In [1]:

```
import cv2
import numpy as np
from math import sqrt
```

In [2]:

```
img1 = cv2.imread('ps0-1-a-1.jpg', cv2.IMREAD_COLOR)
img2 = cv2.imread('ps0-1-a-2.jpg', cv2.IMREAD_COLOR)
```

2.Color planes

a. Swap the red and blue pixels of image 1.

Output:

In [3]:

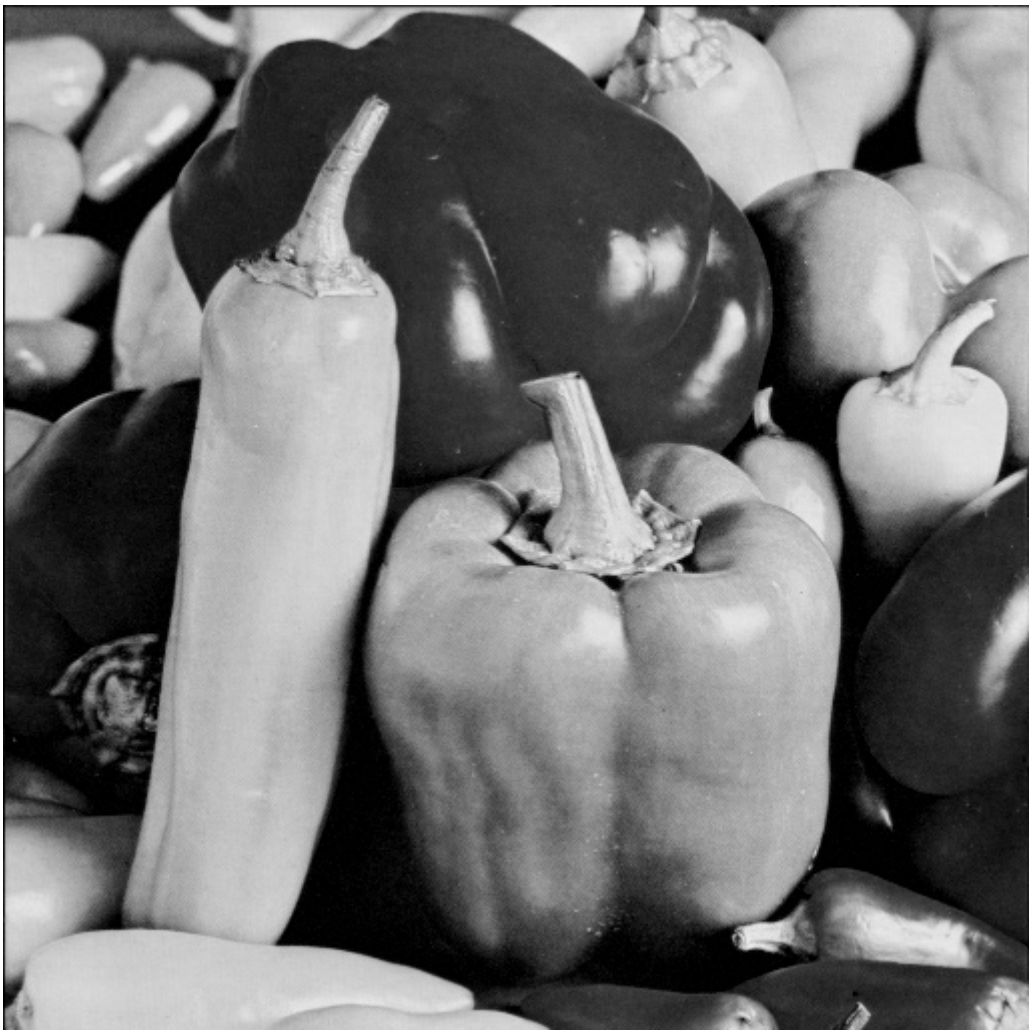
```
img1_swapRB = img1[:,:,::-1]
cv2.imwrite('ps0-2-a.jpg', img1_swapRB)
```

Out[3]:

True

b. Create a monochrome image (M1g) by selecting the green channel of image 1

Output:



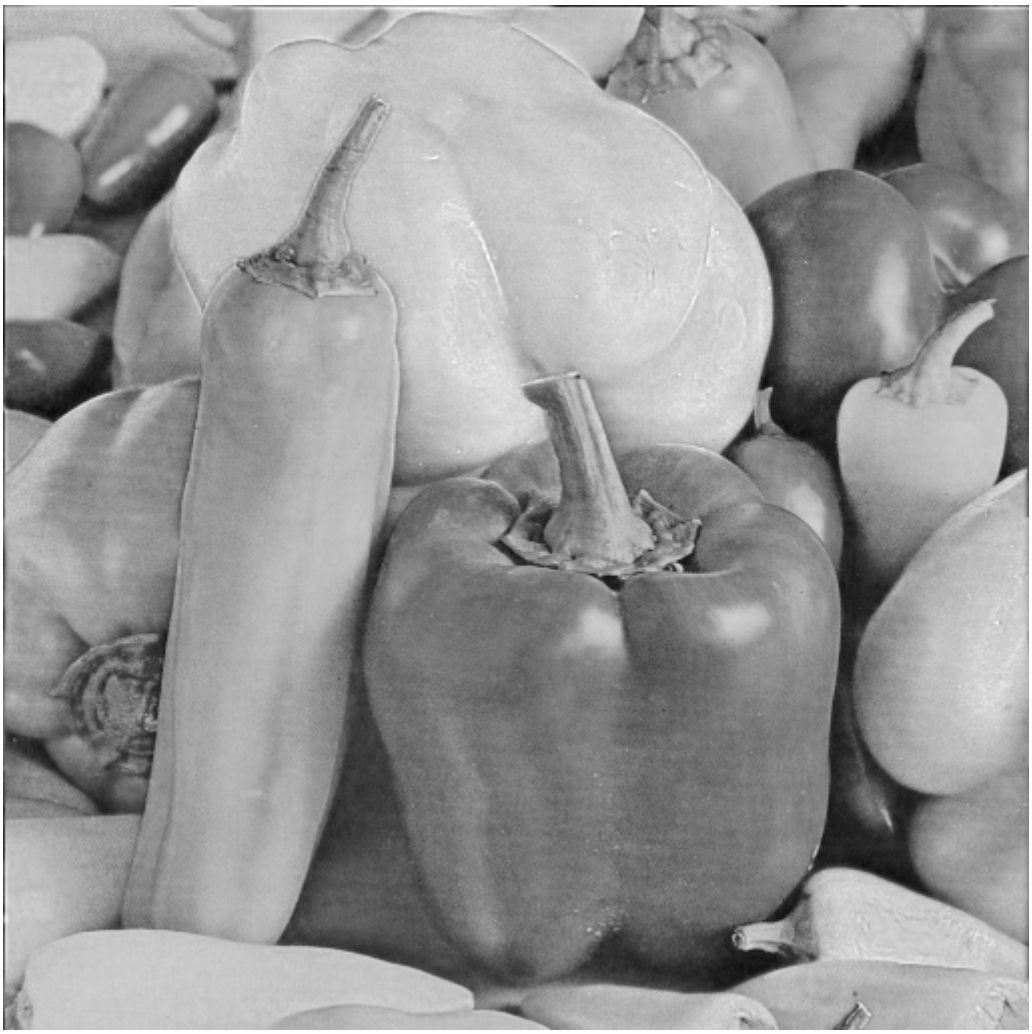
In [4]:

```
m1g = img1[:,:,1]
cv2.imwrite('ps0-2-b.jpg', m1g)
```

Out[4]:

True

c. Create a monochrome image (M1r) by selecting the red channel of image 1
Output:



In [5]:

```
m1r = img1[:,:,2]
cv2.imwrite('ps0-2-c.jpg', m1r)
```

Out[5]:

True

d. Which looks more like what you'd expect a monochrome image to look like? Would you expect a computer vision algorithm to work on one better than the other?

Response: The image selected by the green channel looks better; the grayscale is larger than the image of the red channel. The green scale of the original image is larger than it's red scale.

We can expand the scale of the image produced by c). For example, the min pixel is 100, and the max pixel is 200, we project [100, 200] to [0, 255].

3 Replacement of pixels

a. Take the inner square of 100x100 pixels of monochrome version of image 1 and insert them into the monochrome version of image 2

Output:

In [6]:

```
img_replaced = img2
img_replaced[78:178,78:178,2] = m1r[206:306,206:306]
cv2.imwrite('ps0-3-a.jpg', img_replaced)
```

Out[6]:

```
True
```

4 Arithmetic and Geometric operation

a. What is the min and max of the pixel values of M1g? What is the mean? What is the standard deviation? And how did you compute these?

In [7]:

```
m1g_mean = np.mean(m1g)
m1g_std = np.std(m1g)
print("Max of M1g {}".format(np.max(m1g)))
print("Min of M1g {}".format(np.min(m1g)))
print("Mean of M1g {}".format(m1g_mean))
print("Standard deviation of M1g {}".format(m1g_std))
```

```
Max of M1g 239
Min of M1g 0
Mean of M1g 115.65623092651367
Standard deviation of M1g 74.47395743997122
```

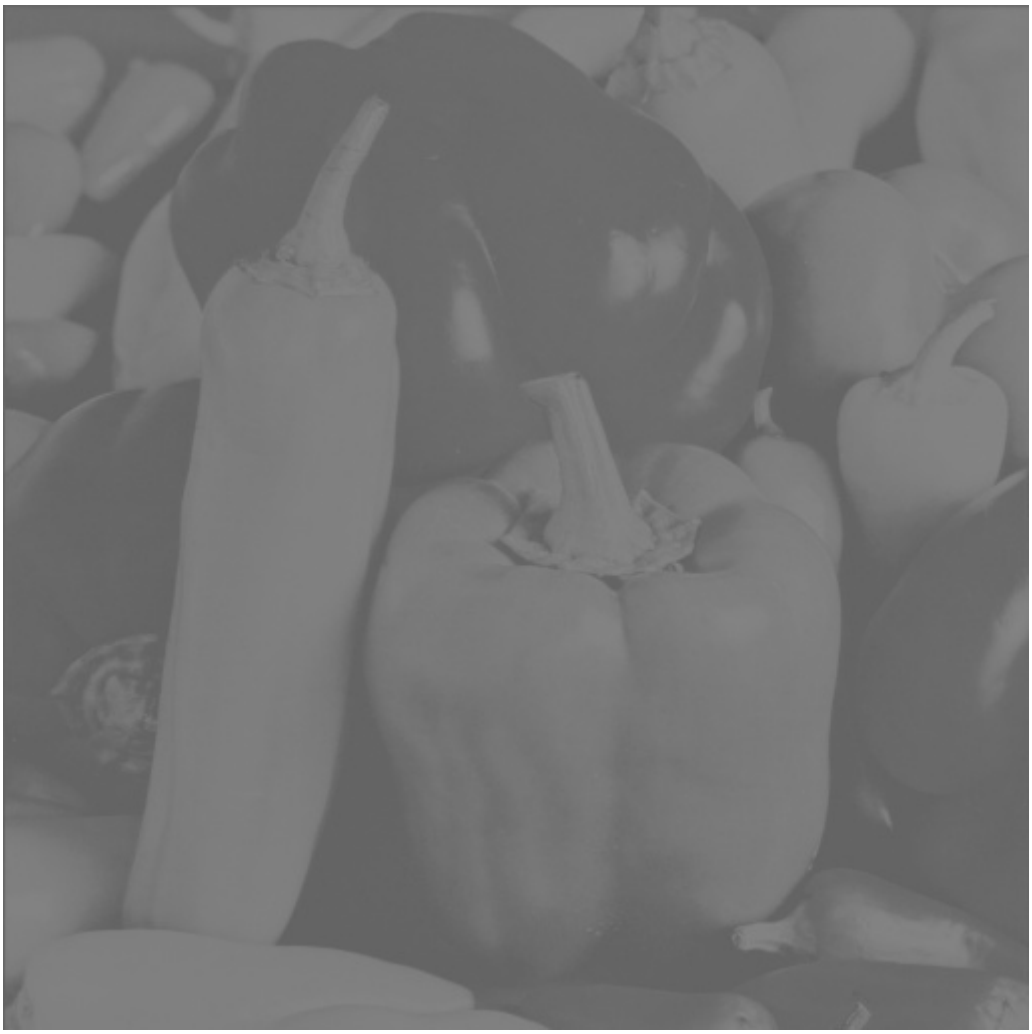
Compute with numpy package or as below:

In [8]:

```
m1g_max = 0
m1g_min = 255
m1g_sum = 0
for i in range(512):
    for j in range(512):
        if m1g[i][j] > m1g_max:
            m1g_max = m1g[i][j]
        if m1g[i][j] < m1g_min:
            m1g_min = m1g[i][j]
        m1g_sum += m1g[i][j]
m1g_avg = m1g_sum / (512 * 512)
m1g_err = 0
for i in range(512):
    for j in range(512):
        m1g_err += (m1g[i][j] - m1g_avg) ** 2
print(m1g_max)
print(m1g_min)
print(m1g_avg)
print(sqrt(m1g_err / (512 * 512)))
```

```
239
0
115.65623092651367
74.47395743996881
```

b. Subtract the mean from all the pixels, then divide by the standard deviation, then multiply by 10 (if your image is zero to 255) or by 0.05 (if your image ranges from 0.0 to 1.0). Now add the mean back in.
Ouptut:



In [9]:

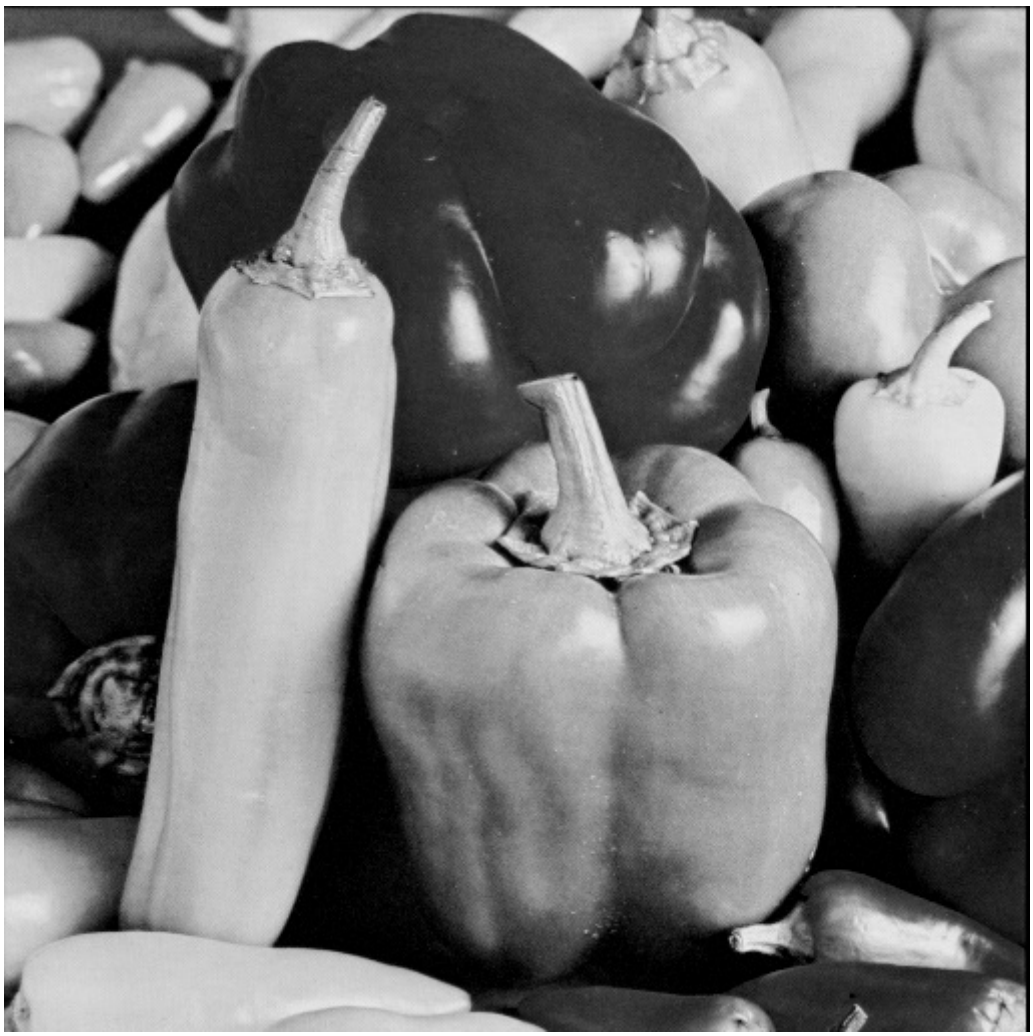
```
m1g_copy = m1g.copy()
m1g_copy = np.subtract(m1g_copy, m1g_mean)
# cv2.imwrite("m1g2-1.png", m1g_copy)
m1g_copy = np.divide(m1g_copy, m1g_std)
# cv2.imwrite("m1g2-2.png", m1g_copy)
m1g_copy = np.multiply(m1g_copy, 10)
# cv2.imwrite("m1g2-3.png", m1g_copy)
m1g_copy = np.add(m1g_copy, m1g_mean)
cv2.imwrite("ps0-4-b.jpg", m1g_copy)
```

Out[9]:

```
True
```

c. Shift M1g to the left by 2 pixels.

Output:



In [10]:

```
m1g_shifted = np.hstack((m1g[:, 2:], np.zeros((512, 2))))

cv2.imwrite('ps0-4-c.jpg', m1g_shifted)
```

Out[10]:

```
True
```

d. Subtract the shifted version of M1g from the original and make sure tha t the values are legal (what do negative numbers for pixels mean anyway?).

In [11]:

```
m1g_sub = np.subtract(m1g, m1g_shifted)
for i in range(512):
    for j in range(512):
        if m1g_sub[i][j] < 0:
            m1g_sub[i][j] = 0
cv2.imwrite('ps0-4-d.jpg', m1g_sub)
```

Out[11]:

```
True
```

5 Noise

a. Take the original colored image and start adding Gaussian noise to the pixels in the green channel. Increase sigma until the noise is somewhat visible.

Output:

Response: Sigma represents the spread width of the gaussian distribution, square of the standard deviation. In the gaussian noise, it is the visibility of the noise pixels

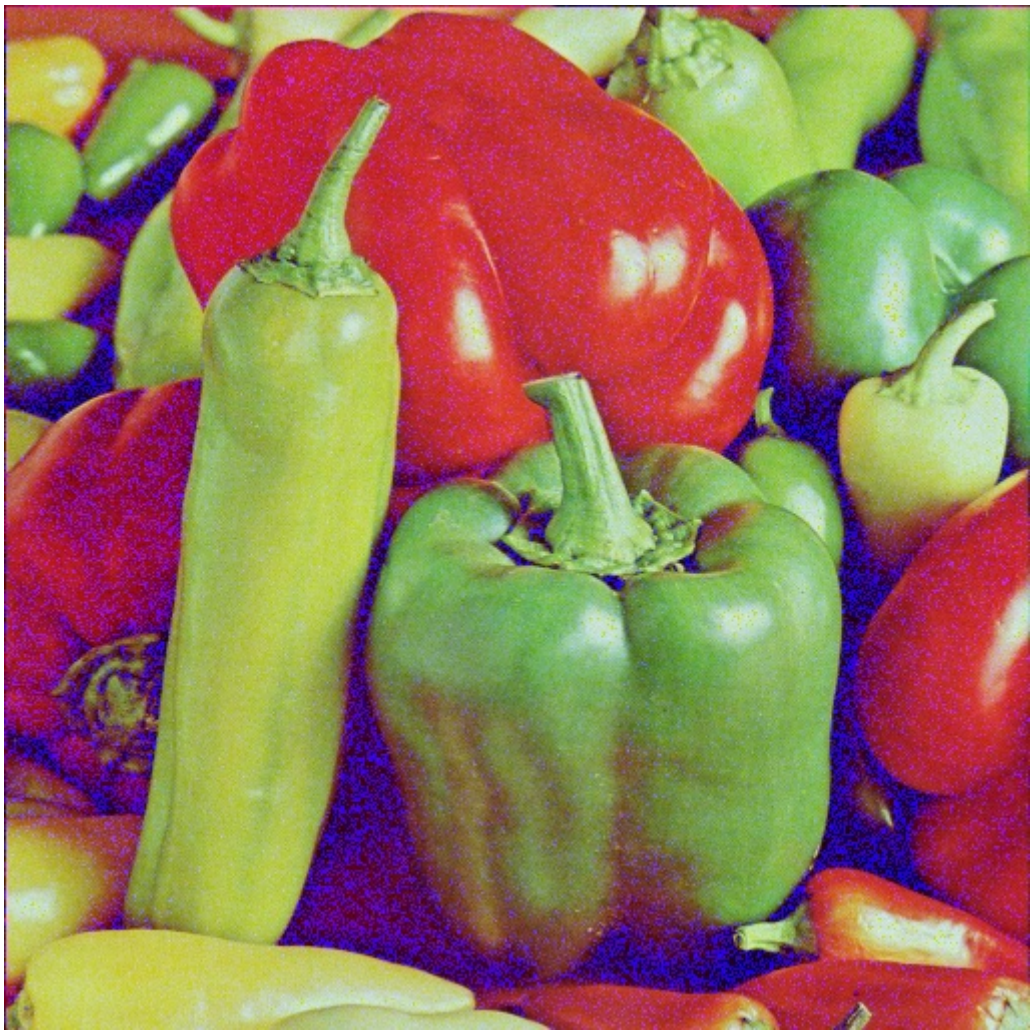
In [12]:

```
mean = 0
sigma = 30

noise = np.random.normal(mean, sigma, img1[:,:,1].shape)
img_noisy = img1.copy()
img_noisy[:, :, 1] = img_noisy[:, :, 1] + noise

img_noisy[:, :, 1] = np.clip(img_noisy[:, :, 1], 0, 255)
# img_noisy = np.uint8(img_noisy * 255)
cv2.imwrite("ps0-5-a.jpg", img_noisy)
```

Out[12]:

```
True
```

b. Now, instead add that amount of noise to the blue channel.
Output:



In [13]:

```
img_noisy_b = img1.copy()
img_noisy_b[:, :, 0] = img_noisy_b[:, :, 0] + noise

img_noisy_b[:, :, 0] = np.clip(img_noisy_b[:, :, 0], 0, 255)
cv2.imwrite("ps0-5-b.jpg", img_noisy_b)
```

Out[13]:

True

c. Which looks better? Why?
Response: The noise added to the green channel is more evident because mos t of the original image has low blue channel values.

In [ ]:

```

```