Homework 1

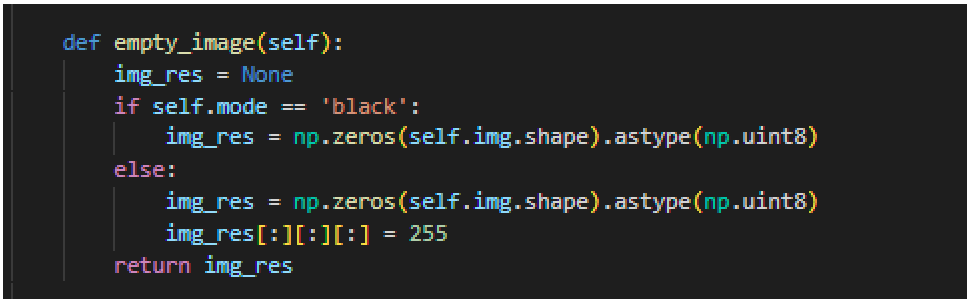
Problem 1

1) **Time required to transmit image= image size (in bits) /modem rate (in bits/sec)**
    - **= (1024x1024x(8+2) (bits) / 10e6(bits/sec))x500**
    - **= 5242.88 sec**
2) **Time required to transmit image= image size (in bits) /modem rate (in bits/sec)**
    - **= (1024x1024x(8+2) (bits) / 10e9(bits/sec))x500**
    - **= 5.24288 sec**

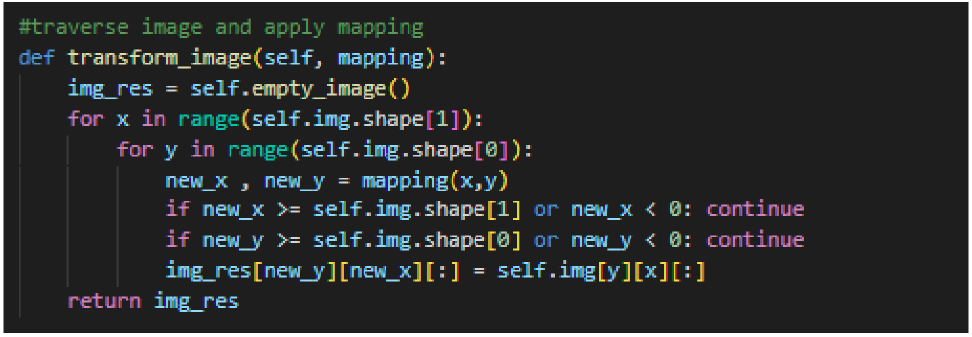
Problem 2

Generating Image:

This was done by first generating a blank image the color of the selected mode and shape of the given original image, “self.mode” is set by default to black which is shown in the constructor. After generating a blank image, we traverse through each element of the original image and map its current (x, y) coordinated to a new position based on the transformation being applied. The mapping function done by calling a helper function that outputs the results from the equation linked to the transformation. This could also be done my doing vector multiplication on a matrix (rotate/ remap the vector). For ease of implementation, I used the vector functions directly.

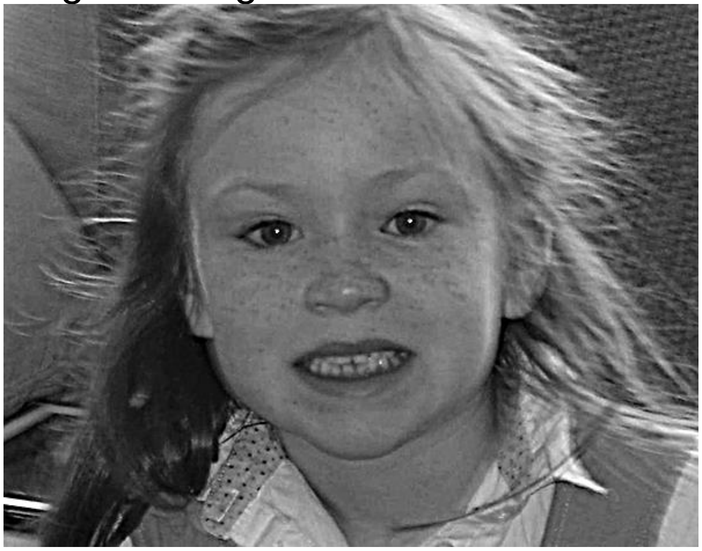


```
    def empty_image(self):
        img_res = None
        if self.mode == 'black':
            img_res = np.zeros(self.img.shape).astype(np.uint8)
        else:
            img_res = np.zeros(self.img.shape).astype(np.uint8)
            img_res[:][:][:] = 255
        return img_res
```

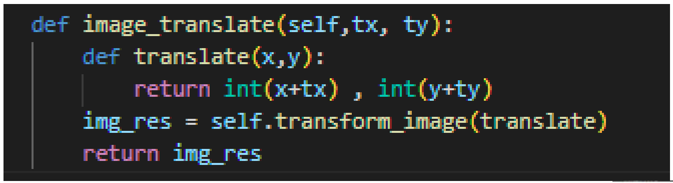


```
#traverse image and apply mapping
def transform_image(self, mapping):
    img_res = self.empty_image()
    for x in range(self.img.shape[1]):
        for y in range(self.img.shape[0]):
            new_x , new_y = mapping(x,y)
            if new_x >= self.img.shape[1] or new_x < 0: continue
            if new_y >= self.img.shape[0] or new_y < 0: continue
            img_res[new_y][new_x][:] = self.img[y][x][:]
    return img_res
```

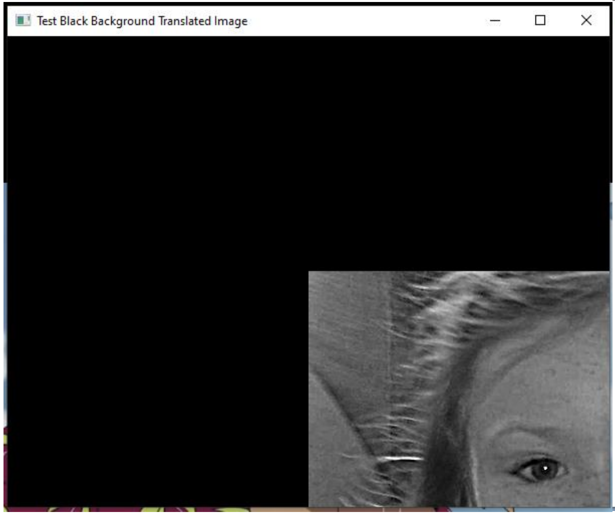
Original Image:

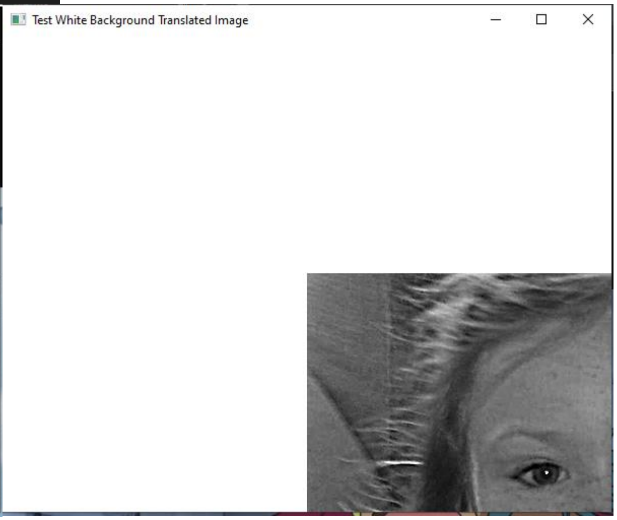


Affine Translation:

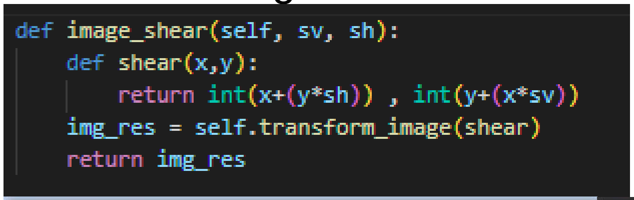


```python
def image_translate(self,tx, ty):
    def translate(x,y):
        return int(x+tx) , int(y+ty)
    img_res = self.transform_image(translate)
    return img_res
```

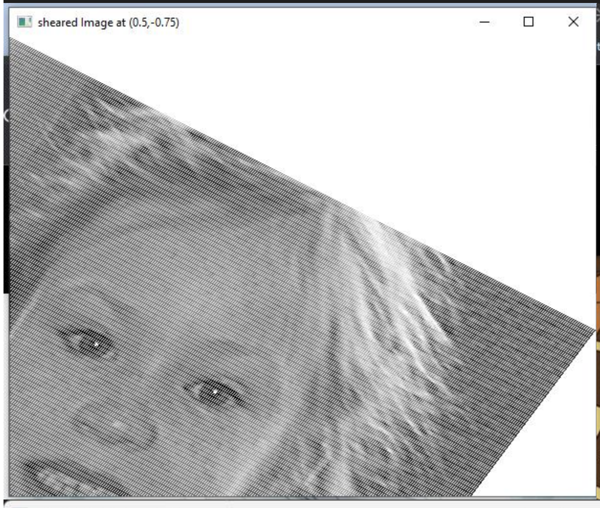


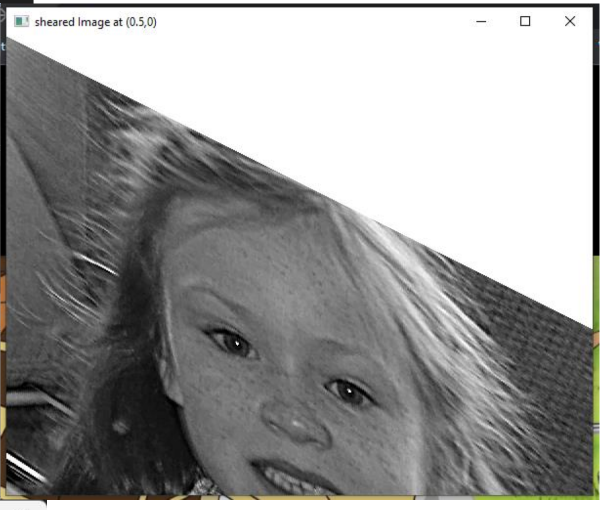


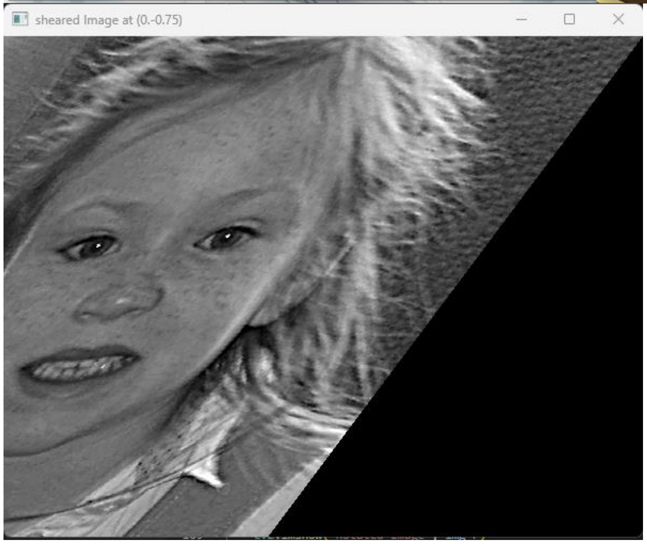
Affine Shearing:



```
def image_shear(self, sv, sh):
    def shear(x,y):
        return int(x+(y*sh)) , int(y+(x*sv))
    img_res = self.transform_image(shear)
    return img_res
```


sheared Image at (0.5,-0.75)


sheared Image at (0.5,0)


sheared Image at (0,-0.75)