that collage would be to create a canvas of appropriate size. To make a canvas large enough to encompass every image I made a for loop that runs through the list, detecting where the bottom right hand corner of each image in the collage lies. Above the for loop the canvas size is set to 0x0, but each time the for loop runs if overwrites that value if the bottom right corner of the image being looped over is greater than the set canvas size.

Next I created a function called buildCollage, which takes for a parameter a list of lists that defines a collage. The function uses a for loop to run through each inner list in the list, that is each list that defines an image's attributes in the collage, and uses the filter functions and putPixmap to place the image on the canvas. buildCollage is able to read each images attributes by using indexes. When buildCollage is looking for the images's x-position it knows to look at the second element of the list it's in, so I can use the code x0 = item[1] to assign the second item of the list to another variable for easy access.

Next I added an attribute "noblue" to each constituent image's defining list. It takes a boolean, or true/false value, and if it's true then buildCollage knows to call the putPixmapNoBlue function from my filter.py file. Thus I can strip the blue pixels from an image before inserting it into my collage, as I did last week with the picture of me on the moon.

Next I created a collage defining list. This is what it looks like:

```
clist = [ clist = [
  [ 'maine1.ppm', 0, 100, 'swapBGR', 0.8, False, None ],
  [ 'maine2.ppm', 240, 270, 'swapGRB', 0.8, False, None ],
  [ 'maine3.ppm', 300, 0, "orignal", 0.8, False, None ],
  [ 'me.ppm', 0, 0, "original", .8, True, None ]
  ]
```

And this is the collage it produced when I called buildCollage(clist):

Our next task was to make another collage building function that create a collage with dimensions appropriate for a facebook cover photo. To achieve this I made a new function that first makes a canvas of size 750x250, and then has all of the same code as my buildCollage function. The only thing I changed was how putPixmap and putPixmapNoBlue in my filter.py file handle when the image they put is bigger than the canvas. Before I used for loops to take each pixel from the source and place it in the correct position on the destination canvas. For my purposes here I modified those for loops so they would pick up a pixel from the source if it's destination pixel was within the edges of the destination image. That if statement looks like this:

```
if i + y < dest.getHeight() and j + x < dest.getWidth():
```

It takes i and j, the coordinates of the pixel being taken from the source image, adds them to y and x, the displacement when adding the source to the destination image, and tests if the sum is less than the size of the destination image. If it is then the putPixel algorithm continues. If it isn't then the for loop goes on to the next pixel. Thus I was able to prevent python from trying to place every out of bounds pixel onto the destination canvas. The results: