Lab 3 / Part 2: Manipulating photos with loops

Use the Lab 3 part 2 response sheet to write down your answers to the questions in this lab.

Getting Started (same as part 1)

In my code below I will be using an image file, beach.jpg, available at
http://sergey.cs.uni.edu/courses/cs1120/mediasources/beach.jpg

The first time through this lab I would ask you to use that image as well so that our answers are consistent. But I will later ask you to use any second photo of your choice. You can grab a different photo from the mediasources or use any photo of your choice. My only suggestion is that you use one whose dimensions aren't much larger than 500 in either direction. If you have a large hi-res image you have taken with your digital camera or cell phone you should open it in some photo tool and scale it down to an "appropriate size".

Activity B: Understanding how this program works

Launch JES. As a reminder, here is is the code you worked with in Part 1):

```
1 def increaseGreen(picture):
2   for pixel in getPixels(picture):
3     greenValue = getGreen(pixel)
4     setGreen(pixel, greenValue*1.5)
```

At first it might look like this program just "magically" changes all of the pixels to have a higher value for the green channel. But this actually isn't correct. It actually is doing this one pixel at a time. Look at the line of code that says:

```
for pixel in getPixels(picture):
```

Let's start with the last part of that command `getPixels(picture):` This command asks the computer to generate a giant list of every single pixel in the picture. It's big. It's ugly. As the book says, you wouldn't want to try and look at that or print it out. Think about it. The beach picture is 640x480 which means that it contains 307,200 different pixels. It would take a lot of time to print details about every one of those pixels. But you can trust us when we tell you that this command makes a BIG list of every single pixel.

The rest of the command - `for pixel in` - basically asks the computer to consider each of those pixels one at a time, storing the one that you are looking at in a variable named pixel. This means that one at a time you are getting a pixel, storing it in a variable called pixel, getting the green value from that pixel, and changing the green value of that pixel. After the setGreen() command is done the computer comes back to the top of the for loop and sets the variable named pixel to point at the NEXT pixel in the list. In this manner, the computer looks at each pixel one at a time and changes it's green value.

While **I don't recommend** doing this very often, I do like to have you do the following once in a while so that you can understand what is going on.

Type the following modified program into the Program Area of JES, save it as a "lab3b.py" and load it.

```
def increaseGreen(picture):
  counter = 0
  for pixel in getPixels(picture):
    greenValue =  getGreen(pixel)
    setGreen(pixel, greenValue * 1.5)
    if counter % 1000 == 0:
      repaint(picture)
    counter += 1
```

Notice what this change does. Without the counter, it would ask the computer to repaint the picture after EVERY SINGLE change that is made. In other words, we are going to repaint the beach picture 307,200 times. I have added the counter so that we don't have to wait minutes (hours?) for the program to execute – i.e., the picture is repainted every 1000th iteration. You can see why I wouldn't want you to do this very often, but by doing it once in a while you can see what is happening.

Run this code by typing:

```
>>> changed = makePicture("beach.jpg")
>>> show(changed)
>>> increaseGreen(changed)
```

It may take a few seconds before you see anything happening. You should notice that the photo is changing one row at a time. (When you get bored of watching this change you can press the "stop" button which is on the right side of the screen on the bar between the Program Area and the Command Area).

Activity C: Exploring some mystery functions

Begin by simply READING the program shown below:

```
6  def mysteryMethod1(picture):
7    for pixel in getPixels(picture):
8      redValue = getRed(pixel)
9      setRed(pixel,redValue*0.5)
```

[Q9] What do you PREDICT this code will do?

After you make your prediction, enter the program into lab3b.py, save, and load. Fix any typos you may have made. Run the program and see what actually happened by doing something like:

```
>>> changed = makePicture("beach.jpg")
>>> mysteryMethod1(changed)
>>> original = makePicture("beach.jpg")
>>> explore(original)
>>> explore(changed)
```

[Q10] What did the code ACTUALLY do? How did you verify this?

[Q11] mysteryMethod1() is a bad name. It doesn't tell us anything about what this does. What would be a better name?

Now let's repeat this process with some other mystery methods…

Begin by simply READING the program shown below:

```
11 def mysteryMethod2(picture):
12   for pixel in getPixels(picture):
13     setRed(pixel,getRed(pixel)*0.5)
14
```

[Q12] What do you PREDICT this code will do?

[Q13] What did the code ACTUALLY do? How did you verify this?

[Q14] mysteryMethod2() is a bad name. It doesn't tell us anything about what this does. What would be a better name?

Begin by simply READING the program shown below:

```
15 def mysteryMethod3(picture):
16   for pixel in getPixels(picture):
17     setBlue(pixel,0)
```

[Q15] What do you PREDICT this code will do?

After you make your prediction, enter the program into lab3b.py, save, load, and test.

[Q16] What did the code ACTUALLY do? How did you verify this?

[Q17] mysteryMethod3() is a bad name. It doesn't tell us anything about what this does. What would be a better name?

Begin by simply READING the program shown below:

```
19 def mysteryMethod4(picture):
20   for pixel in getPixels(picture):
21     r = getRed(pixel)*1.2
22     g = getGreen(pixel)*1.2
23     b = getBlue(pixel)*1.2
24     newColor = makeColor(r,g,b)
25     setColor(pixel,newColor)
```

[Q18] What do you PREDICT this code will do?

After you make your prediction, enter the program into lab3b.py, save, load, and test.

[Q19] What did the code ACTUALLY do? How did you verify this?

[Q20] mysteryMethod4() is a bad name. It doesn't tell us anything about what this does. What would be a better name?

Begin by simply READING the program shown below:

```
27 def mysteryMethod5(picture):
28   for pixel in getPixels(picture):
29     r = getRed(pixel)+20
30     g = getGreen(pixel)+20
31     b = getBlue(pixel)+20
32     newColor = makeColor(r,g,b)
33     setColor(pixel,newColor)
34
```

[Q21] What do you PREDICT this code will do?

After you make your prediction, enter the program into lab3b.py, save, load, and test.

[Q22] What did the code ACTUALLY do? How did you verify this?

[Q23] mysteryMethod5() is a bad name. It doesn't tell us anything about what this does. What would be a better name?

Begin by simply READING the program shown below:

```
35 def mysteryMethod6(picture):
36   for pixel in getPixels(picture):
37     r = getRed(pixel)
38     g = getGreen(pixel)
39     b = getBlue(pixel)
40     newColor = makeColor(b,r,g)
41     setColor(pixel,newColor)
```

[Q24] What do you PREDICT this code will do?

After you make your prediction, enter the program into lab3b.py, save, load, and test.

[Q25] What did the code ACTUALLY do? How did you verify this?

[Q26] mysteryMethod6() is a bad name. It doesn't tell us anything about what this does. What would be a better name?