Computer Science Courses

Project 3: Image Manipulation

Due date: October 17th 23:59PM

This project is an individual project.

1 Purpose

- Learn how to process and manipulate images in python using PIL library
- Learn how to hide information inside image

2 Concepts

- Image object, Color, Pixel structure, Loops, Arrays, bit operators (& and »)

3 PIL Library

- This project will involve extensive use of the PIL Imaging Library. PIL is already installed in the Windows lab and can be downloaded to your home computer from this website:
  - PIL for Python 3 from UC-Irvine [http://www.lfd.uci.edu/~gohlke/pythonlibs/#pil]
  - Use for the version for Python 3.2.

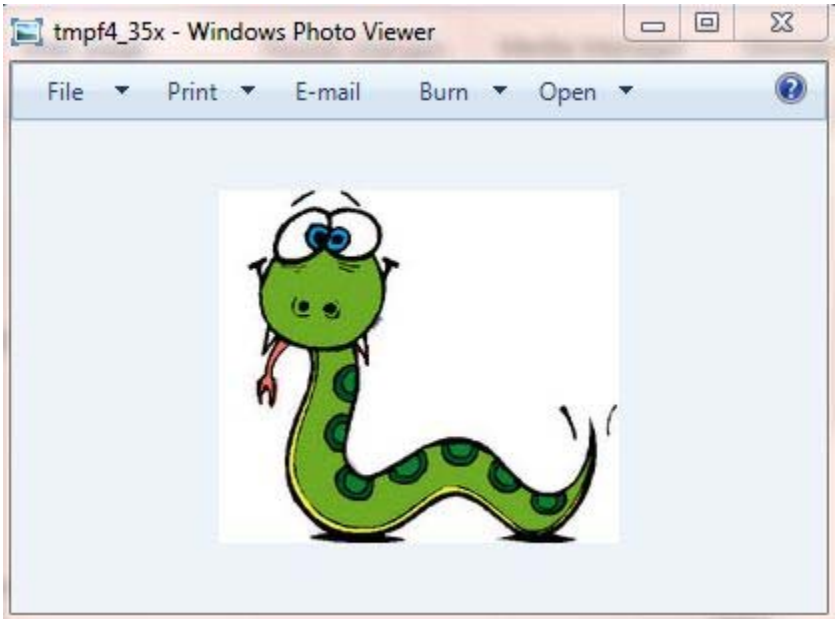
- The handbook below contains helpful information on the various methods you will be using:
  - Python Imaging Library (PIL) Overview [http://www.pythonware.com/media/data/pil-handbook.pdf]

4 Getting Started

In this section, we provide the necessary PIL image manipulation functions you will need in your program. Of course you can always refer to PIL tutorial to learn more functions.

1. Open and Show an image saved in your disk: python.gif

```
from PIL import Image
filename = 'python.gif'
myImage = Image.open(filename)
myImage.show()
```



2. Your image consists of set of pixels, each pixel consists of 3 color values: (Red, Green, Blue). You need to convert your opened image into the RGB format by:

```
myImage = myImage.convert('RGB')
```

3. Pixel Manipulation

Pictures can be viewed as a matrix of pixels. In order to access one pixel, you need to specify it's location. In order to extract the red, green, blue values of the pixel located in x and y location:

```
#The method getpixel, returns 3 values: Red, Green, and Blue respectively

r, g, b = myImage.getpixel((x,y))
```

4. Update the value of a Pixel

In order to update a pixel located in location x, y:

```
myImage.putpixel((x,y),(rValue, gValue, bValue))
```

5. Width and Height of Image

```
width, height = myImage.size
```

6. Saving your modified image under new name

```
newFileName = filename[:-4] #slice the string representing the filename to get rid of the extension .gif
newFileName = newFileName + "todo#xx.gif"
myImage.save(newFileName)
```

7. Bit Level Processing

Each color of red, green, and blue is represented by integer with range 0 to 255. Zero is the dark shade of color and 255 is light shade. You need 8 Bits to represent 255 different color shade combination (2^8 = 256)

```
0: 00000000
1: 00000001
2: 00000010
...
254:11111110
```

```
255:11111111
```

In image processing, it is common to systematically manipulate portions of the bits representing the binary value of color. Example: set the least 2 bits of the red color to zeros, red value = 175 . In order to access the least 2 bits, you can use **Bit wise AND &** the value of the color with dummy pattern: 11111100

```
10101111 #represent 175
11111100 #set the last 2 bits of the dummy pattern to zeros
--------
10101100 #172
```

In Python

```
rValue = 175
newValue = rValue & 0b11111100 #in python you need to prefix the binary value with 0b
```

Another way to manipulate color value is using **Binary Right Shift »** operator.

Given the number 255 with Binary representation: 11111111 and applying binary right shift by 2 will remove the 2 right most bits and inserts 2 zero's in the left most position Result: 00111111

```
11111111 #255
after right shift by 2
00111111 #63
```

In python

```
rValue = 255
newValue = rValue >> 2
print (newValue) #63
```

SET UP

Under your CS177 folder create the folder **project3**.

Download the skeleton code Here

Download the following 5 .gif images under the folder project3. Right click and save these images to your project3 folder.

orangefish.gif

purplefish.gif

bluefish.gif

greenfish.gif

purdue.gif

Todo #1 Open All Images

Write a function called **openAll** that has as input arguments the file names of three image files to be opened, then:

1. Creates the image for each file.

2. Displays each image.

Test case

Input: argument#1 orangefish.gif, argument#2 purplefish.gif, argument#3 purdue.gif

Todo #2 Black and White

Write a function called **blackAndWhite** that has as input argument the file name of an image file (example: bluefish.gif), then transforms the image in black and white and displays it. In order to create a black and white image, do:

1. Prepare the image file: (i.e open the image file and convert it into RGB representation)

2. Display the initial image

3. Loop over each pixel in the image (hint: you'll need a nested for loop) and compute the average of r, g, b. Next, update the pixel value with average.

4. Save the new image in a file called: filename"_BW_mod.gif"

5. Display the black and white image on screen

Test case:

Input: greenfish.gif

Output: greenfish_bw_mod.gif

Todo #3 No Green

Write a function called **noGreen** that has as input argument the file name of an image file and then removes the green component of each pixel in the image. Save the new image in a file called: filename+"_nogreen_mod.gif". Your function will display the original image and then the modified image with no green.

Test case

Input: purplefish.gif

Output: purplefish_nogreen_mod.gif

Todo #4 Zero Low Bits

Write a function called **zeroLowBits** that has as input argument the file name of an image file, then sets the lower 4 bits of each pixel (red, green, and blue components) of the image to zero. Save the new image in a file called filename+"_zerolowbits_mod.gif" Your function will display the original image first, and then modified image with low bit set to zero. Notice that the new image looks very similar to the original.

Test case

Input: bluefish.gif

Output bluefish_zerolowbits_mod-correct.gif

Todo #5 shift High Bits

Write a function called **shiftHighBits** that has two input arguments. The first one is the file name of an image file; the second one is the maxshift value. Your function will first display the original image and then do the following:

Repeat for each k in range 1 to maxshift:

1. Iterate through every pixel in the image and right shift the color bits by k bits.

2. Save the transformed image in a file having as name: filename+"shiftbits"+str(k)+"mod.gif"

Input: argument#1: orangefish.gif , argument#2: 5

Output: orangefish_shiftbits_1mod.gif, orangefish_shiftbits_2mod.gif, orangefish_shiftbits_3mod.gif, orangefish_shiftbits_4mod.gif

Todo #6 Hide Picture

Steganography is the art and science of writing hidden messages in a way that no one besides the sender and recipient suspects the existence of the secret message. A common technique used in steganography consists in hiding a message inside a picture.

One convenient way to hide message in a picture is to use pixel colors. Each color, encoded by a number in range 0-255, uses 8 bits (2^8 - 1 = 255). Note that the high 4 bits contributes more towards the value of the pixel.

```
Example: 11010110:
The high 4 bits 1101: 208
The low 4 bits 0110: 6
```

In this task you will write a function called **hidePicture** that has two arguments representing the file names of two image files, image1 and image2. Your function should hide image2 inside image1 by replacing the four low bits of each color component (r, g, b) of each pixel of image1 with the four high bits of image2 corresponding pixel:

1. Extract the 4 high bits of each color component of each pixel of image2

2. Shift right the result in step#1 by 4 so it becomes 4 low bits

3. Set the 4 low bits in image1 to zero's

4. Assemble the new color pixel: the higher 4 bits of image1 and the shifted higher bits of image2 you calculated in step#2

5. Save the new assembled image as filename1+"_CombinePic_mod.gif"

Now you need to verify that your image hiding has worked. Extract the hidden message from the image you saved in step #4 and display the extracted image as follows:

1. Extract the 4 low bits

2. Shift left the 4 low bits to make them high bits

3. Update the color pixel.

4. Save the extracted image as hiddenPic.gif

Note: When hiding one image into another, make sure to include an if statement to check that the dimensions of the two images are equal.

Test case INPUT argument#1: purdue.gif, bluefish.gif

purdue_combinepic_mod.gif , hiddenpic.gif

Todo #7 Morph Picture

PIL Library contains a function called blend:

```
Image.blend(image1, image2, alpha)
Returns: Image
```

that creates a new image by *morphing* image1 and image2, using a constant alpha.

```
  outImage = image1 * (1.0 - alpha) + image2 * alpha
```

If alpha is 0.0, a copy of the first image is returned. If alpha is 1.0, a copy of the second image is returned. There are no restrictions on the alpha value. If necessary, the result is clipped to fit into the allowed output range.

Note: Both images must have the same size.

In this task, you will create a function called **morphPicture** that has two arguments representing the file names of image1 and image2. Your function will use a for loop to call Image.blend with the following alpha values: 0, 1/10, 2/10, … 1. Your function then saves and displays the result image for each alpha.

The name of the files created by your function must have the following format: "morph"+str(k)+".gif". Note that (**k**) means the value of the variable that controls the for loop.

Test case

INPUT: argument#1: greenFish.gif, argument#2: purpleFish.gif

The expected out depends whether or not you converted image1 and image2 to RGB format before morphing the image

Without RGB conversion:

OUTPUT: morph0.gif , morph1.gif , morph2.gif , morph3.gif , morph4.gif , morph5.gif , morph6.gif , morph7.gif , morph8.gif , morph9.gif , morph10.gif

With RGB conversion

OUTPUT: m0.gif , m1.gif , m2.gif , m3.gif , m4.gif , m5.gif , m6.gif , m7.gif , m8.gif , m9.gif , m10.gif

Todo #8 Main

Write a main function with menu interface to call the previous functions with the following options:

1) Prompt the user to input the names of three files to be opened, then call function OpenAll

2) Prompt the user to input the file name of the image, then call blackAndWhite

3) Prompt the user to input the file name of the image, then call noGreen

4) Prompt the user to input the file name of the image, then call zeroLowBits

5) Prompt the user to input the file name of the image, then call shiftHighBits

6) Prompt the user to input the names of files of image1 and image2 then call hidePicture

7) Prompt the user to input the names of files of image1 and image2 then call morphPicture

8) Exit

The menu should look like as the following one:

```
              Main Menu
_____________________________________
1)Open All
2)Black and White
3)No Green
4)Zero Low Bits
5)Shift High Bits
6)Hide Picture
7)Morph Picture
8)Exit
Please choose option:
```

GRADING RUBRIC

TODO	Points
TODO1	**10**
TODO2	**10**
TODO3	**10**
TODO4	**10**
TODO5	**15**
TODO6	
- Hiding Picture	**10**
- Extract the hidden Picture	**10**
TODO7	**15**
TODO8	**10**
TOTAL	**100**

TURNIN INSTRUCTIONS

Login into your UNIX CS account. Then go to your project3 folder using the following commands:

```
$cd CS177
$cd project3
```

Then launch the following command AS IS:

```
$turnin -v -c cs177=COMMON -p project3 *.py
```