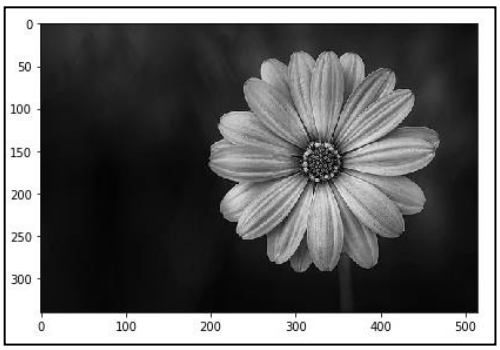
1. **Write a program to read, display and save the image and convert into grayscale using various libraries.**

CODE:

i)Scikit

```
from skimage import color
from skimage import io
img = io.imread('flower1.jpg')
imgGray = color.rgb2gray(img)
io.imshow(imgGray)
```

OUTPUT:



ii) Pillow

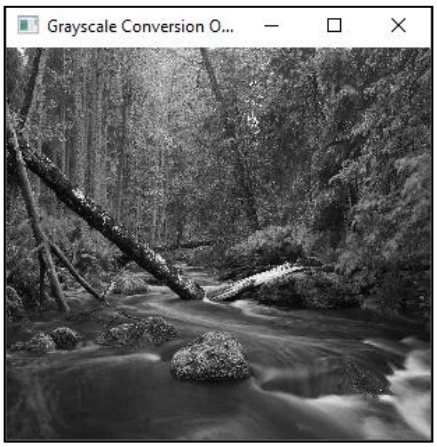
```
from PIL import Image
img = Image.open('flower1.jpg')
imgGray = img.convert('L')
imgGray.show()
```

OUTPUT:

iii)OpenCV

```
import cv2
img_gray=cv2.imread("nature1.jpg",0)
cv2.imshow('Grayscale Conversion OpenCV',img_gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



2. **Develop a program to display:**
   (i) Height and width of the image.
   (ii) Number of channels of image
   (iii) Separate RGB channels
   CODE:

   (i) Height and width of the image.

```
from PIL import Image
filepath='flower1.jpg'
img=Image.open(filepath)
width=img.width
height=img.height
print('width:',width)
print('height:',height)
```

   OUTPUT:

```
width: 514
height: 340
```

   (ii) Number of channels of image

```
import cv2
import numpy
img = cv2.imread("bird3.jpg")
print('No of Channel is: ' + str(img.ndim))
cv2.imshow("Channel", img)
cv2.waitKey()
cv2.destroyAllWindows()
```

   OUTPUT:

```
No of Channel is: 3
```

   (iii) Separate RGB channels

```
import cv2
img=cv2.imread('flower1.jpg')
B,G,R=cv2.split(img)
print(B)
print(G)
print(R)
```

OUTPUT:

```
[[69 69 69 ... 29 30 30]
 [69 69 69 ... 30 30 30]
 [69 69 69 ... 31 32 32]
 ...
 [29 29 28 ... 24 25 25]
 [28 28 28 ... 24 25 25]
 [26 26 26 ... 24 25 25]]
[[24 24 24 ...  7  9  9]
 [24 24 24 ...  8  9  9]
 [24 24 24 ...  9  9  9]
 ...
 [12 12 11 ...  5  7  7]
 [11 11 11 ...  5  7  7]
 [ 9  9  9 ...  5  7  7]]
[[81 81 81 ... 32 34 34]
 [81 81 81 ... 33 34 34]
 [81 81 81 ... 34 37 37]
 ...
 [25 25 24 ... 20 20 20]
 [24 24 24 ... 20 20 20]
 [22 22 22 ... 20 20 20]]
```

```
#blue channel
cv2.imshow("Blue",B)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

```
#green channel
cv2.imshow("Green",G)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:


```
#red channel
cv2.imshow("Red",R)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

3. **Develop a program to resize and rotate the original image.**

CODE:

RESIZE METHOD 1:

```
#resize image using PIL
from PIL import Image
filepath='flower1.jpg'
img=Image.open(filepath)
new_image = img.resize((300, 200))
new_image
```

OUTPUT:

RESIZE METHOD 2:

```
from PIL import Image
filepath='flower1.jpg'
img=Image.open(filepath)
width=25
height=25
new=img.resize((width,height),Image.ANTIALIAS)
new
```

OUTPUT:

ROTATE:

```
from PIL import Image
Original_Image = Image.open("flower1.jpg")
# Rotate Image By 180 Degree
rotated_image1 = Original_Image.rotate(180)
# This is Alternative Syntax To Rotate
# The Image
rotated_image2 = Original_Image.transpose(Image.ROTATE_90)
# This Will Rotate Image By 60 Degree
rotated_image3 = Original_Image.rotate(60)
rotated_image1.show()
rotated_image2.show()
rotated_image3.show()
```

OUTPUT:

4. **Write a program to display matrix representation of an image.**

CODE:

```
#matrix representation of image
import matplotlib.image as image
img=image.imread('flower1.jpg')
print('The Shape of the image is:',img.shape)
print('The image as array is:')
print(img)
```

OUTPUT:

```
The Shape of the image is: (340, 514, 3)
The image as array is:
[[[81 24 69]
  [81 24 69]
  [81 24 69]
  ...
  [32  7 29]
  [34  9 30]
  [34  9 30]]

 [[81 24 69]
  [81 24 69]
  [81 24 69]
  ...
  [33  8 30]
  [34  9 30]
  [34  9 30]]

  [20  7 25]]

 [[22  9 26]
  [22  9 26]
  [22  9 26]
  ...
  [20  5 24]
  [20  7 25]
  [20  7 25]]]
```

5. **Develop a program to convert original image into binary image.**

CODE:

```
import cv2
img = cv2.imread('bird3.jpg',0)
ret, bw_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
cv2.imshow("Binary", bw_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

6. Program to display image attributes.

CODE:

```
from PIL import Image
image = Image.open('flower1.jpg')
print("Filename: ", image.filename)
print("Format: ", image.format)
print("Mode: ", image.mode)
print("Size: ", image.size)
print("Width: ", image.width)
print("Height: ", image.height)
print("Is Animated: ", (getattr(image, "is_animated", False)))
image.close()   # close image file
```

OUTPUT:

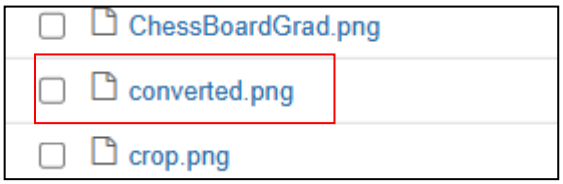
```
Filename:  flower1.jpg
Format:  JPEG
Mode:  RGB
Size: (514, 340)
Width:  514
Height:  340
Is Animated:  False
```

7. **Program to convert an image from one format to other.**

CODE:

```
from PIL import Image
image = Image.open('flower1.jpg')
image.convert('RGB')
image.save("converted.png")
print("Image successfully converted!")
```

OUTPUT:



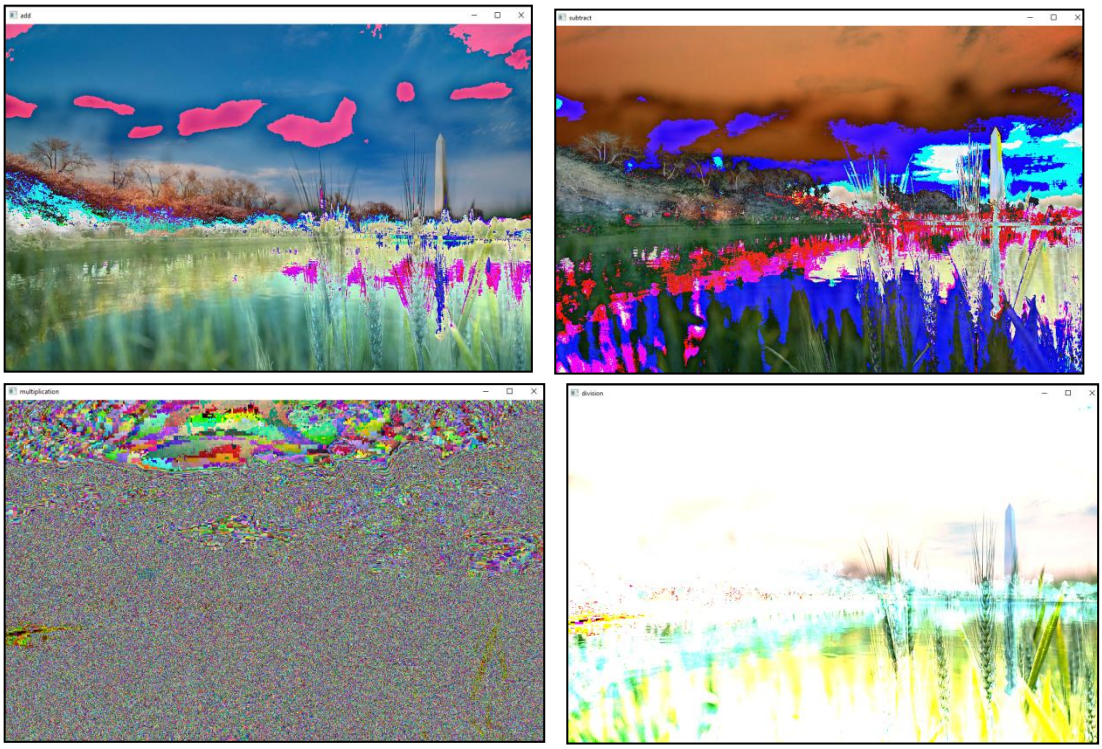
8. **Program to perform arithmetic and logical operations on image.**

CODE:

ARITHMETIC:

```
import cv2
img1=cv2.imread("img.jpeg")
img2=cv2.imread("img4.jpg")
add=img1+img2
sub=img1-img2
mul=img1*img2
div=img1/img2
cv2.imshow("add",add)
cv2.imshow("subtract",sub)
cv2.imshow("multiplication",mul)
cv2.imshow("division",div)
cv2.waitKey(0)
```

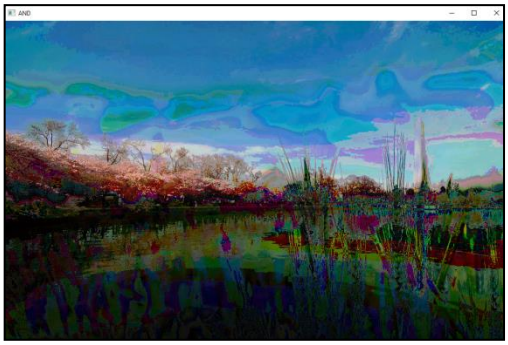
OUTPUT:

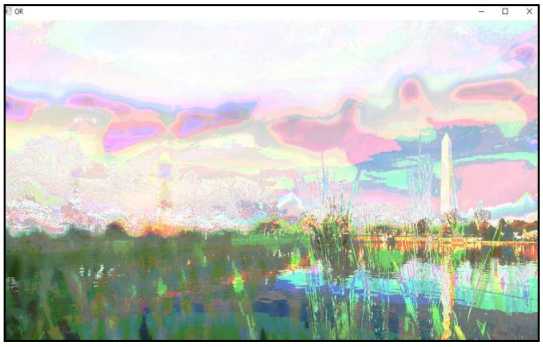


LOGICAL:

```
import cv2
img1=cv2.imread("img.jpeg")
img2=cv2.imread("img4.jpg")
bitwise_AND = cv2.bitwise_and(img1, img2)
bitwise_OR = cv2.bitwise_or(img1, img2)
bitwise_NOT = cv2.bitwise_not(img1)
cv2.imshow('AND',bitwise_AND)
cv2.imshow('OR',bitwise_OR)
cv2.imshow('NOT',bitwise_NOT)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:





9. Program to perform:

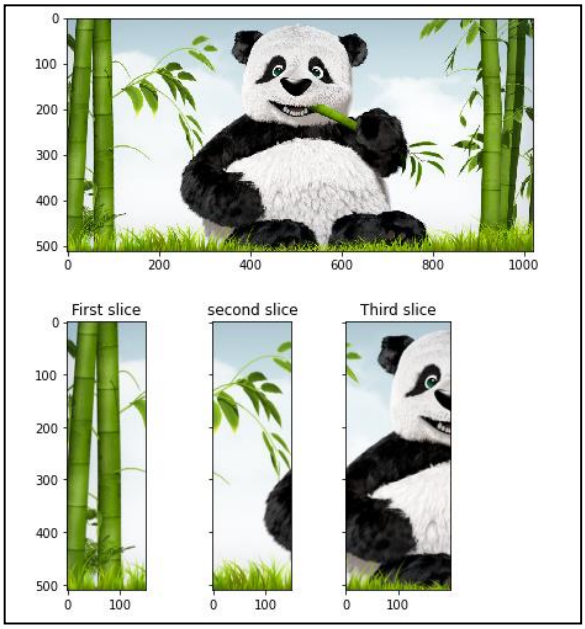
a) **Image slicing**
b) **Blending of images by using mask, filter and blur functions**
c) **Cropping**
d) **Negative of an image**
e) **Drawing on an image**
f) **Writing text on an image**
g) **Finding basic statistics of an image**

a) Image slicing

CODE:

```
from skimage.io import imshow,imread
import matplotlib.pyplot as plt
img1=imread('website.jpg')
imshow(img1)

fig,ax=plt.subplots(1,3,figsize=(6,4), sharey=True)
ax[0].imshow(img1[:, 0:150])
ax[0].set_title('First slice')
ax[1].imshow(img1[:, 150:300])
ax[1].set_title('second slice')
ax[2].imshow(img1[:, 300:500])
ax[2].set_title('Third slice');
```

OUTPUT:



b) Blending of images by using mask, filter and blur functions

CODE:

```
from PIL import Image
img1=Image.open('cat.jfif')
img2=Image.open('tiger.jfif')
alphaBlended=Image.blend(img1,img2,alpha=.4)
alphaBlended.show()
```

OUTPUT:

c) Cropping

CODE:

```
from PIL import Image
im=Image.open('cat.jfif')
w,h=im.size
left=5
top=h/4
right=164
bottom=3*h/4
im1=im.crop((left,top,right,bottom))
im1.show()
```

OUTPUT:

d) Negative of an image

CODE:

```
import cv2
import numpy as np
img=cv2.imread('tiger.jfif')
print(img.dtype)
img_neg=255-img
cv2.imshow('negative',img_neg)
cv2.waitKey(0)
```

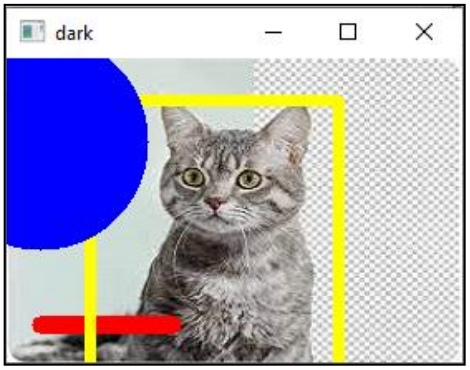
OUTPUT:

e) Drawing on an image

CODE:

```
import numpy as np
import cv2
img =cv2.imread('cat.jfif')
cv2.line(img, (20, 160), (100, 160), (0, 0, 255), 10)
cv2.rectangle(img,(50,25), (200,300),(0,255,255),5)
cv2.circle(img, (20,50), 65, (255,0,0), -1)
cv2.imshow('dark', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



f) Writing text on an image

CODE:

```
from PIL import Image,ImageDraw,ImageFont

img = Image.open('moon.jpg')
I1 = ImageDraw.Draw(img)
fnt=ImageFont.truetype('arial.ttf', 50)
I1.text((28, 36), "Image processing", font=fnt, fill=(255, 0, 0))
img.show()
img.save("moon1.png")
```

OUTPUT:

g) Finding basic statistics of an image

CODE:

```
#finding basic statistics of an image - mean
from PIL import Image, ImageStat

im = Image.open('moon.jpg')
stat = ImageStat.Stat(im)
print(stat.mean)
```

OUTPUT:

```
[55.20498333333333, 55.32209375, 55.40808125]
```

```
#finding basic statistics of an image - median
from PIL import Image, ImageStat
im = Image.open('website.jpg')
stat = ImageStat.Stat(im)
print(stat.median)
```

OUTPUT:

```
[188, 206, 211]
```

```
#finding basic statistics of an image - standard deviation
from PIL import Image, ImageStat
im = Image.open('moon.jpg')
stat = ImageStat.Stat(im)
print(stat.stddev)
```

OUTPUT:

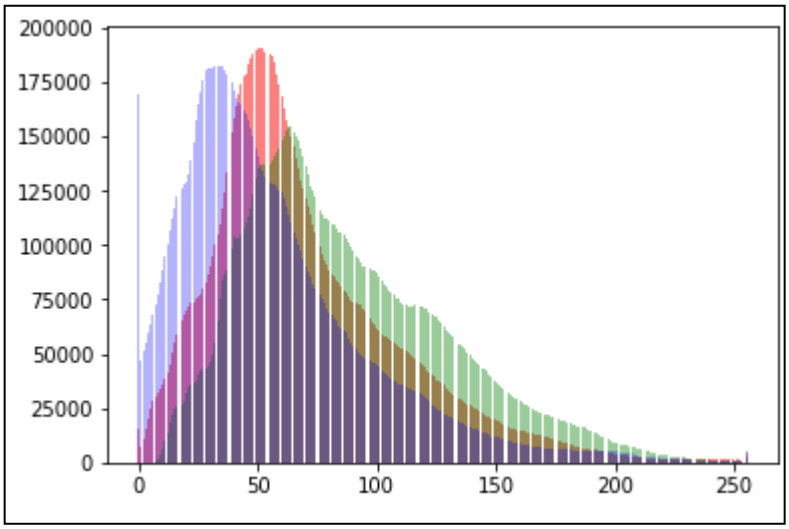
```
[80.99912542326032, 81.12070835715262, 81.08885122265616]
```

10. Program to form histogram of an image.

CODE:

```
from PIL import Image
import matplotlib.pyplot as plt
im = Image.open("nature1.jpg")
pl = im.histogram()
plt.bar(range(256), pl[:256], color='r', alpha=0.5)
plt.bar(range(256), pl[256:2*256], color='g', alpha=0.4)
plt.bar(range(256), pl[2*256:], color='b', alpha=0.3)
plt.show()
```

OUTPUT:

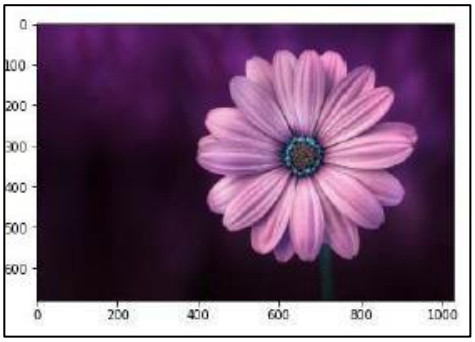


11. Develop a program to perform:
i) Sampling of an image (up and down sampling)
ii) Median filtering
iii) Average filtering
iv) Interpolation
v) Quantization
CODE:

i) Sampling of an image (up and down sampling)
UPSAMPLING:

```
import cv2
from matplotlib import pyplot as plt
image = cv2.imread('flower1.jpg')
cv2.imshow("image before pyrUp: ",image)

image1 = cv2.pyrUp(image)
cv2.imshow('UpSample', image1)
plt.imshow(image1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



DOWN SAMPLING:

```
import cv2
from matplotlib import pyplot as plt
image = cv2.imread('flower1.jpg')
cv2.imshow("image before pyrDown: ",image)

image1 = cv2.pyrDown(image)
cv2.imshow('DownSample', image1)
plt.imshow(image1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

ii) Median filtering

```
import cv2
import numpy as np
img_noisy1=cv2.imread("noisy2.png",0)
m,n=img_noisy1.shape
img_new1=np.zeros([m,n])
for i in range(1,m-1):
  for j in range(1,n-1):
       temp=[img_noisy1[i-1,j-1],
           img_noisy1[i-1,j],
           img_noisy1[i-1,j+1],
           img_noisy1[i,j-1],
           img_noisy1[i,j],
```

```
            img_noisy1[i,j+1],
            img_noisy1[i+1,j-1],
            img_noisy1[i+1,j],
            img_noisy1[i+1,j+1]]
        temp=sorted(temp)
        img_new1[i,j]=temp[4]
img_new1=img_new1.astype(np.uint8)

cv2.imshow('median filtered image',img_new1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

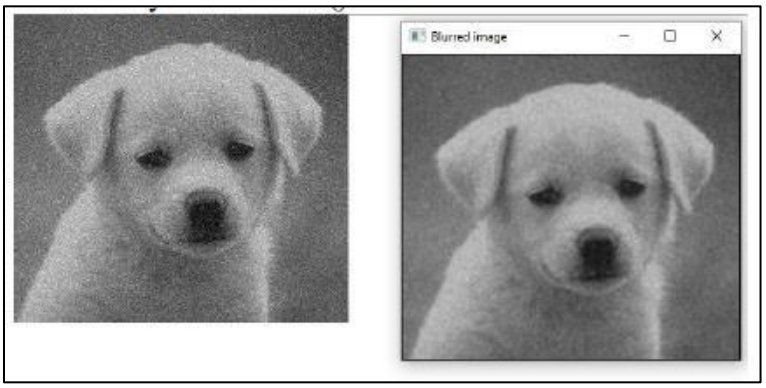
OUTPUT:

iii) Average filtering

```
import cv2
import numpy as np
img=cv2.imread("noisy.png",0)
m,n=img.shape
mask=np.ones([3,3],dtype=int)
mask=mask/9
img_new=np.zeros([m,n])
for i in range(1,m-1):
  for j in range(1,n-1):
    temp=img[i-1,j-1]*mask[0,0]+img[i-1,j]*mask[0,1]+img[i-
1,j+1]*mask[0,2]+img[i,j-
1]*mask[1,0]+img[i,j]*mask[1,1]+img[i,j+1]*mask[1,2]+img[i+1,j-
```

```
1]*mask[2,0]+img[i+1,j]*mask[2,1]+img[i+1,j+1]*mask[2,2]
img_new[i,j]=temp
img_new=img_new.astype(np.uint8)
cv2.imshow('Blurred image',img_new)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



iv) Interpolation

```
importcv2
import numpy as np
image=cv2.imread('night.jpg')
nearest=cv2.resize(image,None,fx=25,fy=25,interpolation=cv2.INTER_NEAREST)
cv2.imshow('Nearest',nearest)
linear=cv2.resize(image,None,fx=5,fy=5,interpolation=cv2.INTER_LINEAR)
cv2.imshow('LINEAR',linear)
bicubic=cv2.resize(image,None,fx=5,fy=5,interpolation=cv2.INTER_CUBIC)
cv2.imshow('BICUBIC',bicubic)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

v) Quantization

```
import cv2
from PIL import Image
image=Image.open('bird3
.jpg')
img=image.quantize(19)
img.show()
```

OUTPUT:

12. Write a program to perform basic image data analysis using intensity transformation:

a) Image negative b) Log transformation c) Gamma correction

a) Image negative

CODE:

```
!pip install imageio
%matplotlib inline
import imageio
import matplotlib.pyplot as plt
import warnings
import matplotlib.cbook
warnings.filterwarnings("ignore",category=matplotlib.cbook.mplDeprecatio
n)
pic=imageio.imread('flower1.jpg')
plt.figure(figsize=(6,6))
negative=255-pic
plt.figure(figsize=(6,6))
plt.imshow(negative)
plt.axis('off')
```

OUTPUT:

b) Log transformation

CODE:

```
%matplotlib inline
import imageio
import numpy as np
import matplotlib.pyplot as plt
pic=imageio.imread('flower1.jpg')
```

```
gray=lambda rgb:np.dot(rgb[...,:3],[0.299,0.587,0.114])
gray=gray(pic)
max_=np.max(gray)
def log_transform():
return(255/np.log(1+max_))*np.log(1+gray)
plt.figure(figsize=(5,5))
plt.imshow(log_transform(),cmap=plt.get_cmap(name='gray'))
plt.axis('off')
```

OUTPUT:

c) Gamma correction

CODE:

```
import imageio
import matplotlib.pyplot as plt
pic=imageio.imread('flower1.jpg')
gamma=2.2
gamma_correction=((pic/255)**(1/gamma))
plt.figure(figsize=(5,5))
plt.imshow(gamma_correction)
plt.axis('off')
```

OUTPUT:

13. Write a program to perform basic image manipulation:

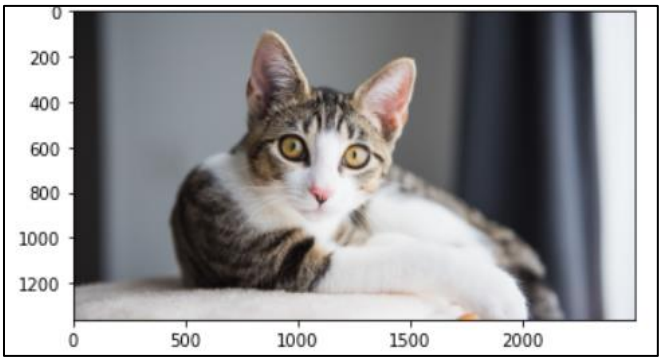
a) Sharpness b) Flipping c) Cropping

a) Sharpness

CODE:

```
from PIL import Image
from PIL import ImageFilter
import matplotlib.pyplot as plt
my_image=Image.open('kit.jpg')
sharp=my_image.filter(ImageFilter.SHARPEN)
sharp.save('sharpen.jpg')
sharp.show()
plt.imshow(sharp)
plt.show()
```

OUTPUT:

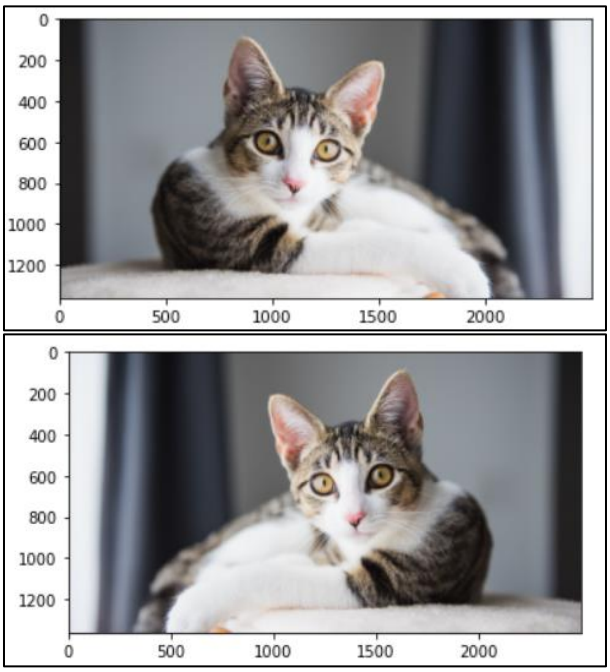


b) Flipping

CODE:

```
import matplotlib.pyplot as plt
img=Image.open('kit.jpg')
plt.imshow(img)
plt.show()
flip=img.transpose(Image.FLIP_LEFT_RIGHT)
flip.save('flip.jpg')
plt.imshow(flip)
plt.show()
```

OUTPUT:

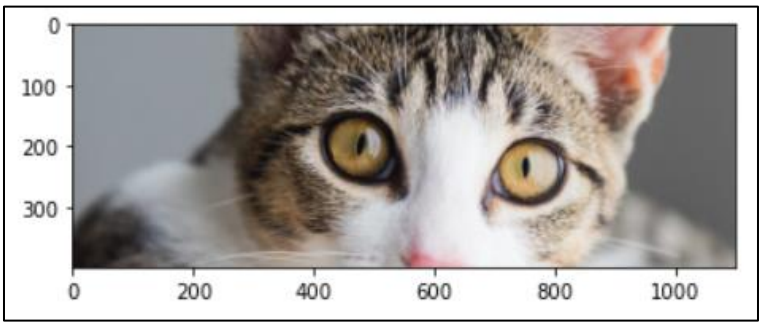


c) Cropping

CODE:

```
from PIL import Image
import matplotlib.pyplot as plt
im=Image.open('kit.jpg')
width,height=im.size
im1=im.crop((500,400,1600,800))
im1.show()
plt.imshow(im1)
plt.show()
```

OUTPUT:



14. Program to perform:

i) Image restoration:

a) Restore a damaged image b) Removing Logo's

ii) Noise:

a) Adding noise b) Reducing Noise c) Reducing Noise while preserving edges

iii) Segmentation

a) Super pixel Segmentation

iv) Contours:

a) Contouring shapes b) Find contours of an image that is not binary c) Count the dots in a dice's image

i) Image restoration:

a) Restore a damaged image

CODE:

```
import numpy as np
import cv2
img = cv2.imread('cat_damaged.png')
mask = cv2.imread('cat_mask.png', 0)
dst = cv2.inpaint(img, mask, 3, cv2.INPAINT_NS)
cv2.imwrite('cat_inpainted.png', dst)
```

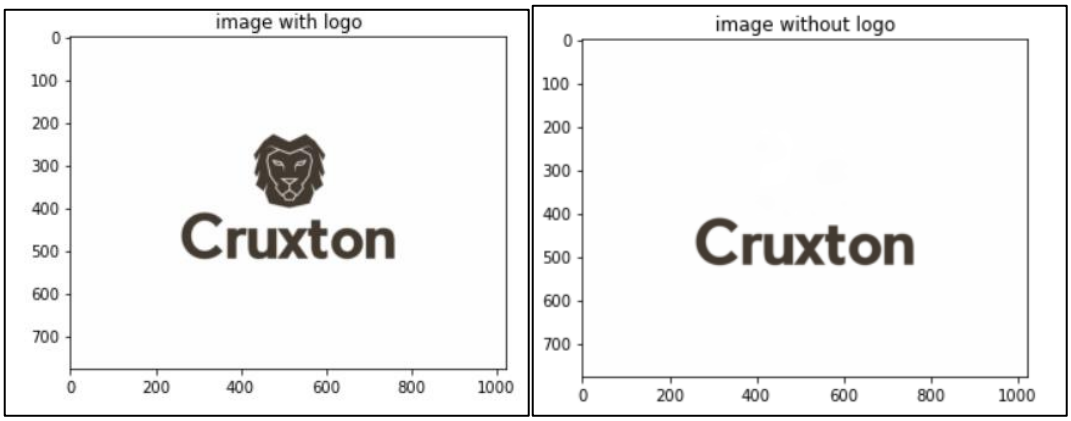
OUTPUT:

b) Removing Logo's

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from skimage.restoration import inpaint
from skimage.transform import resize
from skimage import color
image_with_logo=plt.imread('lg.png')
mask=np.zeros(image_with_logo.shape[:-1])
mask[200:410,400:610]=1
image_logo_removed=inpaint.inpaint_biharmonic(image_with_logo,
mask,multichannel=True)
plt.title('image with logo')
plt.imshow(image_with_logo)
plt.show()
plt.title('image without logo')
plt.imshow(image_logo_removed)
plt.show()
```

OUTPUT:



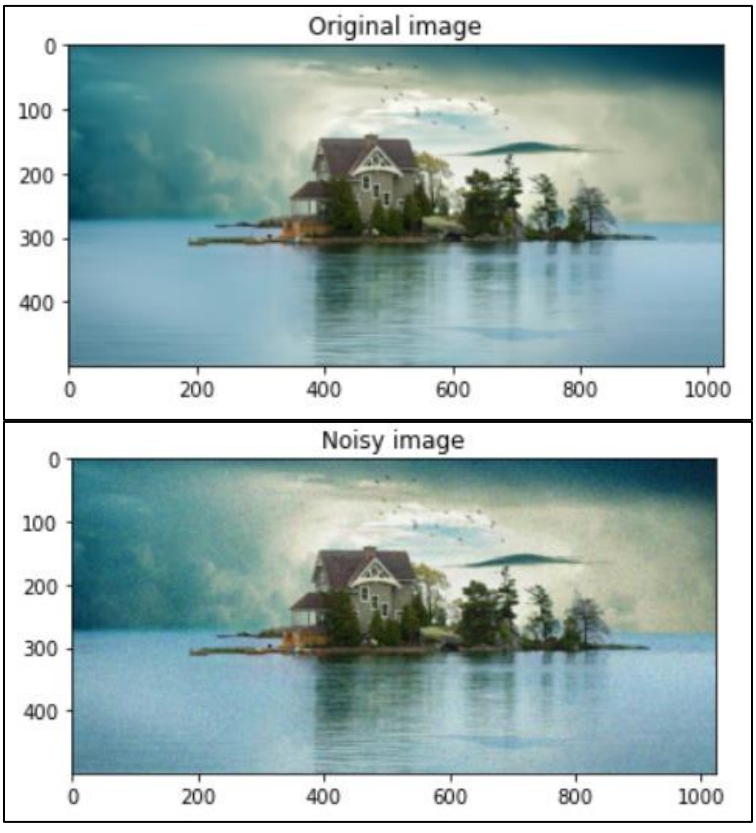
ii) Noise:

a) Adding noise

CODE:

```
import matplotlib.pyplot as plt
from skimage.util import random_noise

nature_image=plt.imread('b.jpg')
noisy_image=random_noise(fruit_image)
plt.title('Original image')
plt.imshow(nature_image)
plt.show()
plt.title('Noisy image')
plt.imshow(noisy_image)
plt.show()
```

OUTPUT:

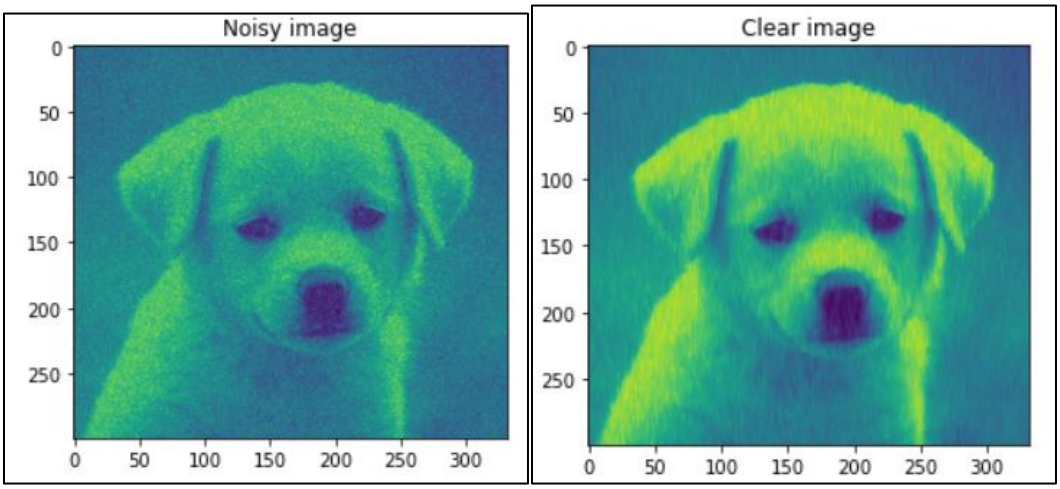


b) Reducing Noise

CODE:

```
import matplotlib.pyplot as plt
from skimage.restoration import denoise_tv_chambolle
noisy_image=plt.imread('n.png')
denoised_image=denoise_tv_chambolle(noisy_image,multichannel=True)
plt.title('Noisy image')
plt.imshow(noisy_image)
plt.show()
plt.title('Clear image')
plt.imshow(denoised_image)
plt.show()
```

OUTPUT:



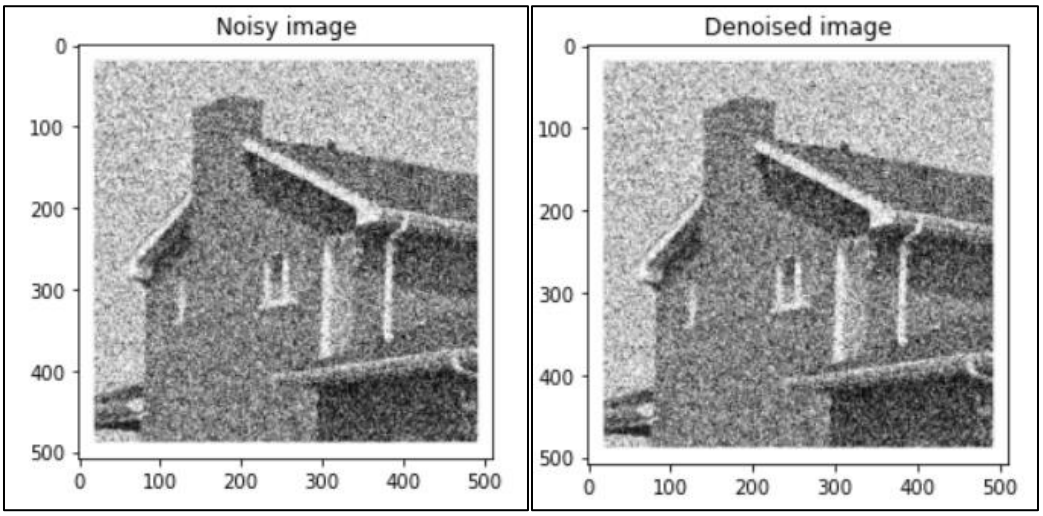
c) Reducing Noise while preserving edges

CODE:

```
import matplotlib.pyplot as plt
from skimage.restoration import denoise_bilateral
landscape_image=plt.imread('noise.jpg')
denoised_image=denoise_bilateral(landscape_image,multichannel=True)
plt.title('Noisy image')
plt.imshow(landscape_image)
```

```
plt.show()
plt.title('Denoised image')
plt.imshow(denoised_image)
plt.show()
```

OUTPUT:



iii) Segmentation:

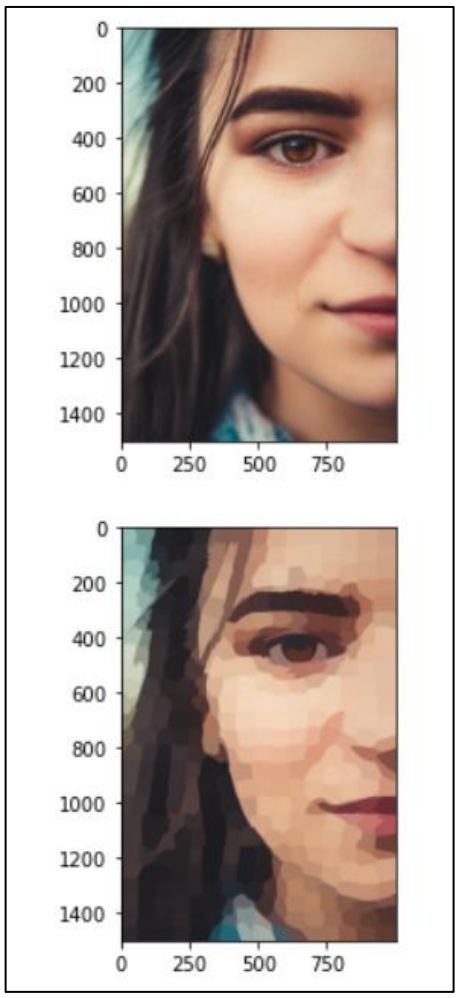
a) Super pixel Segmentation

CODE:

```
from skimage.segmentation import slice

from skimage.color import label2rgb

import matplotlib.pyplot as plt
img=plt.imread('face.jpg')

#obtain the segmentation with 400 regions
segments=slic(img,n_segments=400, compactness=20)

#put segments on top of original image to compare
segmented_image=label2rgb(segments,img,kind='avg')
```

```
#Show the segmented image
plt.imshow(img.astype('uint8'))
plt.show()
plt.imshow(segmented_image.astype('uint8'))
plt.show()
```

OUTPUT:



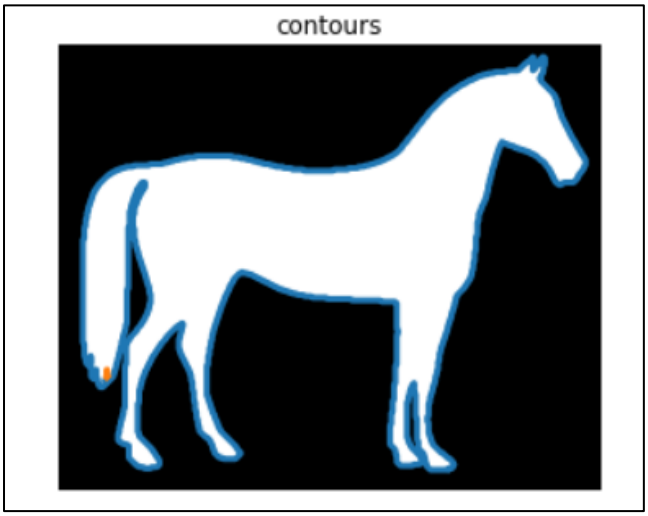
iv) Contours:

a) Contouring shapes

CODE:

```
def show_image_contour(image,contours):
    plt.figure()
    for n,contour in enumerate(contours):
            plt.plot(contour[:,1],contour[:,0],linewidth=3)
    plt.imshow(image,interpolation='nearest',cmap='gray_r')
    plt.title('contours')
    plt.axis('off')

from skimage import measure,data
import matplotlib.pyplot as plt

img=data.horse()

contours=measure.find_contours(img,level=0.8)
show_image_contour(img,contours)
```

OUTPUT:



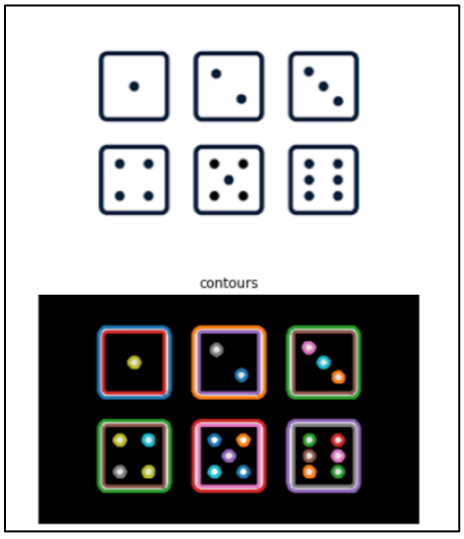
b) Find contours of an image that is not binary

CODE:

```
def show_image_contour(image,contours):
        plt.figure()
```

```
        for n,contour in enumerate(contours):
            plt.plot(contour[:,1],contour[:,0],linewidth=3)
        plt.imshow(image,interpolation='nearest',cmap='gray_r')
        plt.title('contours')
        plt.axis('off')

from skimage import measure,data
import matplotlib.pyplot as plt

#Find contours of an image that is not binary
from skimage.io import imread
from skimage.filters import threshold_otsu
from skimage import color

image_dices=plt.imread('dice.png')
plt.axis('off')
plt.imshow(image_dices)
image_dices=color.rgb2gray(image_dices)
thresh=threshold_otsu(image_dices)
binary=image_dices>thresh
contours=measure.find_contours(binary,level=0.8)
show_image_contour(image_dices,contours)
```

OUTPUT:



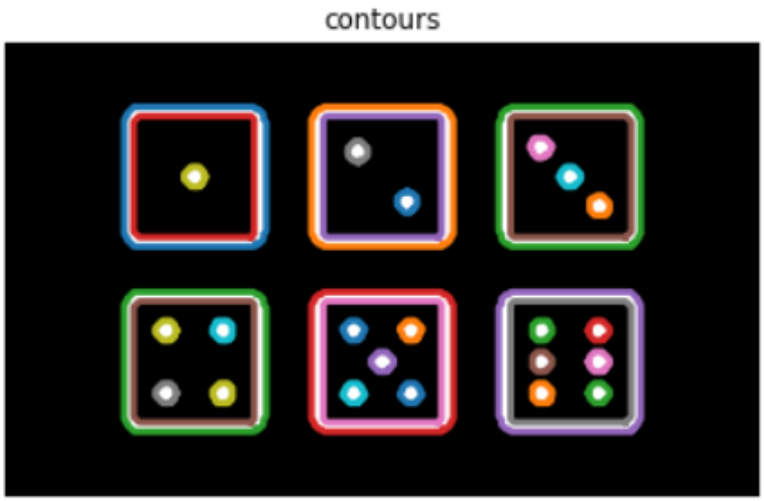
c)Count the dots in a dice's image

CODE:

```
import numpy as np
shape_contours=[cnt.shape[0] for cnt in contours]
max_dots_shape=50
dots_contours=[cnt for cnt in contours if np.shape(cnt)[0]<max_dots_shape]
show_image_contour(binary,contours)
print('Dice;s dots number:{}.'.format(len(dots_contours)))
```

OUTPUT:

```
Dice;s dots number:21.
```

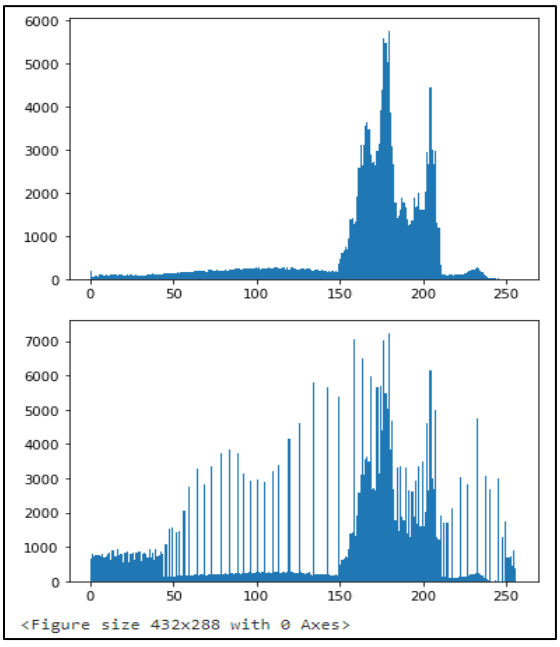


15.Write a program to perform histogram equalization of an image.

CODE:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img=cv2.imread('king.jpg',0)
plt.hist(img.ravel(),256,[0,256])
plt.show()
plt.savefig('hist.png')

equ=cv2.equalizeHist(img)
res=np.hstack((img,equ))
cv2.imshow('Equalized Image',res)
cv2.imwrite('Equalized Image.png',res)

plt.hist(res.ravel(),256,[0,256])
plt.show()
plt.savefig('equal-hist.png')
```

OUTPUT:



16.Develop a program to iterate through the folders and read all the image files and display its name respectively.

CODE:

```
#Listing images that ends with .png
import os
from os import listdir

# get the path/directory
folder_dir= "D:/images"
for images in os.listdir(folder_dir):

# check if the image ends with png
if (images.endswith(".png")):
        print(images)
```

OUTPUT:

```
bot.png
bot1.png
bottle.png
box1.png
catdamaged.png
catmask.png
filter.png
img1.png
img2.png
img3.png
s3.png
text.png
text1.png
watermark.png
wt.png
```

```
#Listing all the images from the directory
import os
from os import listdir

# get the path or directory
folder_dir="D:/images"
for images in os.listdir(folder_dir):
  print(images)
```

OUTPUT:

```
1img.jpg
2img.jpg
bot.png
bot1.png
bottle.png
box.jpg
box1.png
catdamaged.png
catmask.png
damaged img2.jpg
filter.png
i1.jpg
i10.jpg
i2.jpeg
i3.jpg
i4.jfif
i7.jpg
i9.jpg
img1.png
img2.png
img3.png
s1.jpg
s3.png
Sandipan Dey - Hands-On Image Processing with Python-Packt Publishing (2018).pdf
text.png
text1.png
text2.jpg
watermark.png
wt.png
```

17. Develop a program to iterate through the folders and read all image file, apply image transformation.[Rotate, resize] and display the resultant images.

CODE:

```
from PIL import Image import os
os.getcwd()
```

OUTPUT:

'C:\\Users\\User\\image processing'

```
os.listdir()
```

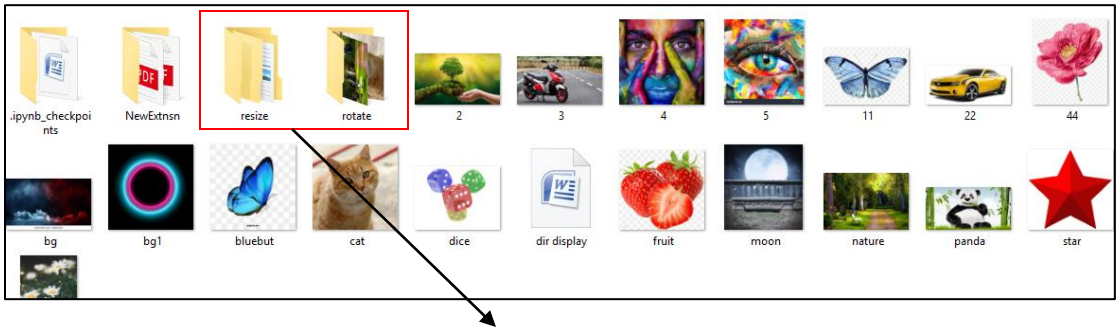
OUTPUT:

```
['.ipynb_checkpoints',
 'alpha blending.ipynb',
 'average filtering.ipynb',
 'back.jpg',
 'biharmonic.png',
 'bird3.jpg',
 'bitwise.ipynb',
 'canvas.ipynb',
 'cat_damaged.png',
 'cat_inpainted.png',
 'cat_mask.png',
 'color spaces.ipynb',
 'contours.ipynb',
 'converted.png',
 'crop.png',
 'cropping.ipynb',
 'damaged_astronaut.png',
 'dice.jpg',
 'dice2.jpg',
```

```
# Creating new Directory using OS library
os.mkdir('NewExtnsn')

for f in os.listdir("."):
    if f.endswith(".jpg"):
        i = Image.open(f)
        fn, fext = os.path.splitext(f)
        i.save("NewExtnsn/{}.pdf".format(fn))
```

```
# Creating new multiple Directories using OS library
os.makedirs('resize//small')
os.makedirs('resize//tiny')

size_small = (600,600) # small images of 600 X 600 pixels
size_tiny = (200,200)  # tiny images of 200 X 200 pixels
for f in os.listdir("."):
    if f.endswith(".jpg"):
        i = Image.open(f)
        fn, fext = os.path.splitext(f)
        i.thumbnail(size_small)
        i.save("resize/small/{}_small{}".format(fn, fext))
        i.thumbnail(size_tiny)
        i.save("resize/tiny/{}_tiny{}".format(fn, fext))

# Creating new Directory using OS library
os.mkdir('rotate')

for f in os.listdir("."):
    if f.endswith(".jpg"):
        i = Image.open(f)
        fn, fext = os.path.splitext(f)
        im = i.rotate(90)
        im.save("rotate/{}_rot.{}".format(fn, fext))
```

OUTPUT:



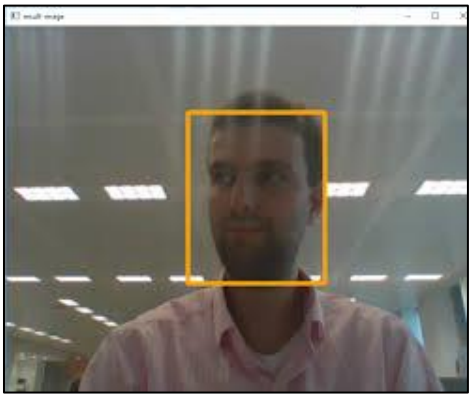
Newly Created Folders(Rotate and Resize)

18. Develop a program to detect the face by using webcam.

CODE:

```
import cv2
cam = cv2.VideoCapture(0)
cv2.namedWindow("test")
img_counter = 0

while True:
    ret, frame = cam.read()
    if not ret:
        print("failed to grab frame")
        break
    cv2.imshow("test", frame)
    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        img_name = "opencv_frame_{}.png".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

cam.release()
cv2.destroyAllWindows()
```

OUTPUT:



19. Develop a program to capture the photo when spacebar is clicked.

CODE:

```
import cv2

cam = cv2.VideoCapture('video.mp4')
cv2.namedWindow("test")
img_counter = 0

while True:
    ret, frame = cam.read()
    if not ret:
        print("failed to grab frame")
        break
    cv2.imshow("test", frame)
    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        img_name = "opencv_frame_{}.png".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

cam.release()

cv2.destroyAllWindows()
```

OUTPUT:

```
opencv_frame_0.png written!
opencv_frame_1.png written!
opencv_frame_2.png written!
opencv_frame_3.png written!
opencv_frame_4.png written!
opencv_frame_5.png written!
opencv_frame_6.png written!
opencv_frame_7.png written!
opencv_frame_8.png written!
opencv_frame_9.png written!
opencv_frame_10.png written!
opencv_frame_11.png written!
opencv_frame_12.png written!
opencv_frame_13.png written!
opencv_frame_14.png written!
failed to grab frame
```

20. Develop a program to perform Haar Cascade to detect face and eye from images.

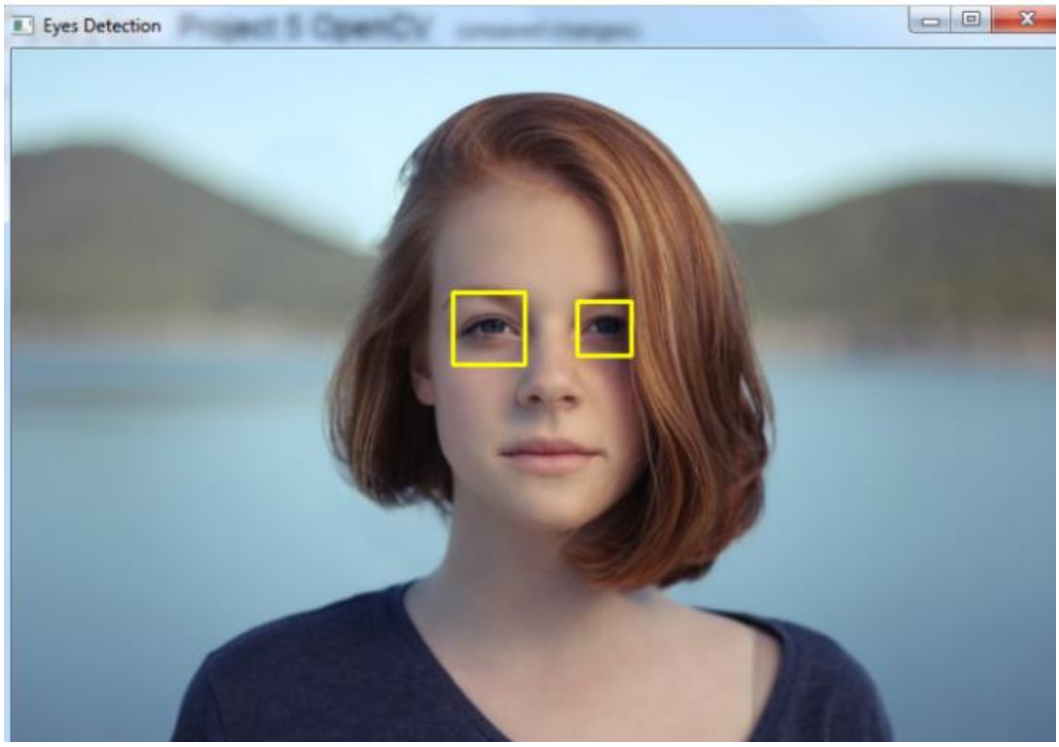
CODE:

```
# eyes detection
import cv2

# read input image
img=cv2.imread('faces.jpg')

# convert to grayscale of each frames
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
# read the haarcascade to detect the faces in an image
face_cascade=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# read the haarcascade to detect the eyes in an image
eye_cascade=cv2.CascadeClassifier('haarcascade_eye_tree_eyeglasses.xml')

# detects faces in the input image
faces=face_cascade.detectMultiScale(gray,1.3,4)
print('Number of detected faces:',len(faces))

# loop over the detected faces
for (x,y,w,h) in faces:
roi_gray=gray[y:y+h,x:x+w]
roi_color=img[y:y+h,x:x+w]

# detects eyes of within the detected face area (roi)
eyes=eye_cascade.detectMultiScale(roi_gray)

# draw a rectangle around eyes
for (ex,ey,ew,eh) in eyes:
cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,255),2)

# display the image with detected eyes
cv2.imshow('Eyes Detection',img)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

Number of detected faces: 1



21.Write a program to develop montage of images.

CODE:

```
import skimage.io
import skimage.util

a=skimage.io.imread('image1.jpg')
print(a.shape)

b=a//2
c=a//3
d=a//4
m=skimage.util.montage([a,b,c,d],multichannel=True)
print(m.shape)

skimage.io.imsave('skimage_montage_default.jpg',m)
```

OUTPUT:

22.Develop a program to perform various edge detection programs.

CODE:

```
#canny edge detection
import cv2
import numpyas np
import matplotlib.pyplot as plt
plt.style.use('seaborn')

image1=cv2.imread("flower.jpg")
image1=cv2.cvtColor(image1,cv2.COLOR_BGR2RGB)

gray_image=cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
edged_image=cv2.Canny(gray_image,threshold1=20,threshold2=100)

plt.figure(figsize=(20,20))
plt.subplot(1,3,1)
plt.imshow(image1,cmap='gray')
plt.title('original image')
plt.axis('off')

plt.subplot(1,3,2)
plt.imshow(gray_image,cmap='gray')
plt.title('grayscale image')
plt.axis('off')

plt.subplot(1,3,3)
plt.imshow(edged_image,cmap='gray')
plt.title('Canny edge detected image')
plt.axis('off')
plt.show()
```

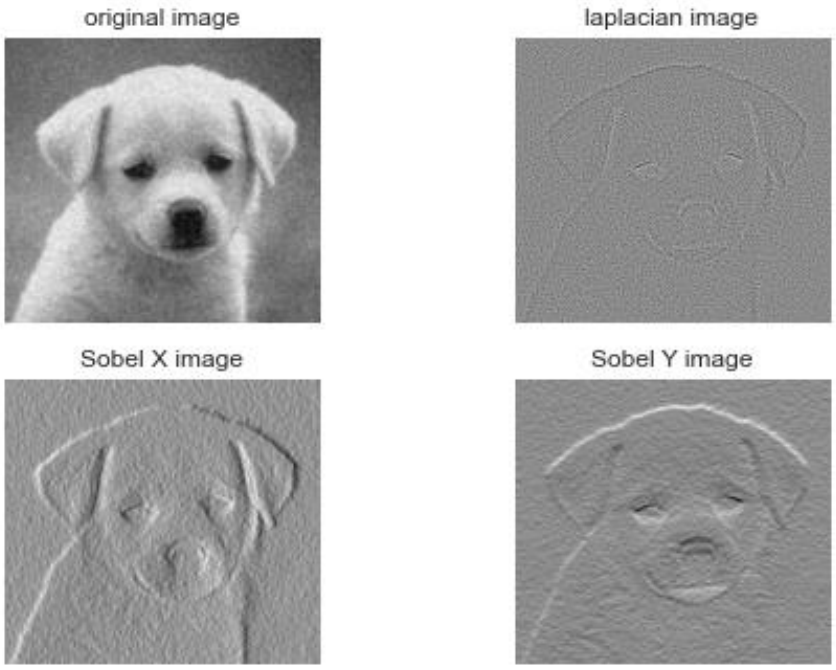
OUTPUT:

#laplacian and sobel edge detecting

```
import numpy as np
import matplotlib.pyplot as plt

image1=cv2.imread("noisy.png")
gray=cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)

img=cv2.GaussianBlur(gray,(3,3),0)

laplacian=cv2.Laplacian(img,cv2.CV_64F)
sobelx=cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely=cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

plt.subplot(2,2,1)
plt.imshow(img,cmap='gray')
plt.title('original image')
plt.xticks([])
plt.yticks([])

plt.subplot(2,2,2)
plt.imshow(laplacian,cmap='gray')
plt.title('laplacian image')
plt.xticks([])
plt.yticks([])

plt.subplot(2,2,3)
```

```
plt.imshow(sobelx,cmap='gray')
plt.title('Sobel X image')
plt.xticks([])
plt.yticks([])

plt.subplot(2,2,4)
plt.imshow(sobely,cmap='gray')
plt.title('Sobel Y image')
plt.xticks([])
plt.yticks([])

plt.show()
```

OUTPUT:

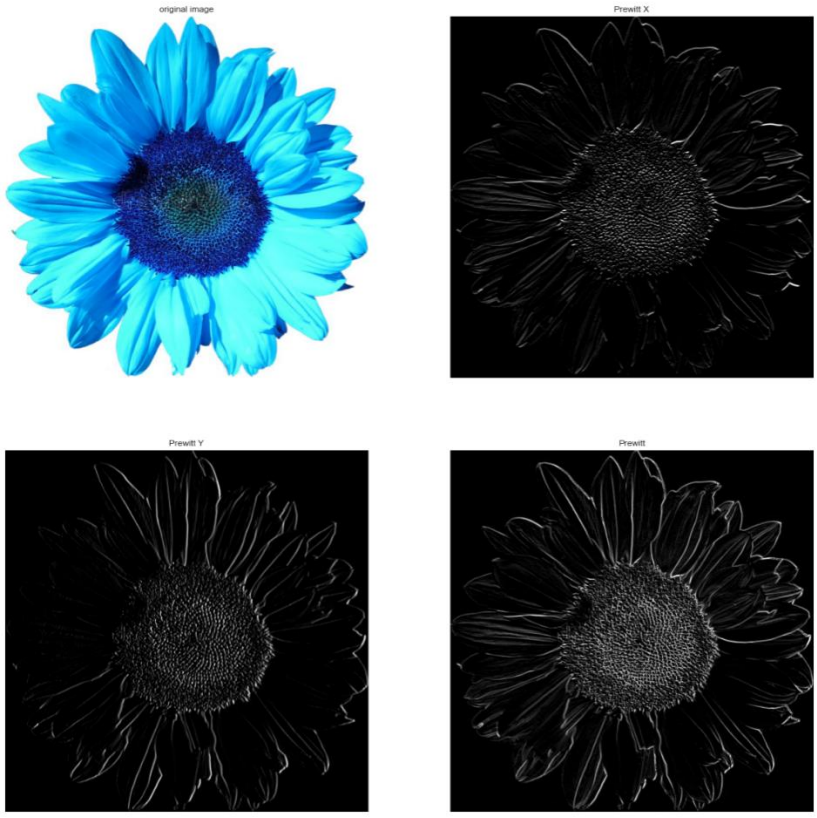


#edge detection using prewitt operator

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image1=cv2.imread("flower.jpg")
gray=cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
```

```
img=cv2.GaussianBlur(gray,(3,3),0)

kernelx=np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely=np.array([[-1,0,1],[-1,0,1],[-1,0,1]])

img_prewittx=cv2.filter2D(img,-1,kernelx)
img_prewitty=cv2.filter2D(img,-1,kernely)

plt.figure(figsize=(20,20))
plt.subplot(2,2,1)
plt.imshow(image1,cmap='gray')
plt.title('original image')
plt.axis('off')

plt.subplot(2,2,2)
plt.imshow(img_prewittx,cmap='gray')
plt.title("Prewitt X")
plt.axis('off')
plt.subplot(2,2,3)
plt.imshow(img_prewitty,cmap='gray')
plt.title("Prewitt Y")
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(img_prewittx+img_prewitty,cmap='gray')
plt.title("Prewitt")
plt.axis('off')
plt.show()
```

OUTPUT:



#roberts edge detection

```
import cv2
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt
roberts_cross_v=np.array([[1,0],[0,-1]])

roberts_cross_h=np.array([[0,1],[-1,0]])

img=cv2.imread("flower.jpg",0).astype('float64')
img/=255.0
vertical=ndimage.convolve(img,roberts_cross_v)
horizontal=ndimage.convolve(img,roberts_cross_h)

edged_img=np.sqrt(np.square(horizontal)+np.square(vertical))
```

```
edged_img*=255
cv2.imshow("Output image",edged_img)

cv2.waitKey(0)
cv2.destroyAllWindows()
plt.figure(figsize=(20,20))
plt.subplot(1,3,1)
plt.imshow(image1,cmap='gray')
plt.title('original image')
plt.axis('off')

plt.figure(figsize=(20,20))
plt.subplot(1,3,2)
plt.imshow(edged_img,cmap='gray')
plt.title('Roberts edge image')
plt.axis('off')
plt.show()
```

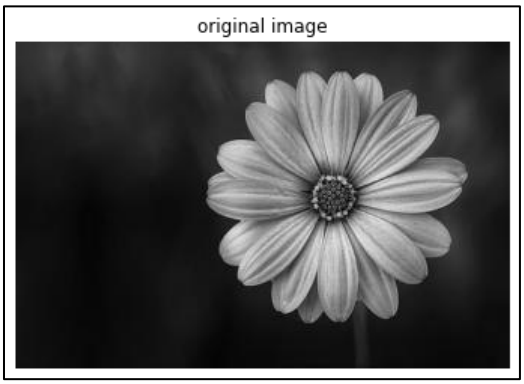
OUTPUT:

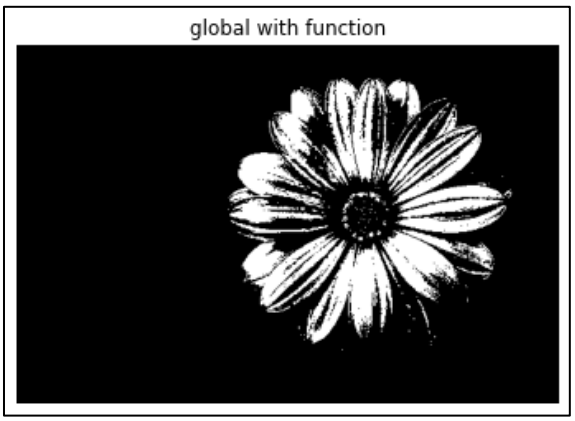
23.Develop a program to perform:
i)Global Thresholding
ii)Adaptive Thresholding
iii)Otsu thresholding using built in functions

i)Global Thresholding
CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load an image in the greyscale
img= cv2.imread('flower1.jpg',cv2.IMREAD_GRAYSCALE)

ret,th1 = cv2.threshold(img,150,255,cv2.THRESH_BINARY)

plt.figure(figsize=(20,20))
plt.subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.title('original image')
plt.axis('off')

plt.figure(figsize=(20,20))
plt.subplot(1,2,2)
plt.imshow(th1,cmap='gray')
plt.title('global with function')
plt.axis('off')

plt.show()
```

OUTPUT:



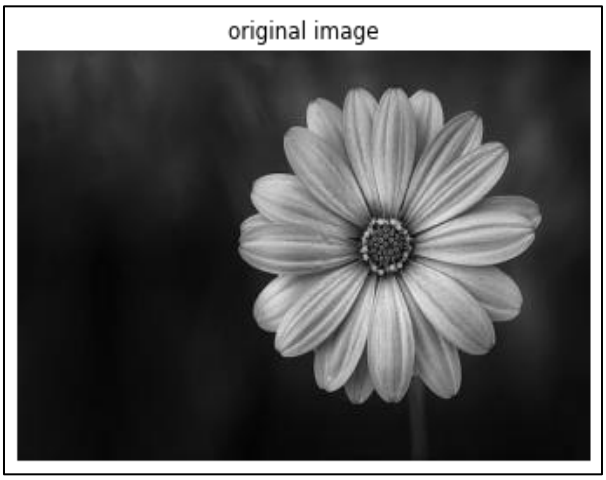


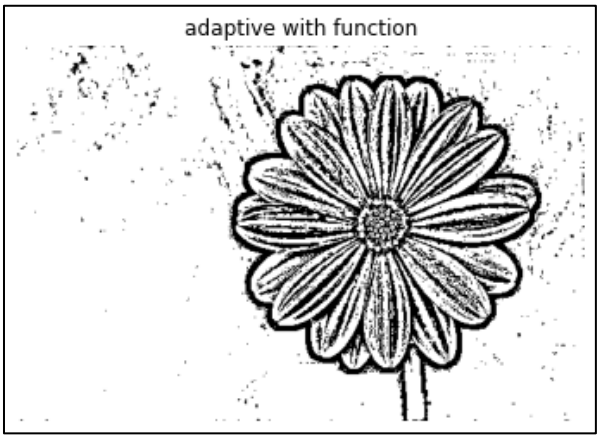
ii)Adaptive Thresholding

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

image =cv2.imread('flower1.jpg',cv2.IMREAD_GRAYSCALE)
th2=cv2.adaptiveThreshold(image,255,cv2.ADAPTIVE_THRESH_MEAN_
C,cv2.THRESH_BINARY,11,2)

th3 = cv2.adaptiveThreshold (blur,255,cv2. ADAPTIVE_THRESH_
GAUSSIAN_ C,cv2.THRESH_BINARY,11,2)

plt.figure(figsize=(20,20))
plt.subplot(1,2,1)
plt.imshow(image,cmap='gray')
plt.title('original image')
```

```
plt.axis('off')
plt.figure(figsize=(20,20))
plt.subplot(1,2,2)
plt.imshow(th2,cmap='gray')
plt.title('adaptive with function')
plt.axis('off')
plt.show()
```

OUTPUT:



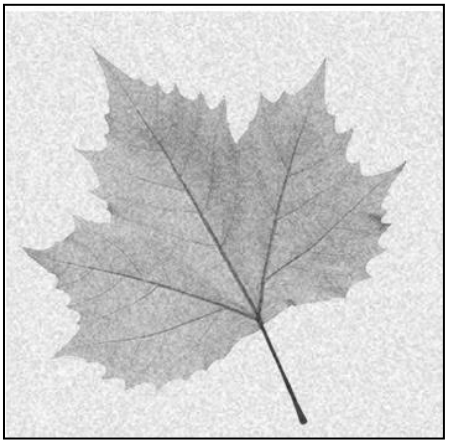


iii)Otsu thresholding using built in functions

```
import cv2
import numpy as np

img=cv2.imread('noisy.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('gray',img)
```

```
blur=cv2.GaussianBlur(img,(7,7),0)
cv2.imshow('blur',img)

x,threshold=cv2.threshold(blur,200,255,cv2.THRESH_BINARY)
cv2.imshow('Binary threshold',threshold)

ret2,th2=cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_
OTSU)
cv2.imshow('Otsus Thresholding',th2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



24.Develop a program perform region splitting approach on image.

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
import cv2
coins = data.coins()
hist = np.histogram(coins, bins=np.arange(0, 256))
fig, (ax1) = plt.subplots()
plt.axis("off")
ax1.imshow(coins, cmap=plt.cm.gray,interpolation='nearest')
```

OUTPUT:

```
from skimage.filters import sobel
elevation_map = sobel(coins)
fig, ax = plt.subplots(figsize=(4, 3))
ax.imshow(elevation_map, cmap=plt.cm.gray, interpolation='nearest')
ax.axis('off')
ax.set_title('elevation_map')
```

OUTPUT: