Homework 0

In this homework, we will go through basic linear algebra and image manipulation using python to get everyone on the same page for the prerequisite skills for this class.

One of the aims of this homework assignment is to get you to start getting comfortable searching for useful library functions online. So in many of the functions you will implement, you will have to look up helper functions.

In [2]:
```
#Imports the print function from newer versions of python
from __future__ import print_function

#Setup

# The Random module for implements pseudo-random number generators
import random

# Numpy is the main package for scientific computing with Python.
# This will be one of our most used libraries in this class
import numpy as np


#Imports all the methods in each of the files: linalg.py and imageManip.
py
from linalg import *
from imageManip import *

#Matplotlib is a useful plotting library for python
import matplotlib.pyplot as plt
# This code is to make matplotlib figures appear inline in the
# notebook rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'


# Some more magic so that the notebook will reload external python modul
es;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-i
n-ipython
%load_ext autoreload
%autoreload 2
%reload_ext autoreload
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Question 1: Linear Algebra Review

In this section, we will review linear algebra and learn how to use vectors and matrices in python using numpy. By the end of this section, you will have implemented all the required methods in `linalg.py`.

Question 1.1 (5 points)

First, let's test whether you can define the following matrices and vectors using numpy. Look up `np.array()` for help. In the next code block, define M as a $(4, 3)$ matrix, a as a $(1, 3)$ row vector and b as a $(3, 1)$ column vector:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$
$$a = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$
$$b = \begin{bmatrix} -1 \\ 2 \\ 5 \end{bmatrix}$$

In [3]:
```
### YOUR CODE HERE
M = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
a = np.array([[1,1,0]])
b = np.array([[-1],[2],[5]])
### END CODE HERE
print("M = \n", M)
print("The size of M is: ", M.shape)
print()
print("a = ", a)
print("The size of a is: ", a.shape)
print()
print("b = ", b)
print("The size of b is: ", b.shape)
```

```
M =
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
The size of M is:  (4, 3)

a =  [[1 1 0]]
The size of a is:  (1, 3)

b =  [[-1]
 [ 2]
 [ 5]]
The size of b is:  (3, 1)
```

Question 1.2 (5 points)

Implement the `dot_product()` method in `linalg.py` and check that it returns the correct answer for $a^T b$.

In [34]:
```
# Now, let's test out this dot product. Your answer should be [[1]].
aDotB = dot_product(a, b)
print(aDotB)

print("The size is: ", aDotB.shape)
```

```
[[1]]
The size is:  (1, 1)
```

Question 1.3 (5 points)

Implement the `complicated_matrix_function()` method in `linalg.py` and use it to compute $(a^T b)Ma^T$

IMPORTANT NOTE: The `complicated_matrix_function()` method expects all inputs to be two dimensional numpy arrays, as opposed to 1-D arrays. This is an important distinction, because 2-D arrays can be transposed, while 1-D arrays cannot.

To transpose a 2-D array, you can use the syntax `array.T`

In [41]:
```
# Your answer should be $[[3], [9], [15], [21]]$ of shape(4, 1).
ans = complicated_matrix_function(M, a, b)
print(ans)
print()
print("The size is: ", ans.shape)
```

```
[[ 3]
 [ 9]
 [15]
 [21]]

The size is:  (4, 1)
```

In [44]:
```
M_2 = np.array(range(4)).reshape((2,2))
a_2 = np.array([[1,1]])
b_2 = np.array([[10, 10]]).T
print(M_2.shape)
print(a_2.shape)
print(b_2.shape)
print()

# Your answer should be $[[ 20], [100]]$ of shape(2, 1).
ans = complicated_matrix_function(M_2, a_2, b_2)
print(ans)
print()
print("The size is: ", ans.shape)
```

```
(2, 2)
(1, 2)
(2, 1)

[[ 20]
 [100]]

The size is:  (2, 1)
```

Question 1.4 (10 points)

Implement `svd()` and `get_singular_values()` methods. In this method, perform singular value decomposition on the input matrix and return the largest k singular values (k is specified in the method calls below).

In [46]:
```
# Let's first only get the first singular value and print it out. It sho
uld be ~ 25.46.
only_first_singular_value = get_singular_values(M, 1)
print(only_first_singular_value)

# Now, let's get the first two singular values.
# Notice the first singular value is a lot larger than the second one.
first_two_singular_values = get_singular_values(M, 2)
print(first_two_singular_values)

# Let's make sure that the first singular value in both is the same.
assert only_first_singular_value[0] == first_two_singular_values[0]
```

```
[25.46240744]
[25.46240744  1.29066168]
```

Question 1.5 (10 points)

Implement `eigen_decomp()` and `get_eigen_values_and_vectors()` methods. In this method, perform eigenvalue decomposition on the following matrix and return the largest k eigen values and corresponding eigen vectors (k is specified in the method calls below).

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

In [82]:

```
# Let's define M.
M = np.array([[1,2,3],[4,5,6],[7,8,9]])

# Now let's grab the first eigenvalue and first eigenvector.
# You should get back a single eigenvalue and a single eigenvector.
val, vec = get_eigen_values_and_vectors(M[:,:3], 1)
print("First eigenvalue =", val[0])
print()
print("First eigenvector =", vec[0])
print()
assert len(vec) == 1

# Now, let's get the first two eigenvalues and eigenvectors.
# You should get back a list of two eigenvalues and a list of two eigenv
ector arrays.
val, vec = get_eigen_values_and_vectors(M[:,:3], 2)
print("Eigenvalues =", val)
print()
print("Eigenvectors =", vec)
assert len(vec) == 2
```

```
First eigenvalue = 16.116843969807043

First eigenvector = [-0.23197069 -0.52532209 -0.8186735 ]

Eigenvalues = [16.116843969807043, -1.1168439698070427]

Eigenvectors = [array([-0.23197069, -0.52532209, -0.8186735 ]), array
([-0.78583024, -0.08675134,  0.61232756])]
```

Part 2: Image Manipulation

Now that you are familiar with using matrices and vectors. Let's load some images and treat them as matrices and do some operations on them. By the end of this section, you will have implemented all the methods in `imageManip.py`

In [92]:
```
# Run this code to set the locations of the images we will be using.
# You can change these paths to point to your own images if you want to
 try them out for fun.

image1_path = './image1.jpg'
image2_path = './image2.jpg'

def display(img):
    # Show image
    plt.figure(figsize = (5,5))
    plt.imshow(img)
    plt.axis('off')
    plt.show()
```

Question 2.1 (5 points)

Implement the load method in imageManip.py and read the display method below. We will use these two methods through the rest of the notebook to visualize our work.

In [88]:
```
image1 = load(image1_path)
image2 = load(image2_path)

display(image1)
display(image2)
```

Question 2.2 (10 points)

Implement the `dim_image()` method by converting images according to $x_n = 0.5 * x_p^2$ for every pixel, where x_n is the new value and x_p is the original value.

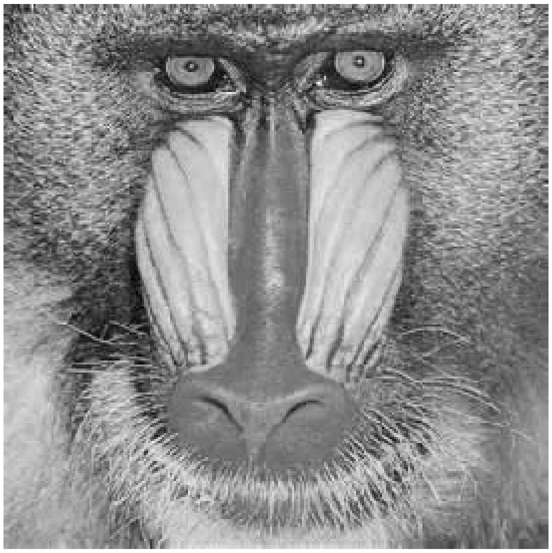
Note: Since all the pixel values of the image are in the range $[0, 1]$, the above formula will result in reducing these pixels values and therefore make the image dimmer.

In [89]:
```
new_image = dim_image(image1)
display(new_image)
```

Question 2.3 (10 points)

Implement the convert_to_grey_scale method and convert the image into grey scale.

In [90]:
```
grey_image = convert_to_grey_scale(image1)
display(grey_image)
```



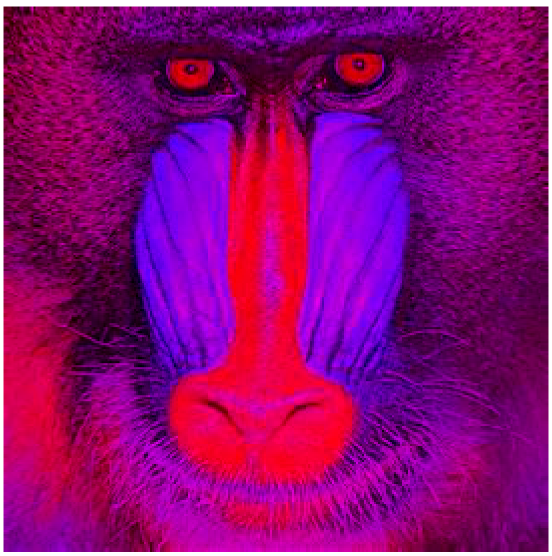
Question 2.4 (10 points)

Implement the `rgb_exclusion()`, in which the input image is decomposed into the three channels: R, G and B and return the image excluding the specified channel.

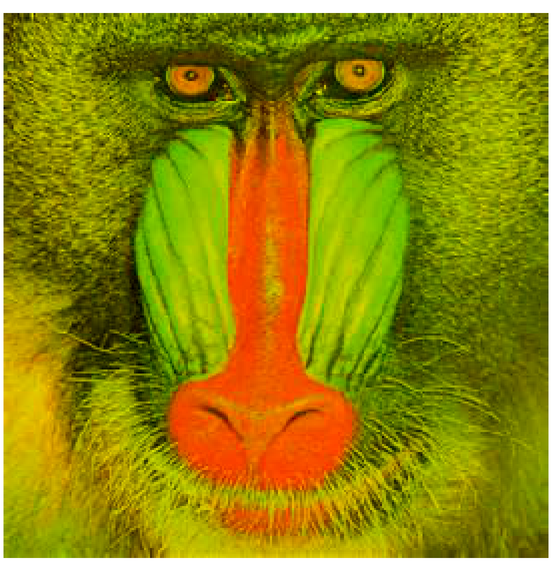
In [91]:
```
without_red = rgb_exclusion(image1, 'R')
without_blue = rgb_exclusion(image1, 'B')
without_green = rgb_exclusion(image1, 'G')

print("Below is the image without the red channel.")
display(without_red)

print("Below is the image without the green channel.")
display(without_green)

print("Below is the image without the blue channel.")
display(without_blue)
```

Below is the image without the red channel.

Below is the image without the green channel.



Below is the image without the blue channel.

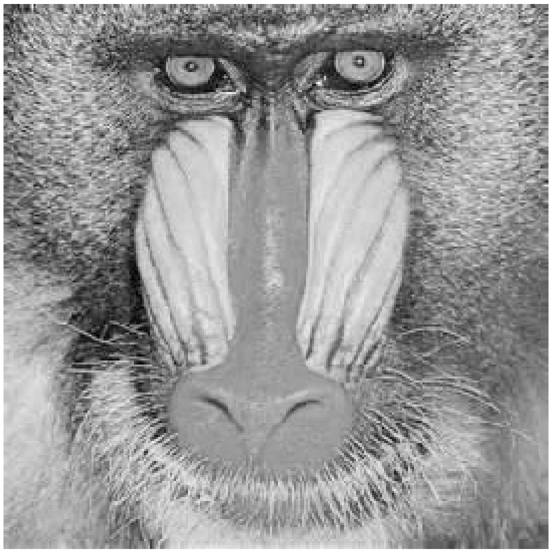


Question 2.5 (10 points)

Implement the lab_decomposition, in which the input image is decomposed into the three channels: L, A and B and return the values for the specified channel.

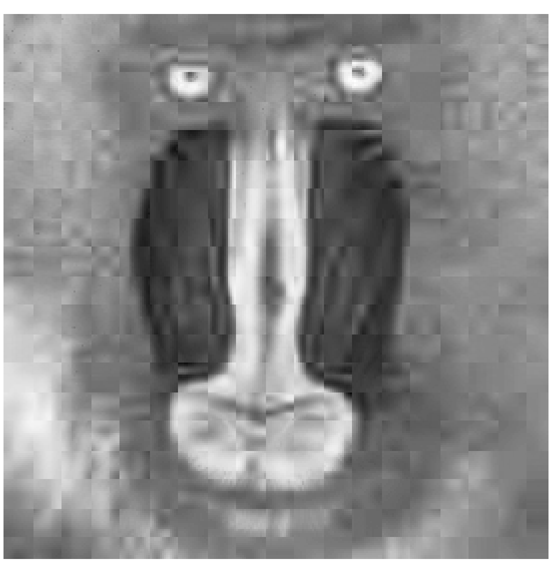
In [83]:
```
image_l = lab_decomposition(image1, 'L')
image_a = lab_decomposition(image1, 'A')
image_b = lab_decomposition(image1, 'B')


print("Below is the image with only the L channel.")
display(image_l)

print("Below is the image with only the A channel.")
display(image_a)

print("Below is the image with only the B channel.")
display(image_b)
```

Below is the image with only the L channel.



Below is the image with only the A channel.

Below is the image with only the B channel.



Question:

Explain in 2-3 sentences what the L, A and B channels are and what happens when you take away the L and A channels.

Answer:

L is the lightness channel and has color range from black to white, A is from green to red and B is from blue to yellow. When we take away L and A, it will only display a color gradient from blue to yellow.

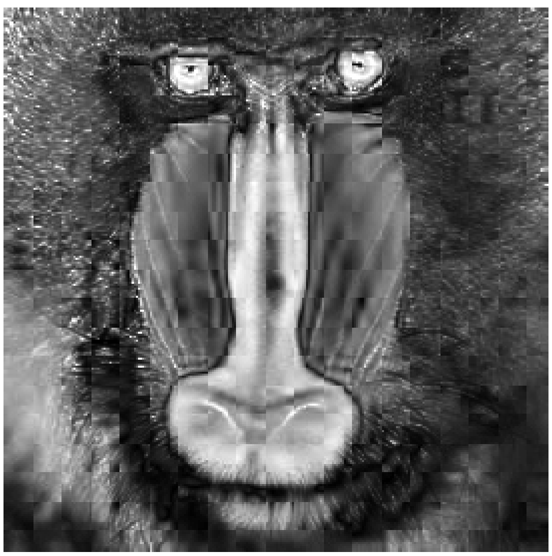
Question 2.6 (10 points)

Implement the `hsv_decomposition()`, in which the input image is decomposed into the three channels: H, S and V and return the values for the specified channel.

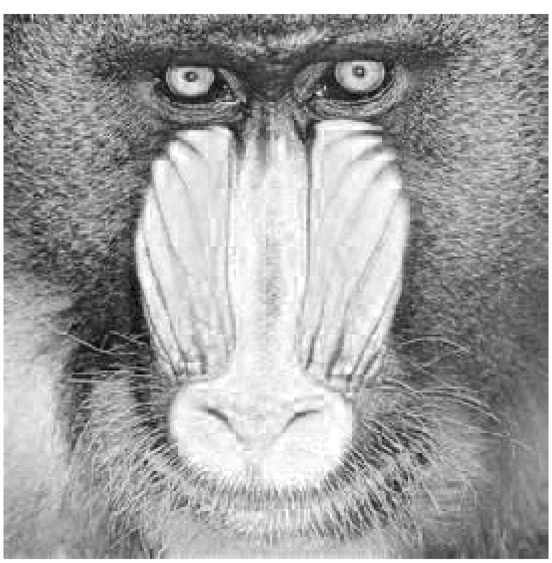
In [86]:
```
image_h = hsv_decomposition(image1, 'H')
image_s = hsv_decomposition(image1, 'S')
image_v = hsv_decomposition(image1, 'V')

print("Below is the image with only the H channel.")
display(image_h)

print("Below is the image with only the S channel.")
display(image_s)

print("Below is the image with only the V channel.")
display(image_v)
```

Below is the image with only the H channel.

Below is the image with only the S channel.



Below is the image with only the V channel.



Question:

Explain in 2-3 sentences what the H, S and V channels are and what happens when you take away the both the H and S channels.

Answer:

HSV correspond to hue, saturation and value. Hue represents the ranges of color, saturation is how intense the color is and value is the brightness from black to white. When we take away H and S, we are only left with V which would only give us a black and white gradient, thus displaying black and white picture(greyscale).

Question 2.7 (10 points)

In mix_images method, create a new image such that the left half of the image is the left half of image1 and the right half of the image is the right half of image2. Exclude the specified channel for the given image.

You should see the left half of the monkey without the red channel and the right half of the house image with no green channel.

In [114]:
```
image_mixed = mix_images(image1, image2, channel1='R', channel2='G')
display(image_mixed)

#Sanity Check: the sum of the image matrix should be 76421.98
np.sum(image_mixed)
```

Out[114]: 76421.98431372548

Extra credit (10 points)

The following questions are optional and will go towards you extra credit grade.

Implement `mix_quadrants` function in `imageManip.py`.

In [115]:
```
mixed_quadrants = mix_quadrants(image1)
display(mixed_quadrants)
```

In [ ]:
```
```