

[Extra] Gold Problem 3: Looks Good!

Copied from:

<https://www.cs.hmc.edu/twiki/bin/view/CS5/FunWithImagesGold2010> on 3/22/2017

up to +10 points: individual or pair

This is an optional, extra credit problem this week -- it introduces some image-processing ideas using Python's list comprehensions (and functions!)

(filename: `hw4pr3.zip`)

In this problem you will have the opportunity to write programs that algorithmically alter images... .

Our PNG library...

This problem uses a Python library that reads and writes `png` images. Fortunately, both Mac OS and Windows have built-in programs (Preview; Paint) that convert almost any image to the *portable network graphics*, or `png`, format.

Grab the starter files from here:

[hw4pr3.zip](#)

Trying them out...


Open up the `hw4pr3.py` file in your usual way (no special instructions needed...). Run the file with `ipython -i hw4pr3.py` and you should then be able to run `invert()`, which is provided at the bottom of the file and is copied here for reference:

```
def invert():
    """ run this function to read in the in.png image,
        change it, and write out the result to out.png
    """
    Im_pix = getRGB( 'spam.png' ) # read in the in.png image
    print( "The first two pixels of the first row are", )
    print( Im_pix[0][0:2] )
    # remember that Im_pix is a list (the image)
```

```
# of lists (each row) of lists (each pixel is [R,G,B])
New_pix = [ [ change(p) for p in row ] for row in Im_pix ]
# now, save to the file 'out.png'
saveRGB( New_pix, 'out.png' )
```

Read over this function to get a sense of how to read in, change, and output image files. *Think through the key datastructure, `Im_pix`, and its parallel, `New_pixM`.* The next section should help... .

The file used in the above code is the `spam.png` image, used throughout this page. There are two others in the same folder:

Here is the provided `in.png` image: 

Here is the changed (inverted) `out.png` file: 

Similarly, here are the original and inverted Olin images:



How are the images represented?

The key data structure in the above code is `Im_pix`, which contains all of the image's pixel data.

But how does it contain that data? This is the key to changing it successfully!

`Im_pix` is a list of *pixel rows*.

- Each *pixel row* is, in turn, a list of *pixels*
 - Each *pixel* is, in turn, a list of three integers: the pixel's red intensity, the pixel's green intensity, and the pixel's blue intensity.

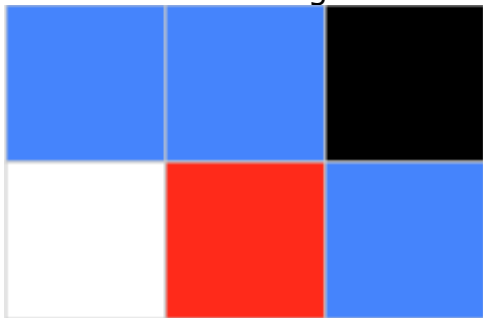
For example, here is a complete 2x3 image (with two rows of three pixels in each row):

```
Im_pix = [ [ [0,0,255], [0,0,255], [0,0,0] ], #  
first row of three pixels  
          [ [255,255,255], [255,0,0], [0,0,255] ] ]  
# second row of three pixels
```

Here, in the first row, the top-left and top-middle pixels are pure blue, and the top-right pixel is black.

The bottom-left pixel is white, the bottom-middle pixel is red, and the bottom-right pixel is blue.

With 1970's video-game-sized pixels, this image looks like this:



The list-comprehension in the example code has two levels:

- The outer level iterates through each row of the image (each one is named `row`, the runner variable for that portion)
 - The code at this outer level is `for row in Im_pix`
- The inner level iterates through each pixel of the image (each one is named `p`, the runner variable for this portion)
 - The code at this inner level is `for p in row`
- Remember that `p`, itself, is a list of three integers: `[red,green,blue]`

With this model in mind, you're ready to "rearrange" the data however you'd like! That's next...

Functions for you to write

First, we will focus on pixel-level operations. For making greyscale and binary images, you will have to focus on the [image luminance](#). In essence, it's how bright or dark the colors in a pixel are (compared to white).

As Wikipedia calculates it, luminance is 21% red, 72% green, and 7% blue. Intuitively, this makes sense because if you think of standard **red**, **green**, and **blue**, green is the lightest and thus has highest positive impact luminance, while blue is darker and has a lower value for luminance. This is useful! You're going to calculate luminance for pixel operations.

Playing with Pixels

- We have provided a function called `invert`, which modifies an image to create its negative. That is, all color values are 255 minus their original value. Especially note the use of list comprehension in `invert`, which iterates over every pixel in the image and calls `change(rgb values)`. It's easiest to also write `greyscale` and `binarize` in the same format: a main function that accepts user input and contains a list comprehension that calls a helper function on each pixel.
- Next, write a function called `greyscale` that modifies an image to make it grayscale. For this, you'll want to do something similar to `invert`, except that new change function will calculate the luminance of the pixel using the operations described above. Since luminance is an indication of how white/black a pixel is, having your helper function return a list of RGB values in grayscale is easy -- just return the same value in each of the three color channels!



An image before and after luminance (greyscaling) conversion.

Getting an **OverflowError: unsigned byte integer is greater than maximum**? This might be because your luminance calculation results in RGB values higher than 255. Make sure that all of your percentages add up to 1.

Getting a **filename is not defined** error? Make sure you have single quotes around the image name when you call it in Python shell, the image is in the

same folder as your python script, and permissions on your image are correct.

- Write a function called `binarize(thresh)`, which binarizes an image (makes it black and white) with a threshold of `thresh` given by the user. This threshold is a brightness value between 0 and 255 - if a pixel is greater than the threshold value, then it should turn white, and if its less than the threshold value, then it should turn black. So, a threshold value of 0 means that your photo will turn white and a threshold of 255 means your photo will turn black.



Binary spam with a threshold value of 100.

Geometric Transformations!

- Write `flipVert`, which flips the image on its horizontal axis (the bottom is on the top and the top is on the bottom). You will use the same basic structure as the earlier problems- one main function that opens the file and calls the helper function in a list comprehension. In `flipVert`, you'll want to iterate over only the **rows** instead of the **pixels** in `Im_pix` and reverse their order. Remember that if `L` is a list, then `L[::-1]` is the reverse of that list.



in.png flipped vertically.

- `flipHoriz`: Flip the image on its vertical axis. This should work in the same way as `flipVert` but flip in the horizontal direction. Instead of reordering the rows, you want to consider how the pixels in the rows reorder when an image is flipped horizontally. Note: flipping `in.png` horizontally has no effect because it is symmetric about the vertical axis...



MAPS!

- `mirrorVert`: Mirror the photo across its horizontal axis (i.e., so that the top part is mirrored upside down on the bottom of the image). The easiest approach is to replace the bottom half of `Im_pix` with the reversed rows from the top half. To do this, you'll use the built-in function `getWH()` in `cs5png.py` to get the height of `Im_pix`.

Warning: You cannot create a copy of a list with `list1 = list2`, because instead of creating a new object, Python will just create a new reference to the old list. However, list slicing does create new copies of data, so consider how you could combine two slices, each of half the size of the original list, in order to mirror the image... .



in.png mirrored vertically

- `mirrorHoriz`: Same as above, but across the vertical axis. Instead of replacing the bottom rows with the reversed top rows (as you did in `mirrorVert`), you'll replace the last half of the pixels in every row with the reversed first half of the pixels.



The bigger question: which side do you open?

- `scale`: Scale the image to half of each of its original dimensions (this will be a quarter of its original area). The easiest way to do this is to eliminate every other pixel in each row (scaling the image horizontally) **and** eliminate every other row (scaling the image vertically).

Even more?

In addition, if you'd like to create your own effects, we'd love to see them!! Please be sure to include a comment or note to the graders explaining what you did. Be creative!

Also, feel free to include one or two images of your own choosing that you've transformed algorithmically... .

Submitting...

Submission warning!: Please zip up screenshots of your results along with your `hw4pr3.py` file into a folder named `hw4pr3`.

Then, zip that folder into a zip archive named `hw4pr3.zip`, and then submit it in the usual way.

As they say in computer vision labs, *So many pixels, so little time!*