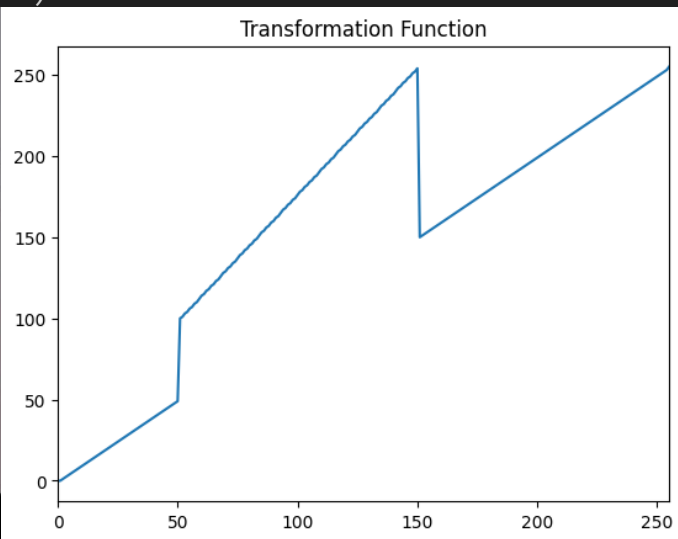
EN3160: Image Processing and Machine Vision GAMIDU.A.G.D 200179H

Assignment 01

Github link- https://github.com/DusaraG/Assignment-1/blob/main/Assignment1.ipynb

**Question 1**

```
arr1=np.linspace(0 , 49,51).astype('uint8')

arr2=np.linspace(50+50,254.5,100).astype('uint8')
arr3=np.linspace(150,255,105).astype('uint8')
arr4=np.concatenate((arr2,arr3))
transform=np.concatenate((arr1,arr4))
img2 = cv.LUT(original, transform)
```



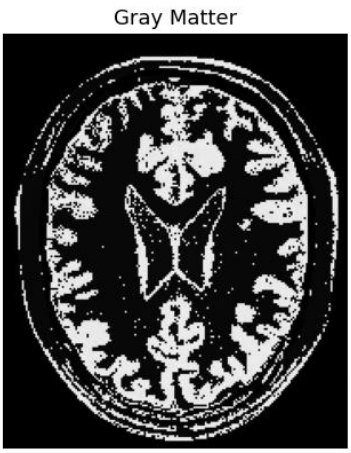

**Question 2**

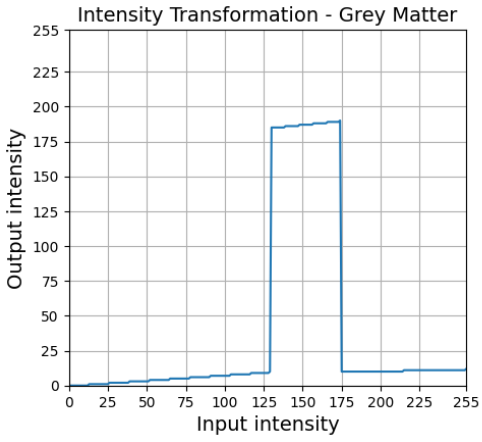
```
t1 = np.linspace(0, 10, 130)

t2 = np.linspace(185, 190, 45)
t3 = np.linspace(10, 12, 81)

z1 = np.linspace(0, 10, 185)
z2 = np.linspace(230, 240, 35)
z3 = np.linspace(10, 12, 36)
```

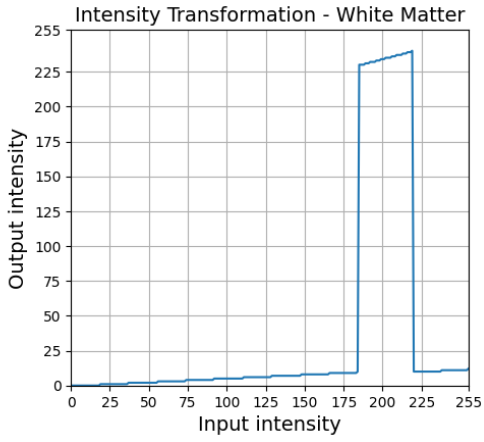
```
t = np.concatenate((t1,t2,t3),axis=0).astype(np.uint8)
z = np.concatenate((z1, z2, z3), axis = 0).astype(np.uint8)
```

```
g = cv.LUT(im,t)
h = cv.LUT(im,z)
```





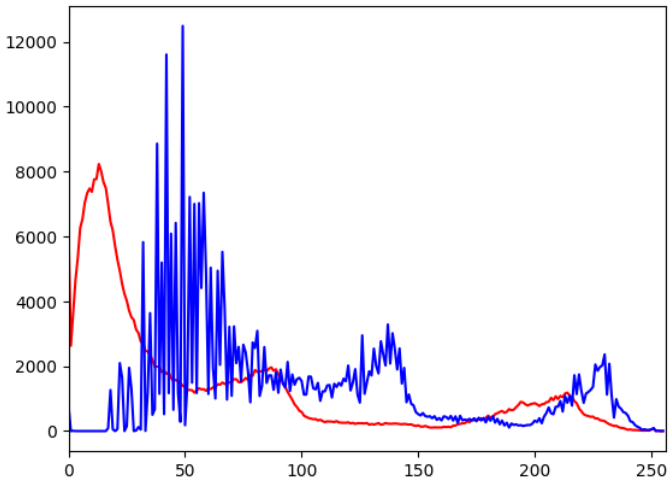




**Question 3** used gamma value=0.5

```
original = cv.imread("highlights_and_shadows.jpg", cv.IMREAD_COLOR)

lab = cv.cvtColor(original, cv.COLOR_BGR2LAB)
l, a, b = cv.split(lab)

#apply gamma correction to the l channel
gamma = 0.5
l = (np.power(l/255, gamma) * 255).astype('uint8')
```

```
#merge the channels back together and convert back to bgr
lab = cv.merge((l, a, b))
img2 = cv.cvtColor(lab, cv.COLOR_LAB2BGR)

#show the histograms of the original and modified images
hist1=cv.calcHist([original], [0], None, [256], [0, 256])
hist2=cv.calcHist([img2], [0], None, [256], [0, 256])
```

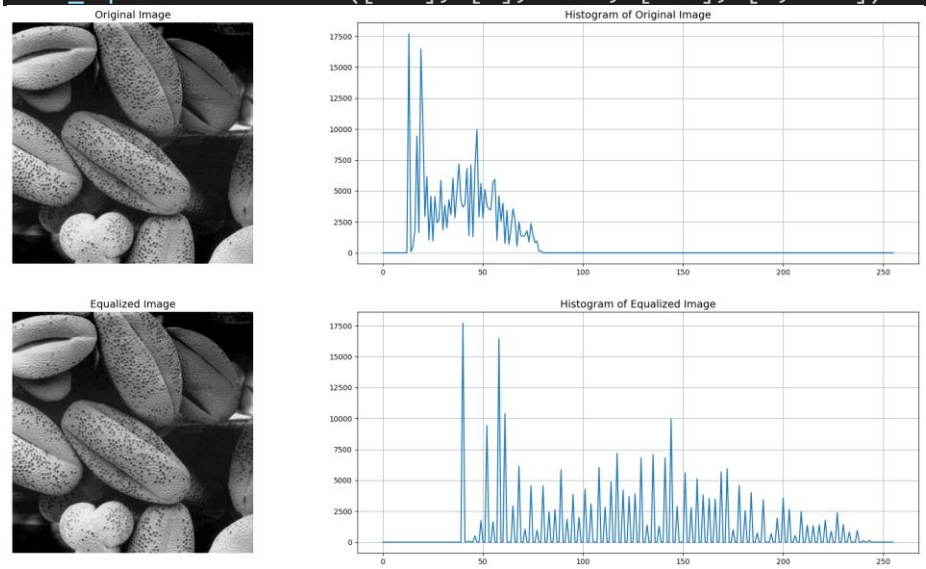

**Question 4**

```
original= cv.imread("spider.png", cv.IMREAD_COLOR)
hsv = cv.cvtColor(original, cv.COLOR_BGR2HSV)
h, s, v = cv.split(hsv)
x= np.linspace(0,255,256)

fig,ax=plt.subplots(1,5,figsize=(15,10))

for a in range(0,5):
```

```
    transform = np.minimum(x+a//5*128*np.exp(-(x-128)**2/(2*70**2)),
255).astype('uint8')
    h=cv.LUT(h,transform)
    hsv = cv.merge((h, s, v))
    img2 = cv.cvtColor(hsv, cv.COLOR_HSV2RGB)
```


**Question 5**

```
ims = im.copy()

# Calculate the histogram of the original image
hist = cv2.calcHist([im], [0], None, [256], [0, 256])

# Find the low and high values in the histogram
low = np.min(np.where(hist > 0))
high = np.max(np.where(hist > 0))

# Perform histogram equalization
ims = np.round((ims - low) * ((255 - 0) / (high - low))).astype(np.uint8)

# Calculate the histogram of the equalized image
hist_eq = cv2.calcHist([ims], [0], None, [256], [0, 256])
```

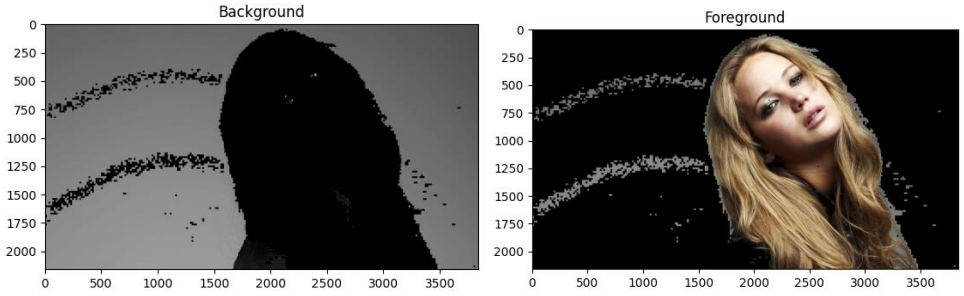


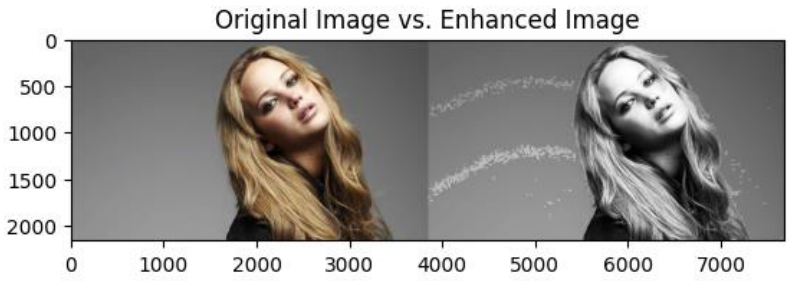
**Question 6**

```
original = cv.imread("jeniffer.jpg", cv.IMREAD_COLOR)

# Convert the image to HSV color space
hsv = cv.cvtColor(original, cv.COLOR_BGR2HSV)

# Split the image into hue, saturation, and value channels
h, s, v = cv.split(hsv)
```

```
ret, mask = cv.threshold(s, 5, 255, cv.THRESH_BINARY)

# Invert the mask
mask_inv = cv.bitwise_not(mask)

# Separate the background and foreground
img1_bg = cv.bitwise_and(original, original, mask=mask_inv)
img1_fg = cv.bitwise_and(original, original, mask=mask)
```

```
fgndEq = cv.equalizeHist(cv.cvtColor(img1_fg, cv.COLOR_RGB2GRAY))

# Combine the equalized foreground with the background
result = cv.add(cv.cvtColor(fgndEq, cv.COLOR_GRAY2RGB), img1_bg)
```

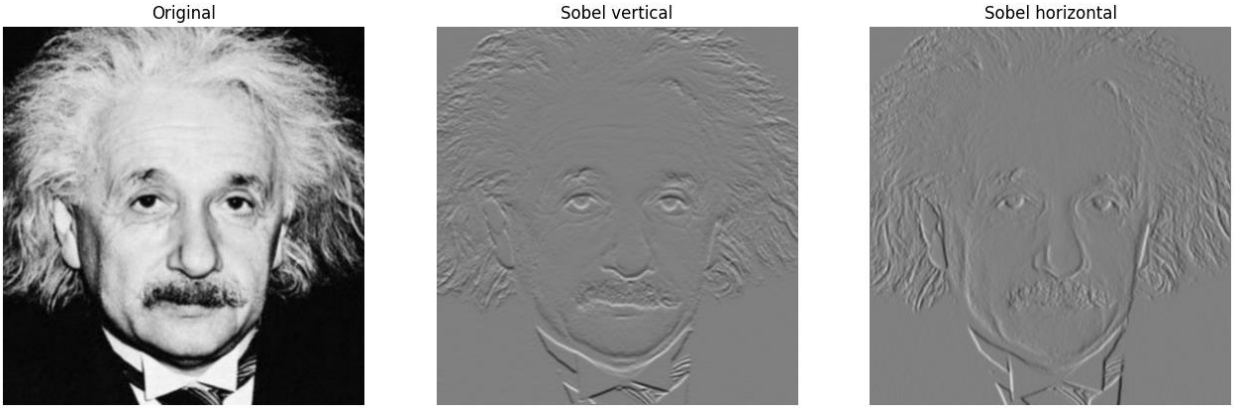


**Question 7**

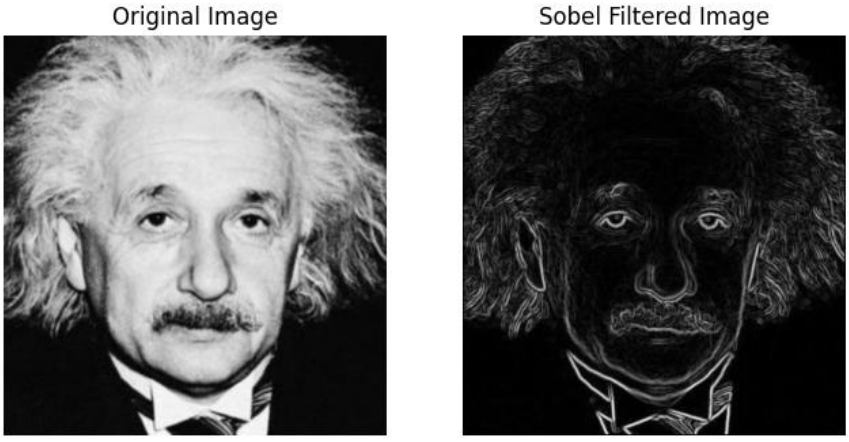
```
sobel_v = np.array([[-1 , -2, -1] , [0 ,0 , 0] , [1 , 2, 1] ], dtype=np.float32)
f_x = cv.filter2D(im, -1, sobel_v)
```

```
sobel_h = np.array([[-1 , 0, 1] , [-2 ,0 , 2] , [-1 , 0, 1] ], dtype=np.float32)
f_h = cv.filter2D(im, -1, sobel_h)
```




Manual Sobel implementation

```
for i in range(1,im.shape[0]-1):
    for j in range(1,im.shape[1]-1):
        f_xx = im[i-1, j-1]*(-1) + im[i-1, j]*(-2) + im[i-1, j+1]*(-1) + im[i+1,
j-1]*1 + im[i+1, j]*2 + im[i+1, j+1]*1
        f_hh = im[i-1, j-1]*(-1) + im[i, j-1]*(-2) + im[i-1, j+1]*1 + im[i+1, j-
1]*(-1) + im[i, j+1]*2 + im[i+1, j+1]*1
        grad = np.sqrt(f_xx**2 + f_hh**2)
        im_g[i, j] = (grad / 1020) * 255
```



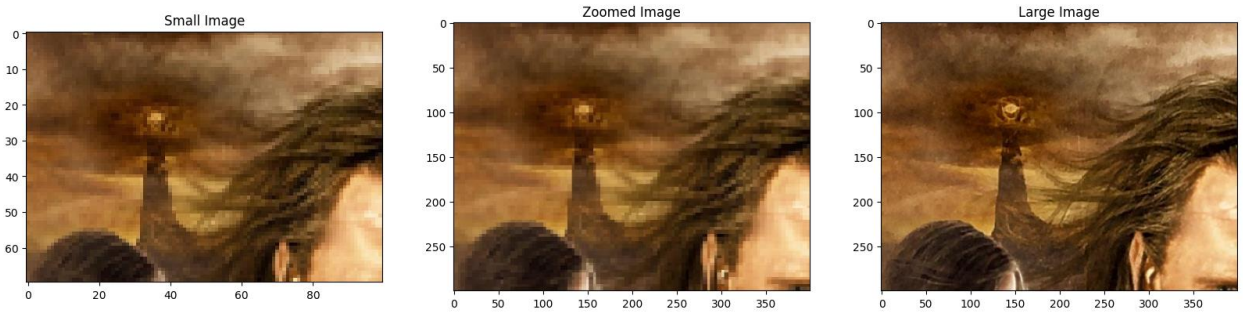
Using the given property

```
im_gd = np.zeros((im.shape[0],im.shape[1]),np.uint8)

mat = np.multiply((np.array([[1],[2],[1]], dtype=np.int32)),(np.array([1,0,-1],
dtype=np.int32)))

sobel_hh = np.array([[0,0,0],[0,0,0],[0,0,0]])
for i in range(len(mat)):
    sobel_hh[i] = mat[i][::-1]

sobel_vv = sobel_hh.T

for i in range(1,im.shape[0]-1):
    for j in range(1,im.shape[1]-1):
        image = np.array([[im[i-1,j-1],im[i,j],im[i+1,j+1]], [im[i,j-
1],im[i,j],im[i,j+1]], [im[i+1,j-1],im[i+1,j],im[i+1,j+1]]], dtype=np.float32)
        ##print(image)
        val_xx = 0
        val_yy = 0
        for k in range(3):
            for m in range(3):
                val_xx += image[k][m]*sobel_hh[k][m]
                val_yy += image[k][m]*sobel_vv[k][m]
        ##print(val_xx, val_yy)
        grad = np.sqrt(val_xx**2 + val_yy**2)
        im_gd[i, j] = (grad / 1020) * 255
```

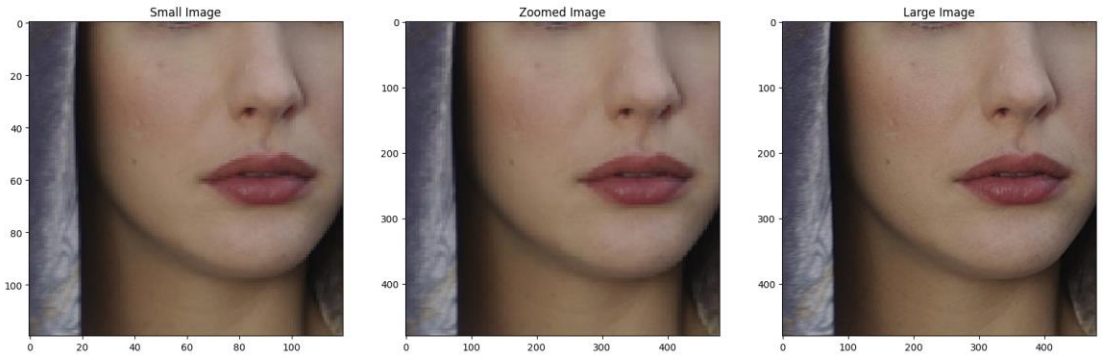
**Question 8**

```
def nn_zoom(image, scaling_factor):
    s = scaling_factor
    img = image

    row = img.shape[0] * s
    column = img.shape[1] * s
    zoomed_Img = np.zeros((row, column, 3), dtype = np.uint8)
    for i in range(row):
        for j in range(column):
                zoomed_Img[i, j] = img[round(i/s - 0.5), round(j/s - 0.5)]

    return zoomed_Img
```

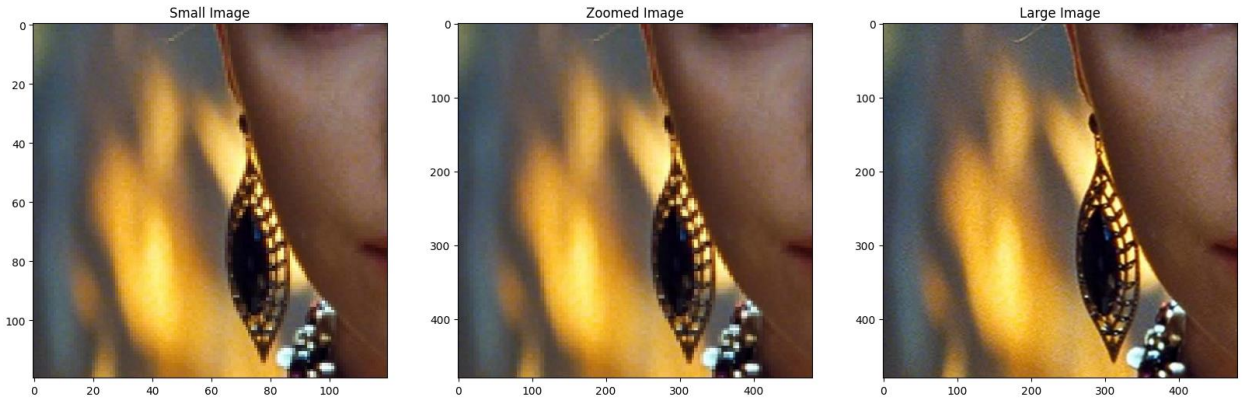
```
im = cv.imread("zooming\im01small.png")
im_large = cv.imread("zooming\im01.png")
assert im is not None

im_znn = nn_zoom(im, 4)
def SSD(img1,img2):
    val = (np.sum((img1.astype("float")-
img2.astype("float"))**2))/float(img1.shape[0]*img1.shape[1])
    return val
```




Ssd=489.0480126350309




ssd=101



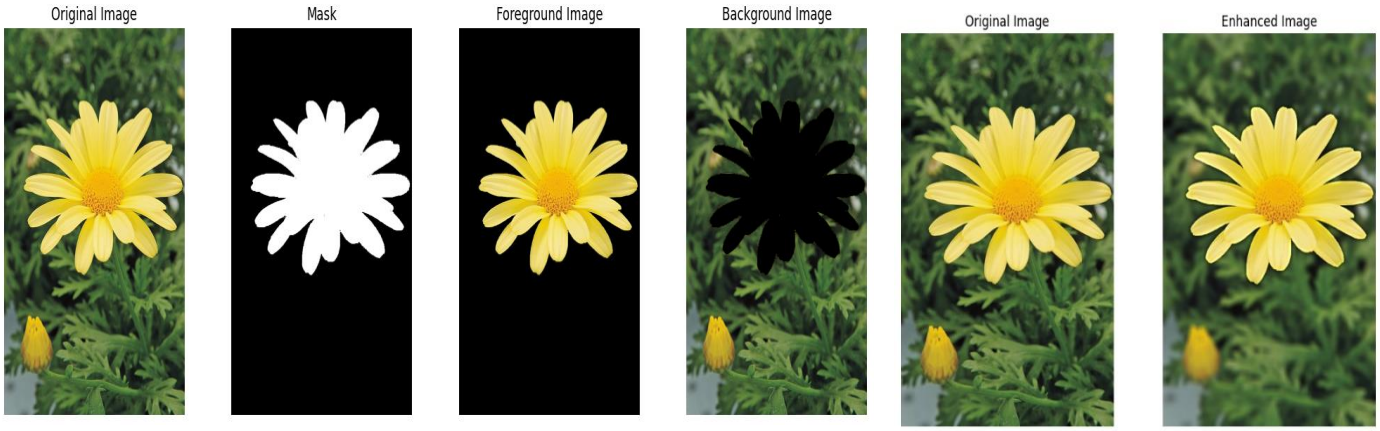
Function for bilinear zoom

```
import math

def bl_zoom(Image, new_h, new_w):
    im = Image
    old_h, old_w, c = im.shape
    resized = np.zeros((new_h, new_w, c))

    if new_h !=0:
         w_scale_factor = (old_w) / (new_w)
    else:
        w_scale_factor = 0

    if new_w != 0:
         h_scale_factor = (old_h) / (new_h)
    else:
         h_scale_factor = 0

    for i in range(new_h):
        for j in range(new_w):
            x = i * h_scale_factor
            y = j * w_scale_factor

            x_floor = math.floor(x)
            x_ceil = min(old_h - 1, math.ceil(x))
            y_floor = math.floor(y)
            y_ceil = min(old_w - 1, math.ceil(y))
            if (x_ceil == x_floor) and (y_ceil == y_floor):
                q = im[int(x), int(y), :]

            elif (x_ceil == x_floor):
                a = im[int(x), int(y_floor), :]
                b = im[int(x), int(y_ceil), :]
                q = a * (y_ceil - y) + b * (y - y_floor)

            elif (y_ceil == y_floor):
                a = im[int(x_floor), int(y), :]
                b = im[int(x_ceil), int(y), :]
                q = (a * (x_ceil - x)) + (b * (x - x_floor))

            else:
                m = im[x_floor, y_floor, :]
                n = im[x_ceil, y_floor, :]
                p = im[x_floor, y_ceil, :]
```

```
                r = im[x_ceil, y_ceil, :]

                a = m * (x_ceil - x) + n * (x - x_floor)
                b = p * (x_ceil - x) + r * (x - x_floor)
                q = a * (y_ceil - y) + b * (y - y_floor)

            resized[i,j,:] = q

    return resized.astype(np.uint8)
```

**Question 9**

```
im = cv.imread("daisy.jpg", cv.IMREAD_COLOR)
im_original = im.copy()

mask = np.zeros(im.shape[:2], np.uint8)
rect = (0, 90, 560, 500)
fgdModel = np.zeros((1, 65), np.float64)
bgdModel = np.zeros((1, 65), np.float64)

cv.grabCut(im, mask, rect, bgdModel, fgdModel, 5, cv.GC_INIT_WITH_RECT)

mask1 = np.where((mask==0) | (mask==2), 0, 1).astype("uint8")
im_fgd = im*mask1[:, :, np.newaxis]
mask2 = np.where((mask==1) | (mask==3), 0, 1).astype("uint8")
im_bgd = im*mask2[:, :, np.newaxis]
```




```
im_blurred = cv.blur(im_bgd, (15, 15))
im_enhanced = cv.add(im_blurred, im_fgd)
```