Intro to Computer Vision **Instructor:** *Meera Hahn*
Problem Set 1
Due: Sep 8th 11:59pm

1 Instructions

1. The assignment must be done in Python3. No other programming languages are allowed.

2. Fill your answers in the answer sheet PPT provided and submit the file under the name: FirstName_LastName_PS1.pdf on Canvas. Please do not modify the layout of the boxes provided in the answer sheet and fit your answers within the space provided.

3. Please enter your code in the designated areas of the python files - `PS1Q1.py, PS1Q2.py, PS1Q3.py`. Please do not add additional functions/imports to the files. Points will be deducted for any changes to code/file names, use of static paths and anything else that needs manual intervention to fix.

4. Please submit `PS1Q1.py, PS1Q2.py, PS1Q3.py and FirstName_LastName_PS1.pdf` in a zipped format with your Emory username as a command line argument (using –emory_username), to Canvas. Please do not create subdirectories within the main directory.

5. For the implementation questions, make sure your code is bug-free and works out of the box. Please be sure to submit all main and helper functions. Be sure to not include absolute paths. Points will be deducted if your code does not run out of the box.

6. If plots are required, you must include them in your report and your code must display them when run. Points will be deducted for not following this protocol.

7. The only libraries allowed for this assignment are numpy, matplotlib, pillow, opencv and skimage. Use pip install -r requirements.txt in a virtual environment to install necessary packages.

2 Background/Warmup

Read through the provided Python, NumPy and Matplotlib introduction code and comments:
http://cs231n.github.io/python-numpy-tutorial/ or
https://filebox.ece.vt.edu/ F15ECE5554ECE4984/resources/numpy.pdf.

Open an interactive session in Python and test the commands by typing them at the prompt. (Skip this step if you are already familiar with Python and NumPy.)
After reading the required documentation in the above section, to test your knowledge ensure that you know the outputs of the following commands without actually running them.

```
import numpy as np
x = np.random.permutation(1000)
a = np.array([[11,22,33],[40,50,60],[77,88,99]])
b = a[2,:]
a = np.array([[11,22,33],[40,50,60],[77,88,99])
b = a.reshape(-1)
f = np.random.randn(5,1)
g = f[f>0]
x = np.zeros(10)+0.5
y = 0.5*np.ones(len(x))
z = x + y
a = np.arange(1,100)
b = a[::-1]
```

3 Short answer problems

1. For each of the following there is a corresponding function in `Problem1.ipynb` and in `PS1Q1.py`. Fill out each function with few lines of code to do each of the following. You are turning in code that is in `PS1Q1.py` and the ipynb is just an easy format to implement and test your solutions.

   (a) Use numpy.random.rand to return the roll of a six-sided die over N trials.

   (b) Let y be the vector: `y = np.array([11, 22, 33, 44, 55, 66])`. Use the reshape command to form a new matrix z that looks like this: `[[11,22],[33,44],[55,66]]`

   (c) Use the `numpy.max` and `numpy.where` functions to set `x` to the maximum value that occurs in `z` (above), and set `r` to the row number (0-indexed) it occurs in and `c` to the column number (0-indexed) it occurs in.

   (d) Let v be the vector: `v = np.array([1, 4, 7, 1, 2, 6, 8, 1, 9])`. Set a new variable `x` to be the number of 1's in the vector `v`

2. Load the 100x100 matrix `inputAPS1Q2.npy` which is the matrix `A`. Fill the template functions to load `inputAPS1Q2.npy` and perform each of the following actions on `A`. Fill out each function with few lines of code to do each of the following. You are turning in code that is in `PS1Q2.py` and the ipynb is just an easy format to implement and test your solutions.

   (a) Plot all the intensities in `A`, sorted in decreasing value. Provide the plot in your answer sheet.(Note, in this case we don't care about the 2D structure of `A`, we only want to sort the list of all intensities.) To ensure consistency, you may use the gray colormap option.

   (b) Display a histogram of `A`'s intensities with 20 bins. Again, we do not care about the 2D structure.

(c) Create and return a new matrix `X` that consists of the bottom left quadrant of `A`. (Use function `prob_2_3` and return `X`.) Provide the histogram in your answer sheet.

(d) Create and return a new matrix `Y`, which is the same as `A`, but with `A`'s mean intensity value subtracted from each pixel. (Use function `prob_2_4` and return `Y`.)

(e) Create and return a new matrix `Z` that represents a color image the same size as `A`, but with 3 channels to represent R, G and B values. Set the values to be red (i.e., R = 1, G = 0, B = 0) wherever the intensity in `A` is greater than a threshold `t =` the average intensity in `A`, and black everywhere else. Save `Z` as `outputZPS1Q2.png` and be sure to view it in an image viewer to make sure it looks right. (Use function `prob_2_5` and return `Z`.)

4 Programming Problems

1. The input color image `inputPS1Q3.jpg` has been provided. Fill the template functions to perform the following transformations. Avoid using loops. Display all the resultant images in your answer sheet. Note: In every part, the image that is returned from your function must have integer values in the range [0, 255] i.e uint8 format. For each of the following there is a corresponding function in `Problem3.ipynb` and in `PS1Q3.py`. Fill out each function with few lines of code to do each of the following. You are turning in code that is in `PS1Q3.py` and the ipynb is just an easy format to implement and test your solutions.

   (a) Load the input color image and swap its red and green color channels. Return the output image. (Use function `prob_3_1` and return `swapImg`.)

   (b) Convert the input color image to a grayscale image. Return the grayscale image. (Use function `prob_3_2` and return `grayImg`.) Perform each of the below transformations on the grayscale image produced in part 2 above.

   (c) Convert the grayscale image to its negative image, in which the lightest values appear dark and vice versa. Return the negative image. (Use function `prob_3_3` and return `negativeImg`.)

   (d) Map the grayscale image to its mirror image (flipping it left to right). Return the mirror image. (Use function `prob_3_4` and return `mirrorImg`.)

   (e) Average the grayscale image with its mirror image (use typecasting). Return the averaged image. (Use function `prob_3_5` and return `avgImg`.)

   (f) Create a matrix N whose size is same as the grayscale image, containing random numbers in the range [0, 255]. Save this matrix in a file called `noise.npy`. Add N to the grayscale image, then clip the resulting image to have a maximum value of 255. Return the clipped image. (Use function `prob_3_6` and return `addedNoiseImg`.) Be sure to submit `noise.npy`.

Tips: Do the necessary typecasting (uint8 and double) when working with or displaying the images. If you can't find some functions in numpy (such as rgb2gray), you can write your own function. For example:

```
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])
```