ELE 492: Image Processing
Assoc. Prof. Seniha Esen Yuksel
Department of Electrical and Electronics Engineering
Hacettepe University

HW-1

Name: Furkan ÇÖZELİ **17.03.201**

#: 21427837

1) I have not received or given any aid in this homework. All the work presented below is my own work.

Furkan ÇÖZELİ

2) Function: readImagesFromFile

With this function, we can get all the photos from a file.

```
def readImagesFromFile(path):
    images = []
    for image in os.listdir(path):
        img = cv2.imread(os.path.join(path,image))
        if img is not None:
            images.append(img)
    return images
```

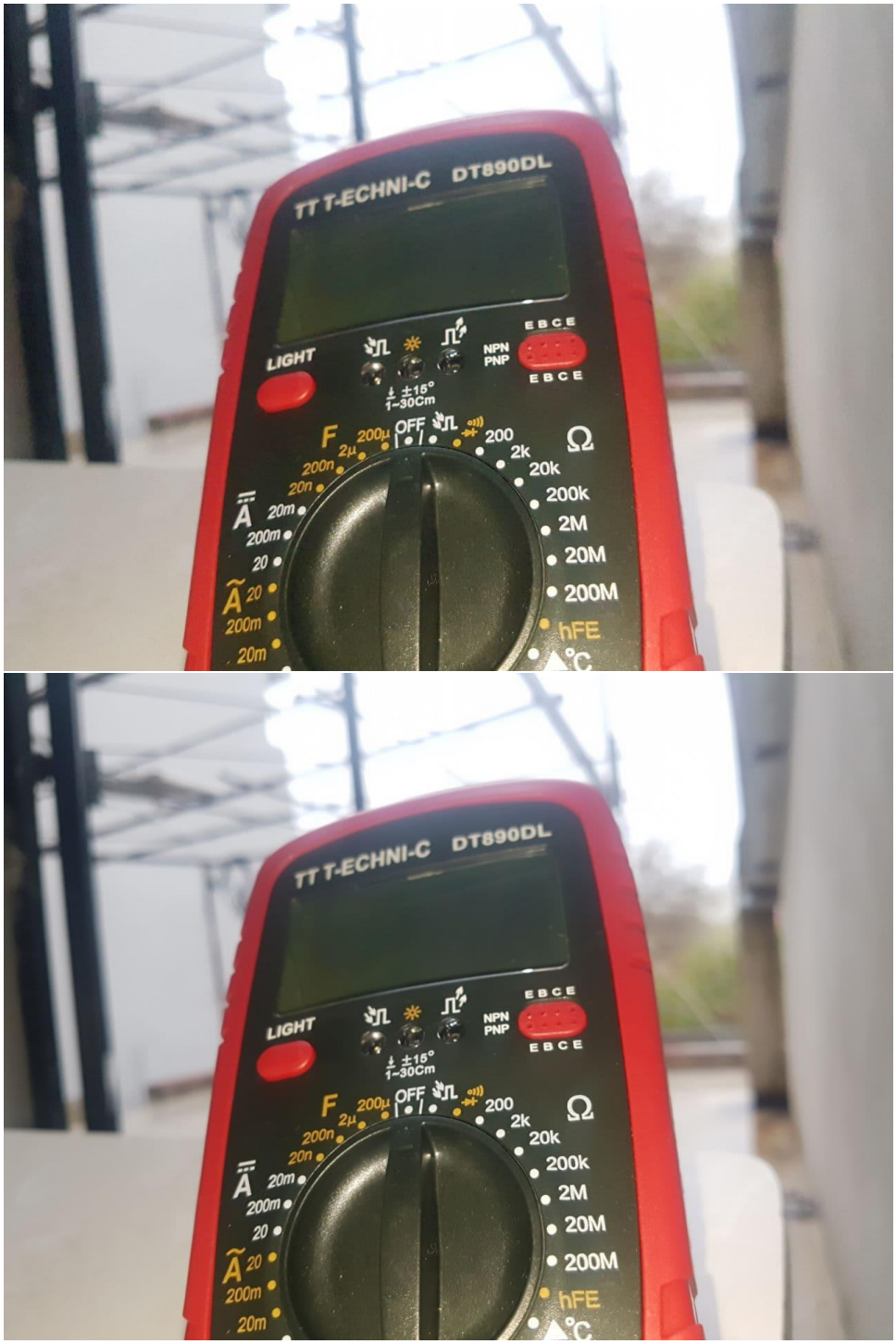
Function: averageImages

With this function, we can average the pixels of more than one photo. This is used to reduce the noise in photography.

```
def averageImages(images):
    numberOfImages = len(images)
    row,col,px = images[0].shape
    newImg = np.zeros((row,col,px),np.int64)

    for i in range(numberOfImages):
        newImg += images[i]
    newImg = newImg / numberOfImages
    newImg = newImg.astype(np.uint8)

    return newImg
```

In order to use this function, it is necessary to take a lot of photos from the same place. I added one of the photos I took below. The other photo is the one I got after applying the function. If you examine the right side of both photos, you can see the noise reduction.

Example:

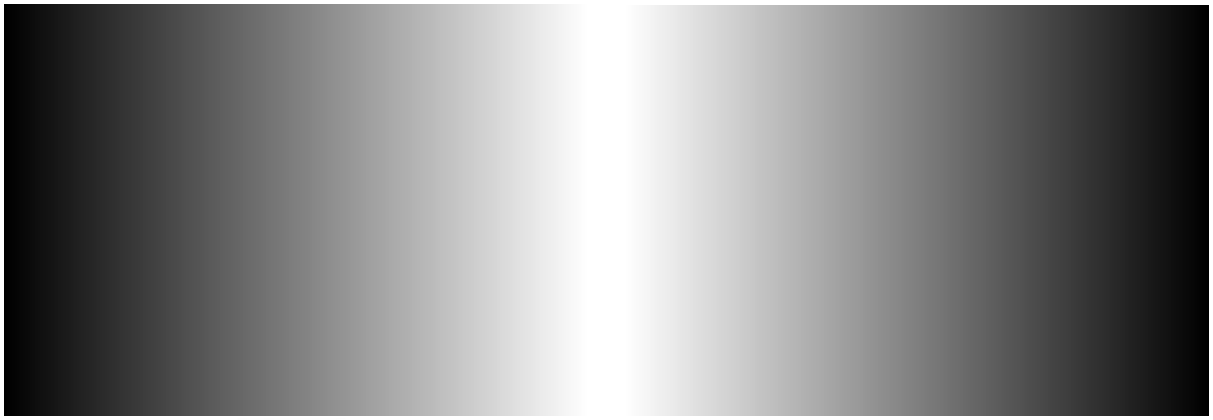


3) Function: negativeImg

```
def negativeImg(img):
    img[:,:] = [255] - img[:,:]
    return img
```

If we subtract the pixel value of the image from 255, we get a negative image. That is, we reverse the pixel values in the image.

Example:

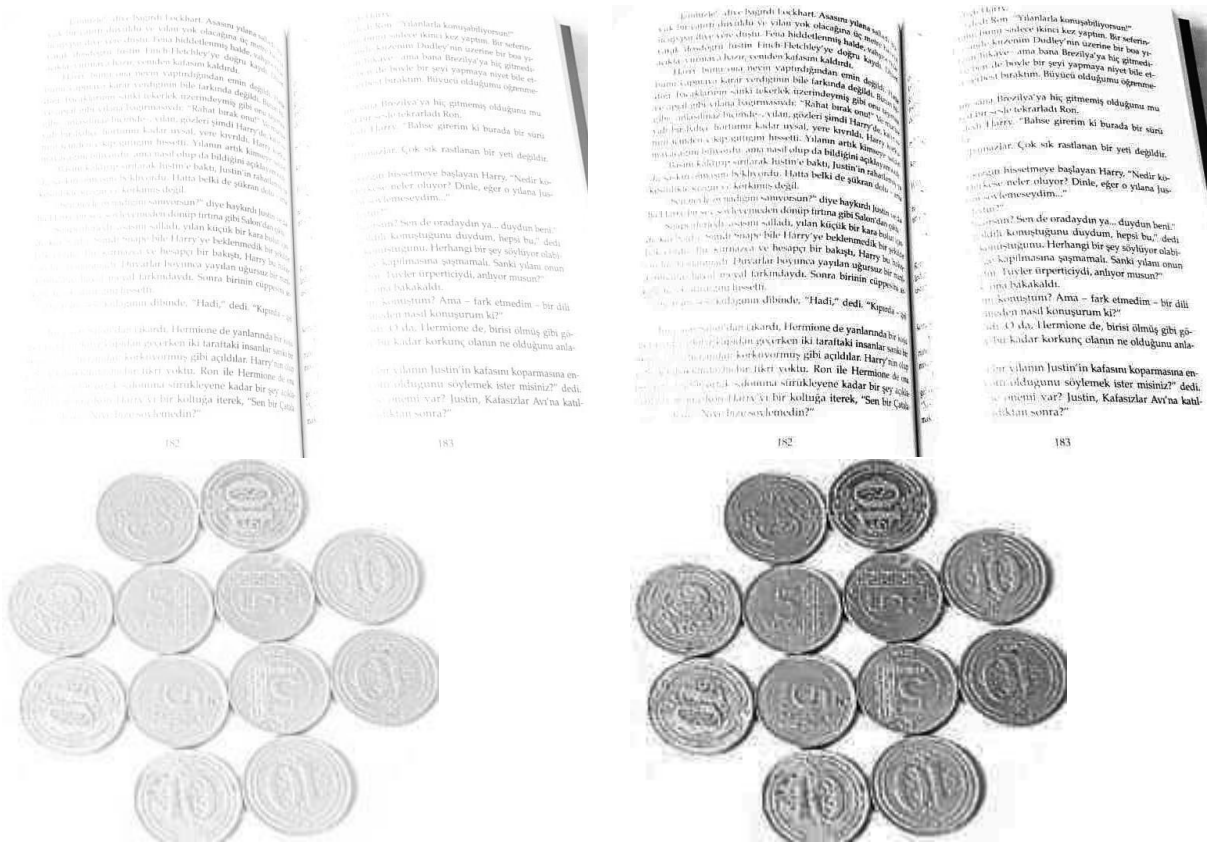


4) Function: gammaTransformation

You can make images lighter or darker with gamma transformation. If the gamma value is greater than 1, you will get a darker image. If the gamma value is less than 1, you will get a lighter image.

```
def gammaTransform(img,gamma):
    c = 255.0/(255.0**gamma)
    img = img.astype(np.float64)
    for (x,y),px in np.ndenumerate(img):
        img[x,y] = (px**gamma)*c
    img = img.astype(np.uint8)
    return img
```

Example:



You see the bright images on the left. I made it darker with the Gamma function. And that way a better view was created. The gamma value is 5 for these images.

The gamma values for these 4 images are 1,3,4 and 5, respectively. The larger the gamma value, the darker the image.

If the gamma value is less than 1, the image will be brighter. But for brighter images, it is more convenient to use log transformation. So I did not show an example for a gamma value less than 1.

5) Gamma correction:

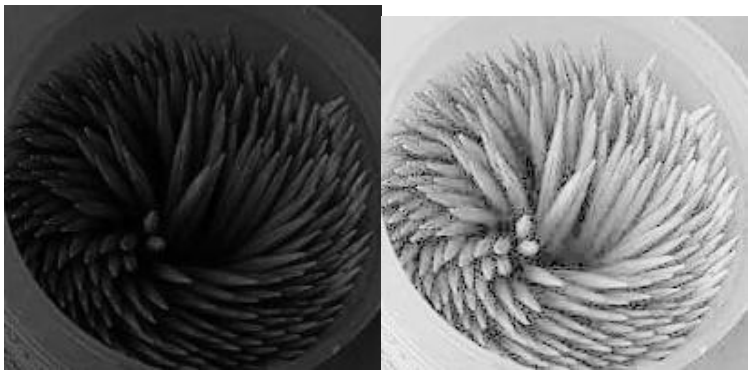
I got the first photo with my gamma correction. I got the second photo with scikit image. The gamma value for both photos is 5. The images of both photos are very close to each other. There are differences only where the lake.

6) Function: logTransformation

The log transformation function is used to get a brighter image.

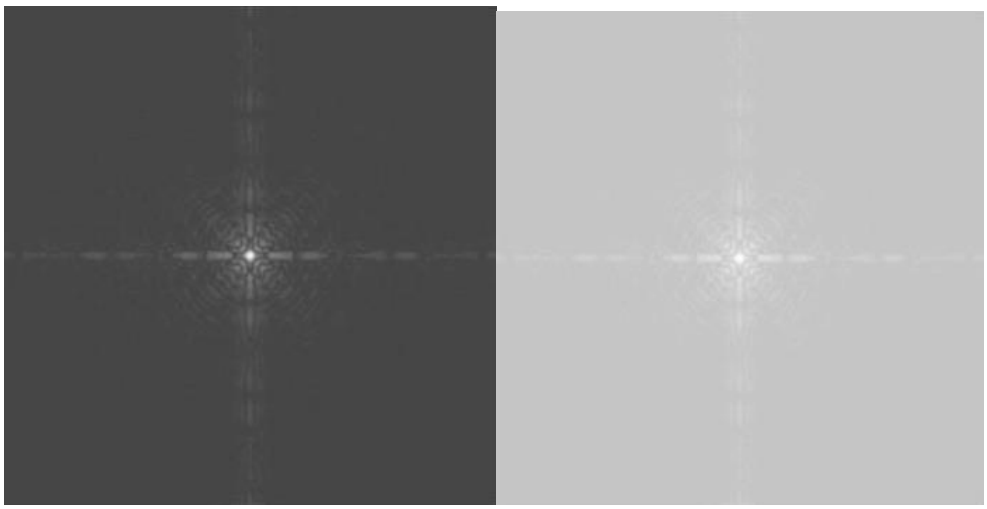
```
def logTransformation(img):
    c = 255 / math.log(1 + np.max(img))

    for (x,y),px in np.ndenumerate(img):
        img[x,y] = int(c * math.log(1 + px))

    return img
```

Example:

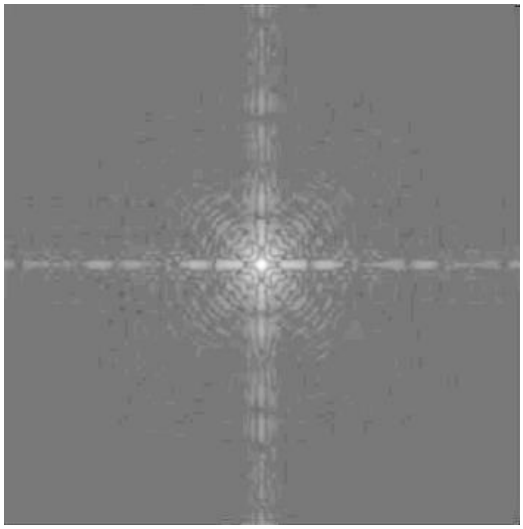


I created a new function with a small change to the log transformation function. In some images, this new function is more effective.

```
def logTransformation2(img):
    minPx = np.min(img)
    img[:,:] -= minPx
    c = 255 / math.log(1 + np.max(img))

    for (x,y),px in np.ndenumerate(img):
        img[x,y] = int(c * math.log(1 + px))

    return img
```



(Gray Image) (created by logTransformation)



(created by logTransformation2)

7) Function: averagePix

This function calculates the average pixel value of the image it takes. and then this average value becomes the new value of the image.

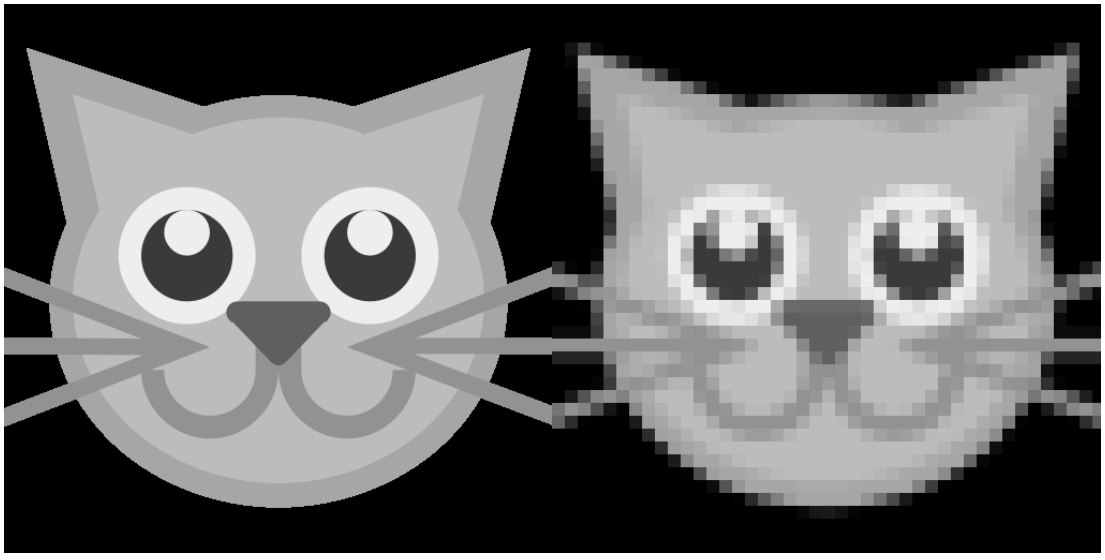
```
def averagePix(img):
    row, col = img.shape
    sumPix = 0

    for (x,y),px in np.ndenumerate(img):
        sumPix += px

    averagePix = int(sumPix/(row*col))
    img[:,:] = [averagePix]

    return img
```

Function: sampling

This function determines certain sized regions on the image. And it calls the averagepix function to calculate the pixel average of these regions. As a result of this process, the image looks like a low resolution image.

```
def sampling(img,delta):
    row, col = img.shape
    divRow = int(row/delta)
    divCol = int(col/delta)

    i=j=0
    while j < divRow:
        while i < divCol:
            img[j*delta:j*delta+delta, i*delta:i*delta+delta] = averagePix(
                img[j*delta:j*delta+delta, i*delta:i*delta+delta])
            i+=1
        i=0
        j+=1

    i=j=0

    if (row - divRow*delta) != 0:
        remeaning = row - divRow*delta
        while i < divCol:
            img[row - remeaning:row, i*delta:i*delta+delta] = averagePix(
                img[row - remeaning:row, i*delta:i*delta+delta])
            i += 1

    if (row - divRow*delta) != 0:
        remeaning = row - divRow*delta
        while j < divRow:
            img[j*delta:j*delta+delta , col - remeaning:col] = averagePix(
                img[j*delta:j*delta+delta , col - remeaning:col])
            j += 1

    if (row - divRow*delta) != 0 and (row - divRow*delta) != 0:
        img[row - remeaning:row , col - remeaning:col] = averagePix(
            img[row - remeaning:row , col - remeaning:col])

    return img
```

Examples:



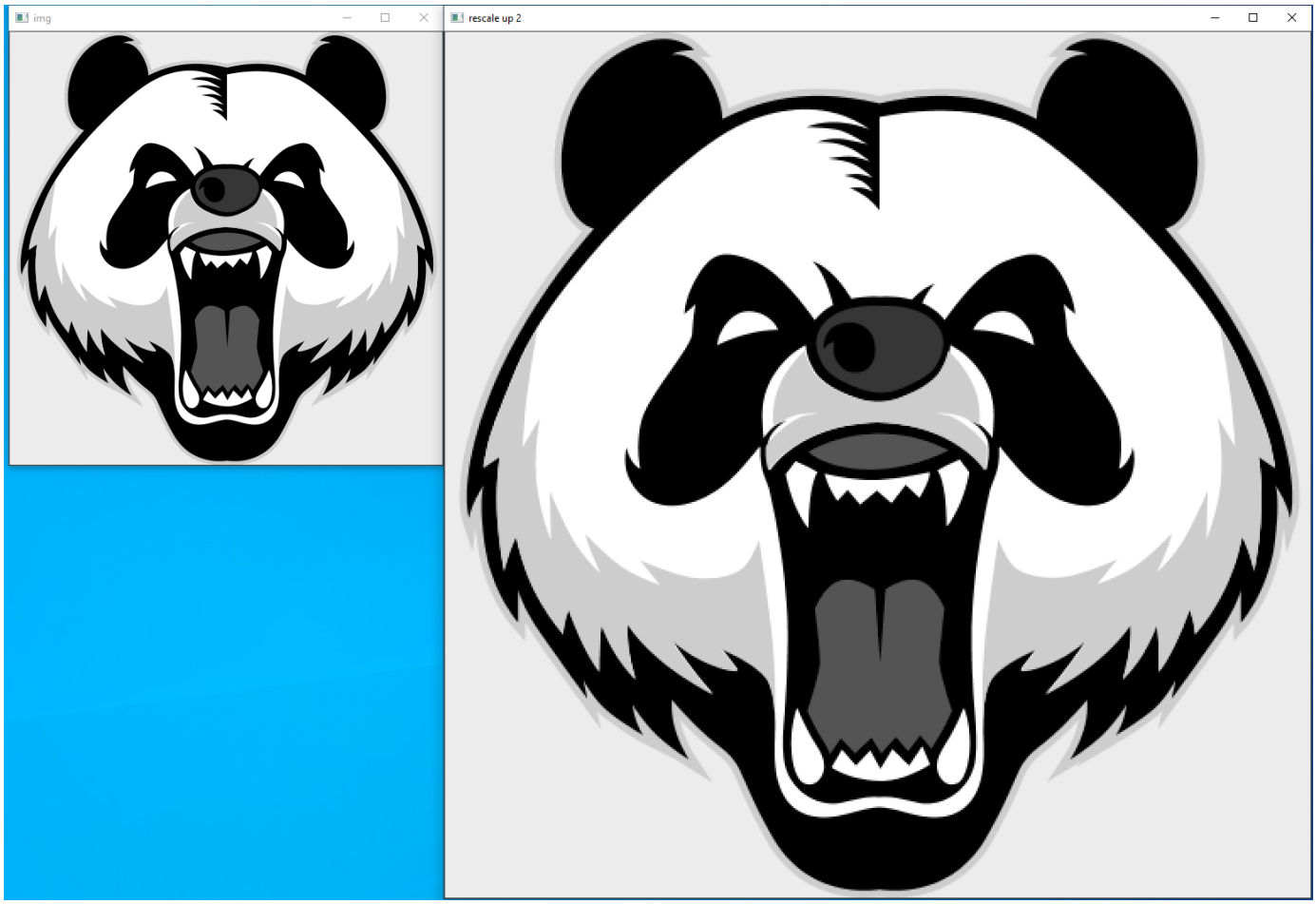
I set the value of the delta as 12 for all images. In the photos on the right, the square regions formed by the delta can be seen comfortably.

Function: upScale

```
def upScale(img,scale):
    row,col = img.shape
    newImg = np.zeros((int(row*scale),int(col*scale)),np.uint8)

    for (x,y),px in np.ndenumerate(img):
        newImg[x*scale:x*scale+scale,y*scale:y*scale+scale] = img[x,y]

    return newImg
```

The upscale function increases the resolution of the image.

Example:

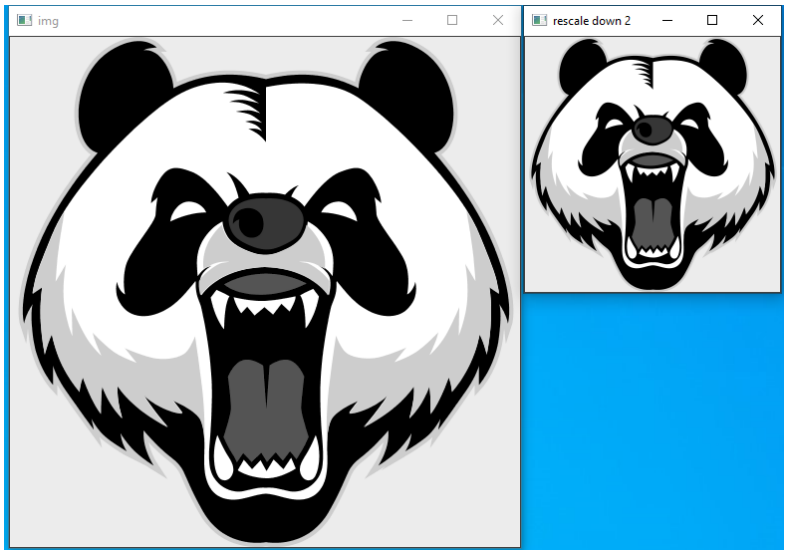


Function: downScale

```
def downScale(img,scale):
    row,col = img.shape
    img = sampling(img,scale)
    newImg = np.zeros((int(row/scale),int(col/scale)),np.uint8)

    for (x,y),px in np.ndenumerate(newImg):
        newImg[x,y] = img[x*scale,y*scale]

    return newImg
```

The downscale function lowers the resolution of the image.

Example:

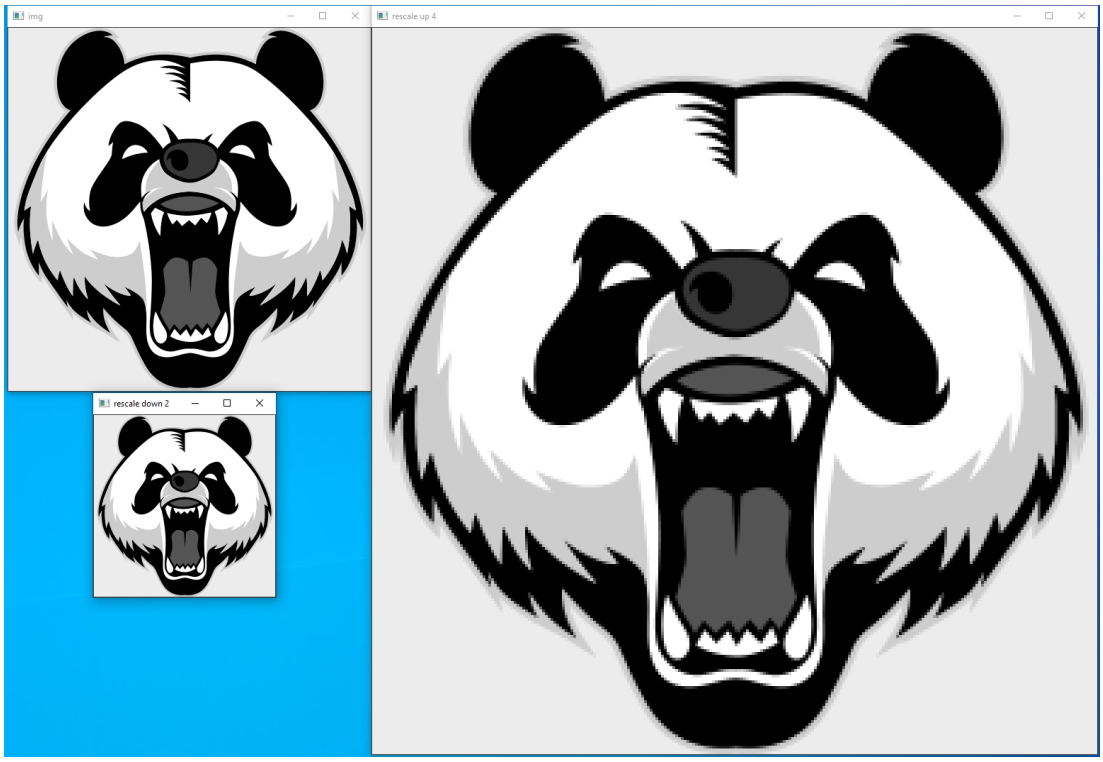


Function: reScale

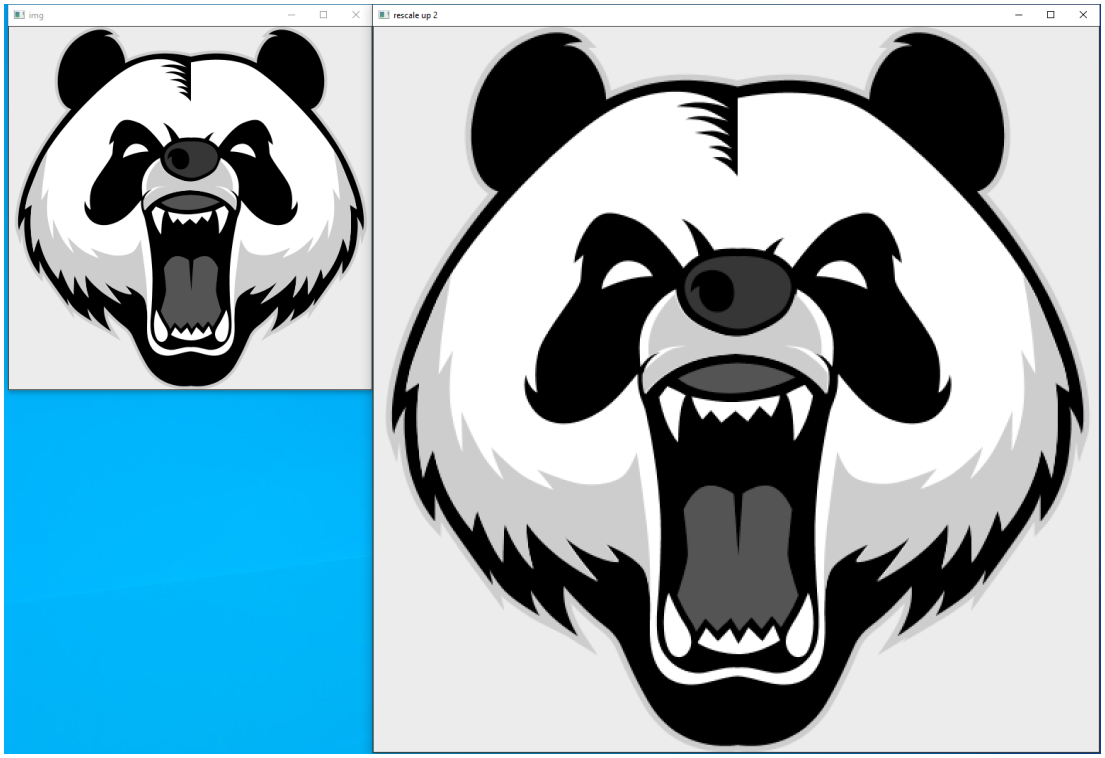
```
def reScale(img,upOrDown,scale):
    if upOrDown == 'up':
        img = upScale(img,scale)
    elif upOrDown == 'down':
        img = downScale(img,scale)

    return img
```

The rescale function is only available for upordown control.

Example:



(Downscale a image by 2, then upscale it by)



(Upscale the image by 2)

Data will be lost if the resolution of the photo is lowered. That's why the first photo looks worse than the other photo.

Explanation about homework: There have been some problems with the jupyter notebook. So I used python's own interface. The codes I wrote here may cause errors in jupyter. But if you use Python's own interface it will not cause any problems. Since the codes take up a lot of space, I added them as a photo above. I am adding the codes to the rest of this place in writing.

Code:

```
def averagePix(img):
    row, col = img.shape
    sumPix = 0

    for (x,y),px in np.ndenumerate(img):
        sumPix += px

    averagePix = int(sumPix/(row*col))
    img[:,:] = [averagePix]

    return img

def sampling(img,delta):
    row, col = img.shape
    divRow = int(row/delta)
    divCol = int(col/delta)

    i=j=0
    while j < divRow:
        while i < divCol:
            img[j*delta:j*delta+delta, i*delta:i*delta+delta] = averagePix(
                img[j*delta:j*delta+delta, i*delta:i*delta+delta])
            i+=1
        i=0
        j+=1
```

```
    i=j=0

    if (row - divRow*delta) != 0:
        remeaning = row - divRow*delta
        while i < divCol:
            img[row - remeaning:row, i*delta:i*delta+delta] = averagePix(
                img[row - remeaning:row, i*delta:i*delta+delta])
            i += 1

    if (row - divRow*delta) != 0:
        remeaning = row - divRow*delta
        while j < divRow:
            img[j*delta:j*delta+delta , col - remeaning:col] = averagePix(
                img[j*delta:j*delta+delta , col - remeaning:col])
            j += 1

    if (row - divRow*delta) != 0 and (row - divRow*delta) != 0:
        img[row - remeaning:row , col - remeaning:col] = averagePix(
            img[row - remeaning:row , col - remeaning:col])

    return img

def upScale(img,scale):
    row,col = img.shape
    newImg = np.zeros((int(row*scale),int(col*scale)),np.uint8)

    for (x,y),px in np.ndenumerate(img):
        newImg[x*scale:x*scale+scale,y*scale:y*scale+scale] = img[x,y]

    return newImg
```

```
def downScale(img,scale):
    row,col = img.shape
    img = sampling(img,scale)
    newImg = np.zeros((int(row/scale),int(col/scale)),np.uint8)

    for (x,y),px in np.ndenumerate(newImg):
        newImg[x,y] = img[x*scale,y*scale]

    return newImg

def reScale(img,upOrDown,scale):
    if upOrDown == 'up':
        img = upScale(img,scale)
    elif upOrDown == 'down':
        img = downScale(img,scale)

    return img

def gammaTransform(img,gamma):
    c = 255.0/(255.0**gamma)
    img = img.astype(np.float64)
    for (x,y),px in np.ndenumerate(img):
        img[x,y] = (px**gamma)*c
    img = img.astype(np.uint8)
    return img

def negativeImg(img):
    img[:,:] = [255] - img[:,:]
    return img
```

```
def logTransformation(img):
    c = 255 / math.log(1 + np.max(img))

    for (x,y),px in np.ndenumerate(img):
        img[x,y] = int(c * math.log(1 + px))

    return img

def logTransformation2(img):
    minPx = np.min(img)
    img[:,:] -= minPx
    c = 255 / math.log(1 + np.max(img))

    for (x,y),px in np.ndenumerate(img):
        img[x,y] = int(c * math.log(1 + px))

    return img

def readImagesFromFile(path):
    images = []
    for image in os.listdir(path):
        img = cv2.imread(os.path.join(path,image))
        if img is not None:
            images.append(img)
    return images

def averageImages(images):
    numberOfImages = len(images)
    row,col,px = images[0].shape
```

```
newImg = np.zeros((row,col,px),np.int64)


for i in range(numberOfImages):
    newImg += images[i]
newImg = newImg / numberOfImages
newImg = newImg.astype(np.uint8)


return newImg
```