CMSC 143: Introduction to Object-Oriented Programming with Robots

Lab 8: Photobooth
Due November 2, 2009

Submit a copy of your python programs as one zip file; all the files should be inside a folder named: `cmsc143_lab8_LASTNAME_FIRSTNAME` with each function in its own, separate file. Each file should have a test that runs your function when the module is run directly (you should include a sample image named `test.jpg` in your folder), but not when it is imported from another file. Your programs should have your name, email, and the date at the top of each file as a comment.

Learning Objectives

- Manipulate Images.
- Understand Image Coordinate Systems
- Use Nested Loops.

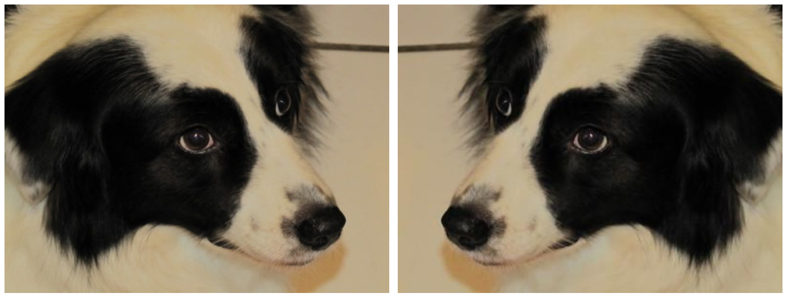
Functions to be Implemented

You must implement the first four functions, the last two, and choose one of the functions 5–8. All your functions return a **new** image, they should not modify the original. Each function should look something like:

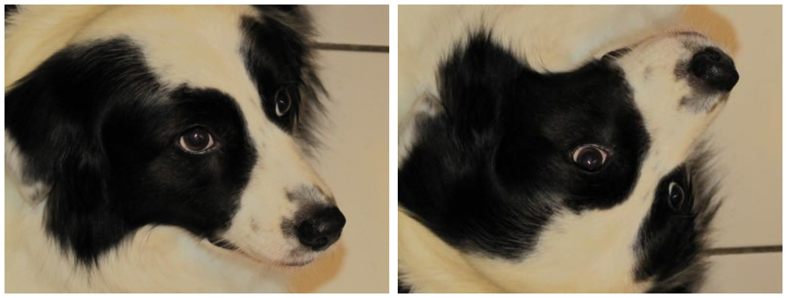
```
def mirror(image):
    INSERT CODE HERE


if __name__ == "__main__":
    p = makePicture('test.jpg')
    show(p)
    np = mirror(image)
    show(np)
    savePicture(np, "mirror.jpg")
```

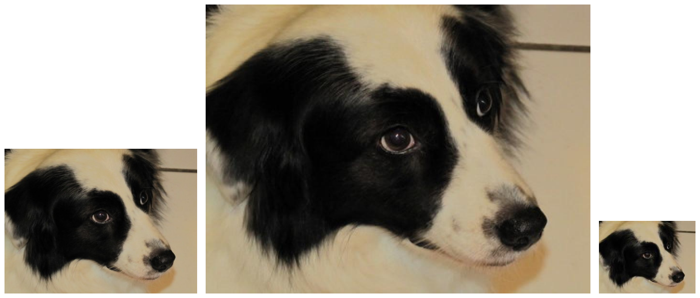
1. `mirror(image)` – Create a mirror image (reversed horizontally)



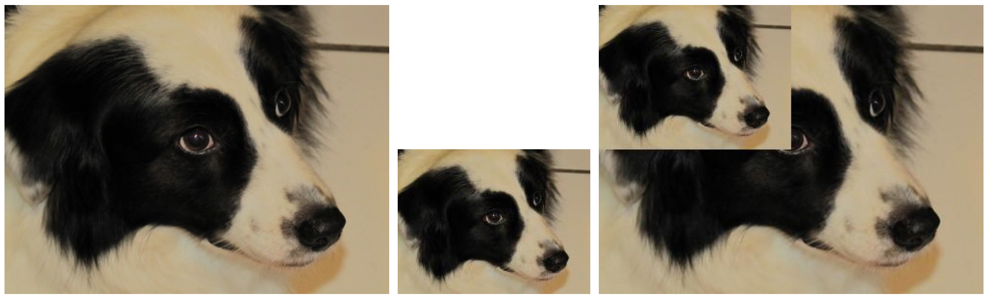
2. `flip(image)` – Create an upside-down image



3. `resize(image, factor)` – Resize the image according to factor; a factor of 2 should make it twice as big, a factor of 0.5 should make it twice as small.

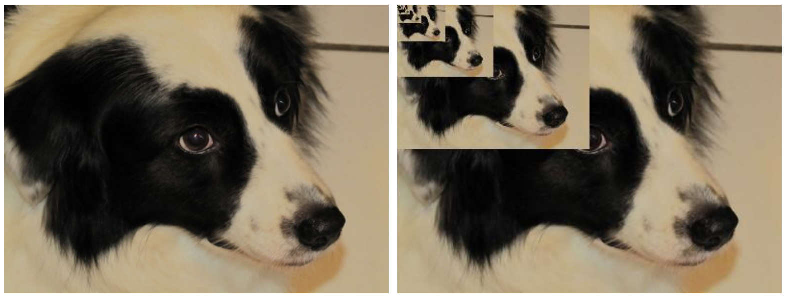


4. `insert(image1, image2, i, j)` – Insert a small image `image2` into a larger image `image1` at the location (i, j) in the larger image.



PICK ONE

5. `nest(image)` – Write a function that uses `resize` and `insert` to nest an image inside of itself, recursively. (Although, your function does not **have** to be recursive.)

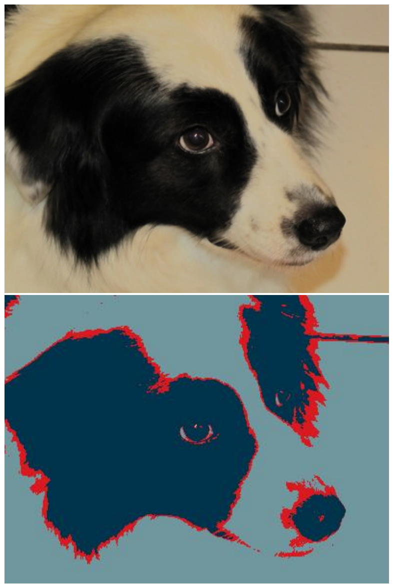


6. `blur(image, neighborhood_size)` – Blur the image by replacing each pixel with the average of itself and its neighboring pixels. If there aren't enough neighboring pixels for some pixel x (pixels on the edges of the image), you should just use x instead of the missing neighbor.

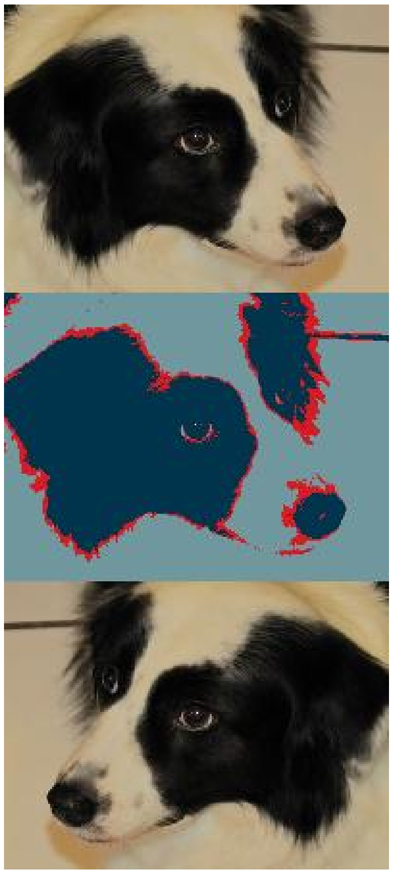
7. `sepia(image)` – Write a function that applies a sepia effect, which raises the red and green channels, and lowers the blue. In particular it adds 2 * `sepiaAmount` to the red, adds `sepiaAmount` to green, and subtracts `sepiaAmount` from blue. 20 is a good value for `sepiaAmount`.

8. `fairey(image)` – Write a function that applies an effect similar to Shepard Fairey's iconic Obama "HOPE" poster. Each pixel is colored one of four colors depending upon its total color saturation value (the sum of RGB values). It assigns roughly equal intervals for each of the four colors. If the saturation is between (0, 181) assign the pixel darkBlue,

Saturation	**Color**
0 – 181	darkBlue (0, 51, 76)
182 – 363	red (217, 26, 33)
364 – 545	lightBlue (112, 150, 158)
545 – 765	yellow (252, 227, 166)



9. `photobooth(listOfImages)` – Write a function that takes a list of images and creates a new image with all the images appended vertically, like the photobooths of years past.



10. Create a program that uses your scribbler's camera to create a photobooth. The scribbler should take 5 pictures, around 2 seconds apart. You might want to beep before taking the image, so your user can smile or make a goofy face. Feel free to apply one of your special effect functions to the images. Then use your photobooth() function to create one big image, and save it to a file.