

```
In [21]: import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Input Image

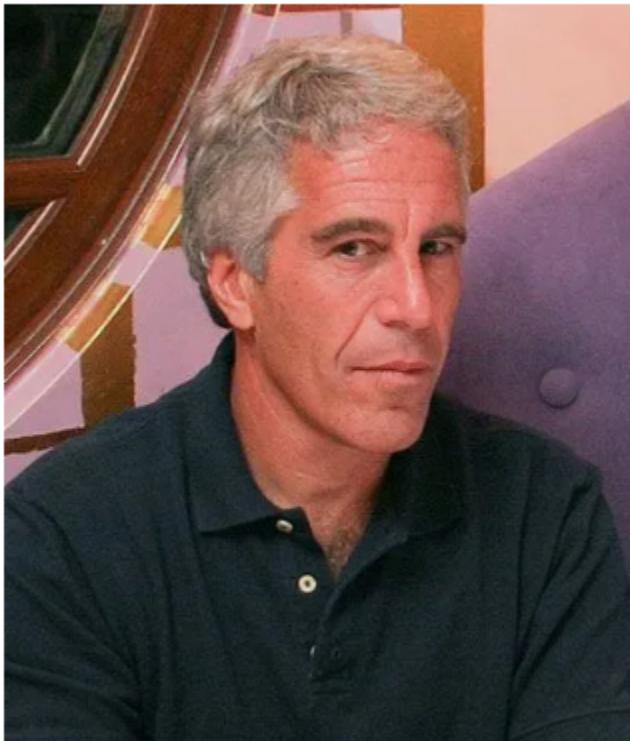
```
In [22]: img = cv2.imread(r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\Jeffrey-Epstein-S

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img_rgb)
plt.title("Original Image")
plt.axis("off")
```

Out[22]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))

Original Image



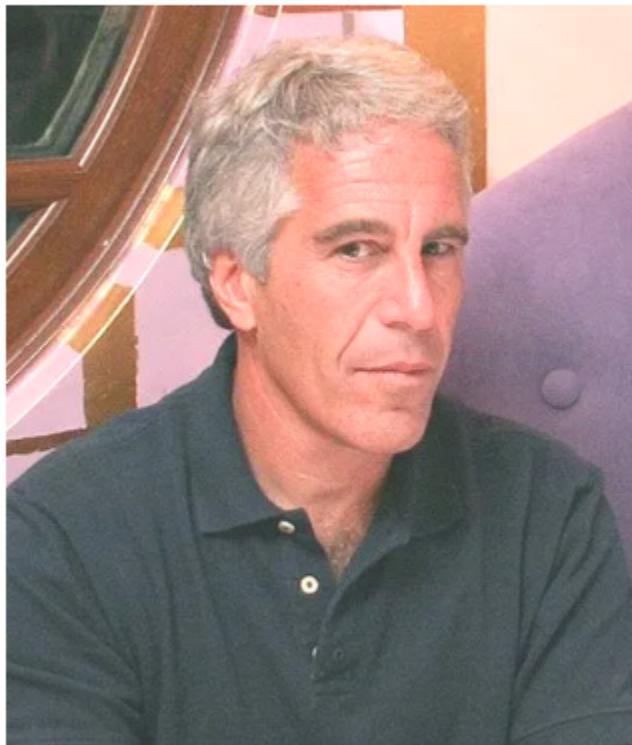
Arithmetic Operations

```
In [23]: bright = cv2.add(img, np.ones(img.shape, dtype='uint8') * 50)
bright_rgb = cv2.cvtColor(bright, cv2.COLOR_BGR2RGB)

plt.imshow(bright_rgb)
plt.title("Image Addition (Brightness Increased)")
plt.axis("off")
```

```
Out[23]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Image Addition (Brightness Increased)

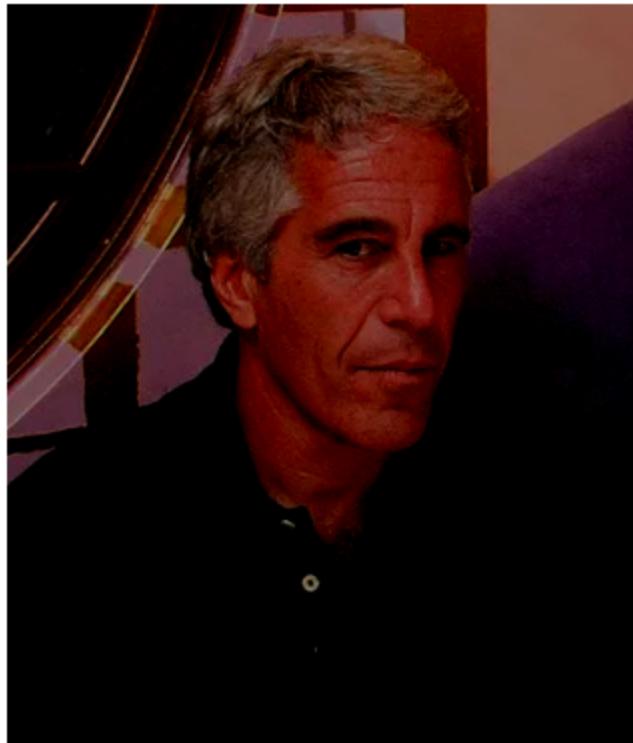


```
In [24]: dark = cv2.subtract(img, np.ones(img.shape, dtype='uint8') * 100)
dark_rgb = cv2.cvtColor(dark, cv2.COLOR_BGR2RGB)

plt.imshow(dark_rgb)
plt.title("Image Subtraction (Brightness Decreased)")
plt.axis("off")
```

```
Out[24]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Image Subtraction (Brightness Decreased)



```
In [25]: mul = cv2.multiply(img, np.array([1.2]))
mul_rgb = cv2.cvtColor(mul, cv2.COLOR_BGR2RGB)

plt.imshow(mul_rgb)
plt.title("Image Multiplication")
plt.axis("off")
```

```
Out[25]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Image Multiplication



```
In [26]: div = cv2.divide(img, np.array([1.5]))
div_rgb = cv2.cvtColor(div, cv2.COLOR_BGR2RGB)

plt.imshow(div_rgb)
plt.title("Image Division")
plt.axis("off")
```

```
Out[26]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Image Division



Logical Operations

```
In [27]: gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(gray, cmap='gray')
plt.title("Grayscale Image")
plt.axis("off")
```

```
Out[27]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Grayscale Image



```
In [28]: _, mask = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
plt.imshow(mask, cmap='gray')
plt.title("Binary Mask")
plt.axis("off")
```

```
Out[28]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Binary Mask

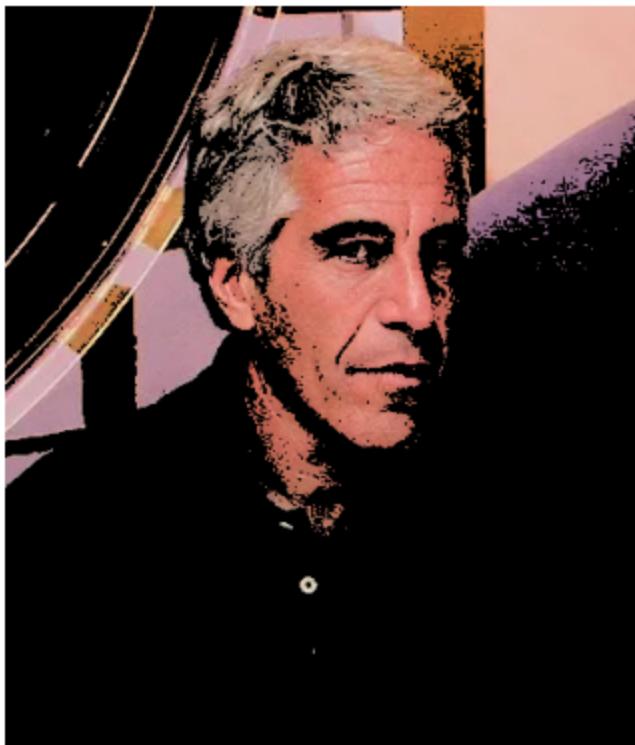


```
In [29]: bit_and = cv2.bitwise_and(img, img, mask=mask)
bit_and_rgb = cv2.cvtColor(bit_and, cv2.COLOR_BGR2RGB)

plt.imshow(bit_and_rgb)
plt.title("Bitwise AND")
plt.axis("off")
```

```
Out[29]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Bitwise AND

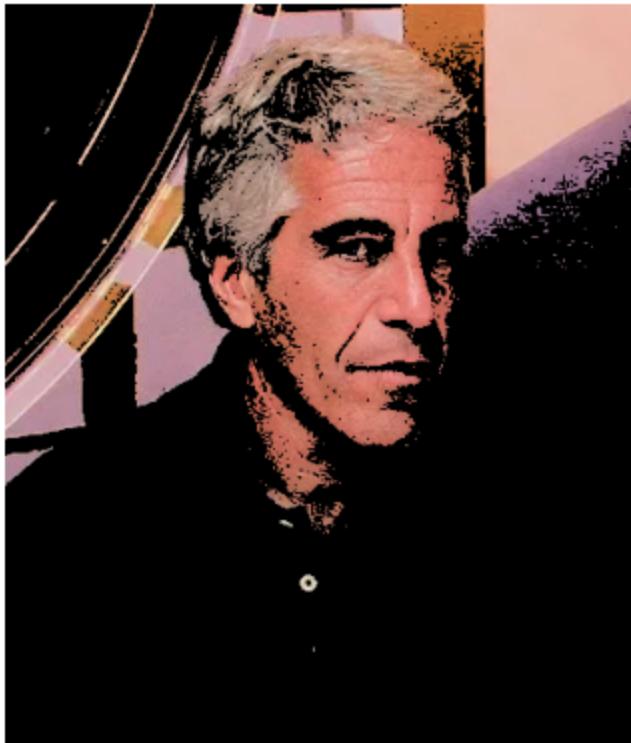


```
In [30]: bit_or = cv2.bitwise_or(img, img, mask=mask)
bit_or_rgb = cv2.cvtColor(bit_or, cv2.COLOR_BGR2RGB)

plt.imshow(bit_or_rgb)
plt.title("Bitwise OR")
plt.axis("off")
```

```
Out[30]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Bitwise OR

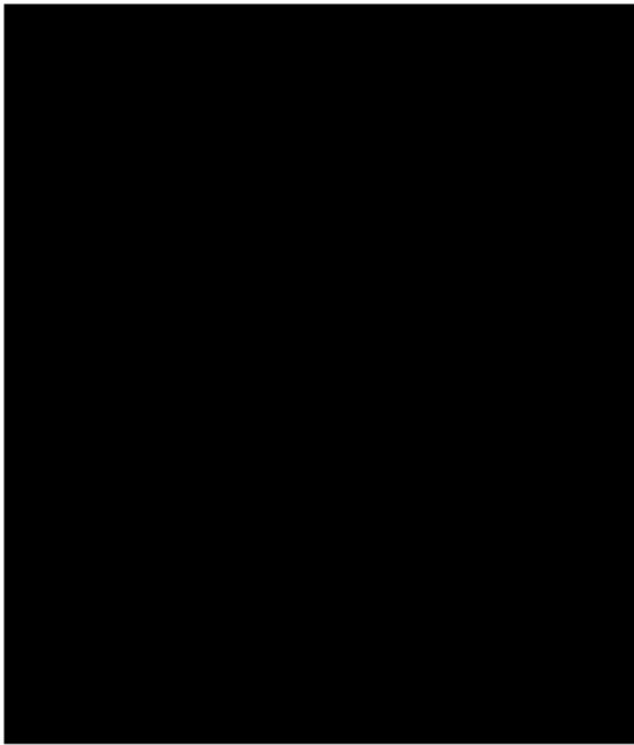


```
In [31]: bit_xor = cv2.bitwise_xor(img, img, mask=mask)
bit_xor_rgb = cv2.cvtColor(bit_xor, cv2.COLOR_BGR2RGB)

plt.imshow(bit_xor_rgb)
plt.title("Bitwise XOR")
plt.axis("off")
```

```
Out[31]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Bitwise XOR



```
In [32]: bit_not = cv2.bitwise_not(img)
bit_not_rgb = cv2.cvtColor(bit_not, cv2.COLOR_BGR2RGB)

plt.imshow(bit_not_rgb)
plt.title("Bitwise NOT (Negative Image)")
plt.axis("off")
```

```
Out[32]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Bitwise NOT (Negative Image)



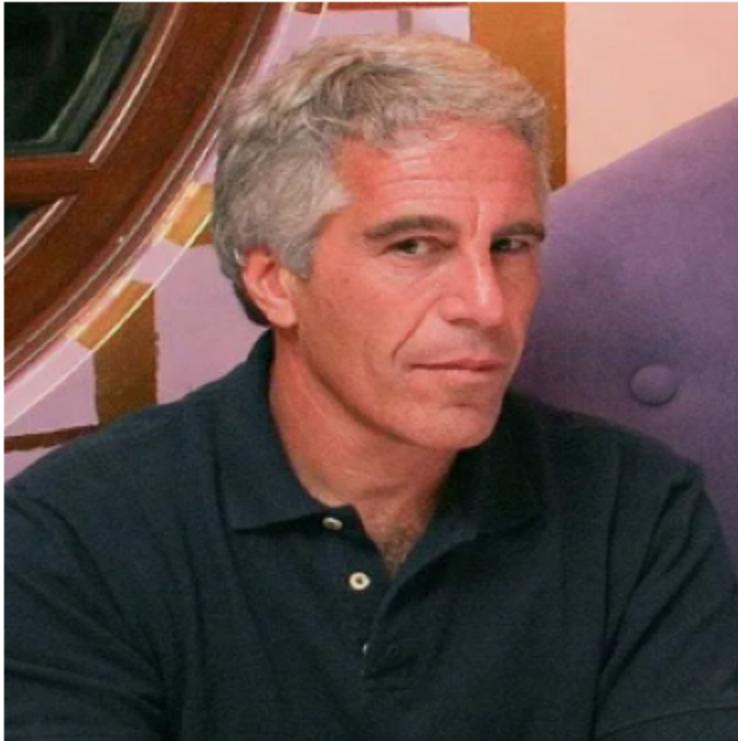
Image Resizing

```
In [33]: resized = cv2.resize(img, (300, 300), interpolation=cv2.INTER_LINEAR)
resized_rgb = cv2.cvtColor(resized, cv2.COLOR_BGR2RGB)

plt.imshow(resized_rgb)
plt.title("Resized Image")
plt.axis("off")
```

```
Out[33]: (np.float64(-0.5), np.float64(299.5), np.float64(299.5), np.float64(-0.5))
```

Resized Image



Rotation and Translation

```
In [34]: (h, w) = img.shape[:2]
center = (w//2, h//2)

matrix = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated = cv2.warpAffine(img, matrix, (w, h))
rotated_rgb = cv2.cvtColor(rotated, cv2.COLOR_BGR2RGB)

plt.imshow(rotated_rgb)
plt.title("Rotated Image (45 Degrees)")
plt.axis("off")
```

Out[34]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))

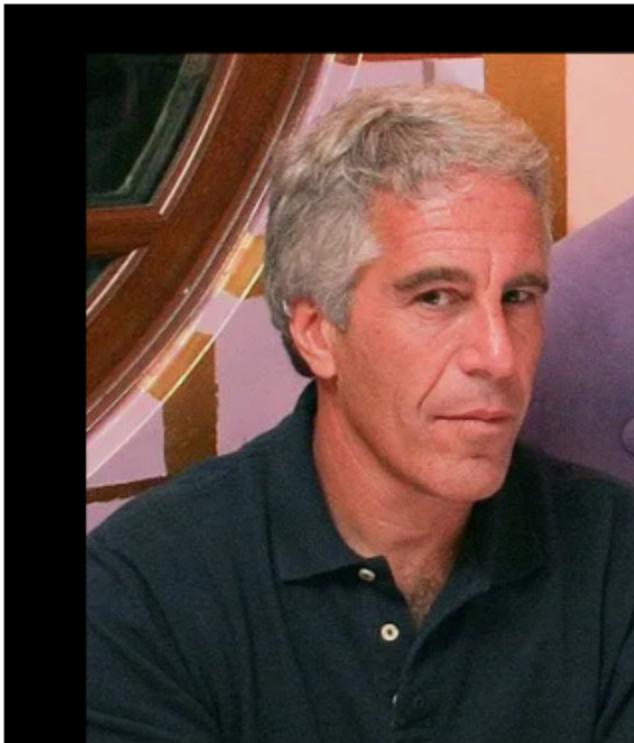
Rotated Image (45 Degrees)



```
In [35]: matrix_t = np.float32([[1, 0, 50],  
                           [0, 1, 30]])  
  
translated = cv2.warpAffine(img, matrix_t, (w, h))  
translated_rgb = cv2.cvtColor(translated, cv2.COLOR_BGR2RGB)  
  
plt.imshow(translated_rgb)  
plt.title("Translated Image")  
plt.axis("off")
```

Out[35]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))

Translated Image

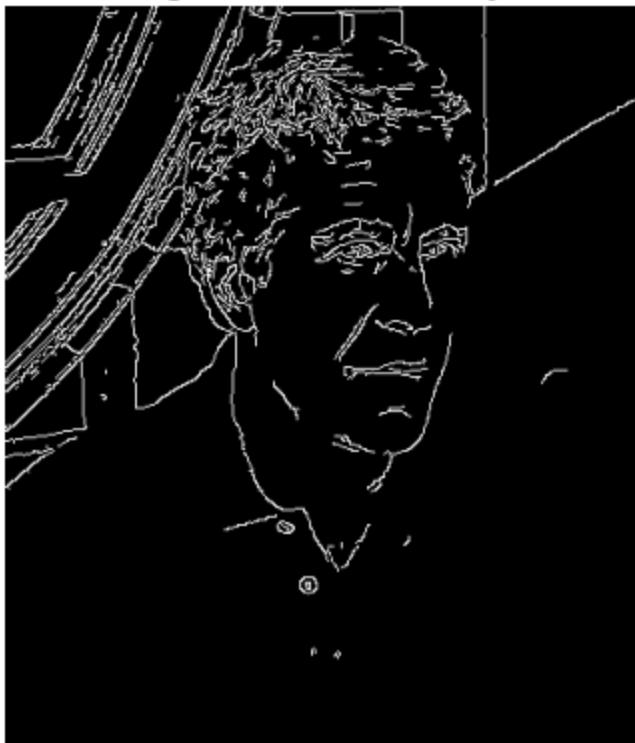


Additional Basic Operations

```
In [36]: edges = cv2.Canny(gray, 100, 200)
plt.imshow(edges, cmap='gray')
plt.title("Edge Detection (Canny)")
plt.axis("off")
```

```
Out[36]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Edge Detection (Canny)



```
In [37]: blur = cv2.GaussianBlur(img, (7,7), 0)
blur_rgb = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)

plt.imshow(blur_rgb)
plt.title("Gaussian Blurred Image")
plt.axis("off")
```

```
Out[37]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Gaussian Blurred Image



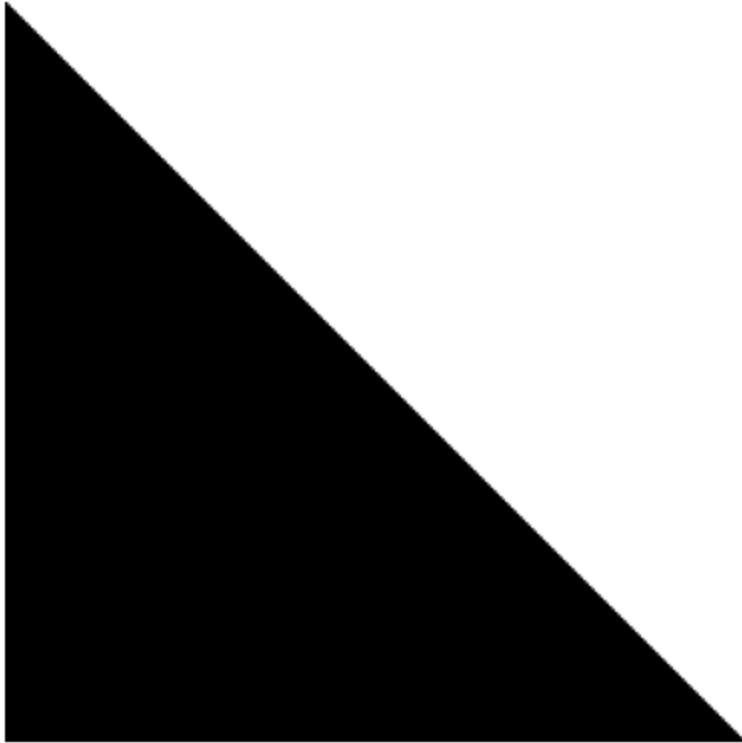
```
In [39]: img1 = np.zeros((256, 256), dtype=np.uint8)

for i in range(256):
    for j in range(256):
        if j > i:
            img1[i, j] = 255    # White
        else:
            img1[i, j] = 0      # Black

plt.imshow(img1, cmap='gray')
plt.title("Diagonal Half Black–White Image")
plt.axis("off")
```

```
Out[39]: (np.float64(-0.5), np.float64(255.5), np.float64(255.5), np.float64(-0.5))
```

Diagonal Half Black-White Image

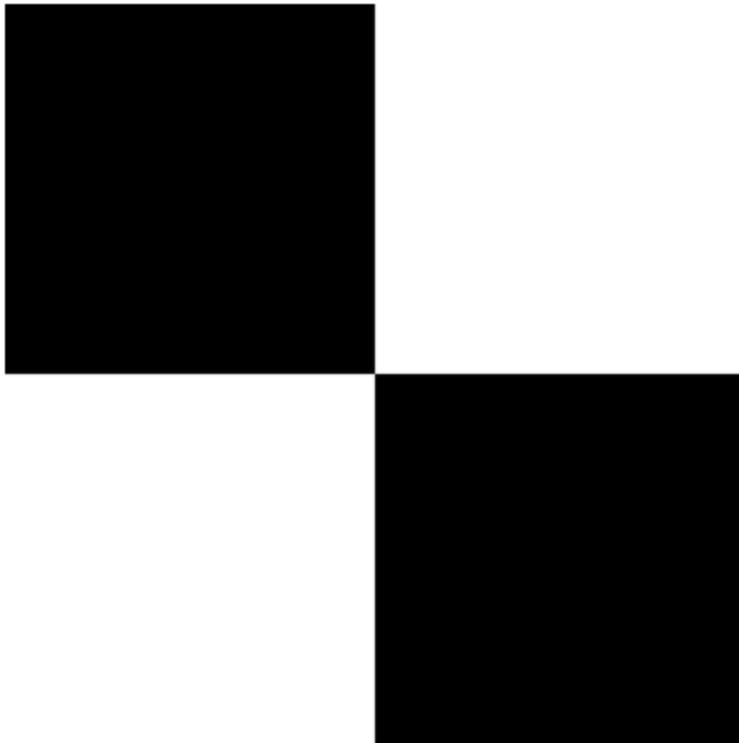


```
In [40]: img2 = np.zeros((256, 256), dtype=np.uint8)

img2[0:128, 0:128] = 0
img2[0:128, 128:256] = 255
img2[128:256, 0:128] = 255
img2[128:256, 128:256] = 0
plt.imshow(img2, cmap='gray')
plt.title("2x2 Quadrant Image")
plt.axis("off")
```

```
Out[40]: (np.float64(-0.5), np.float64(255.5), np.float64(255.5), np.float64(-0.5))
```

2x2 Quadrant Image



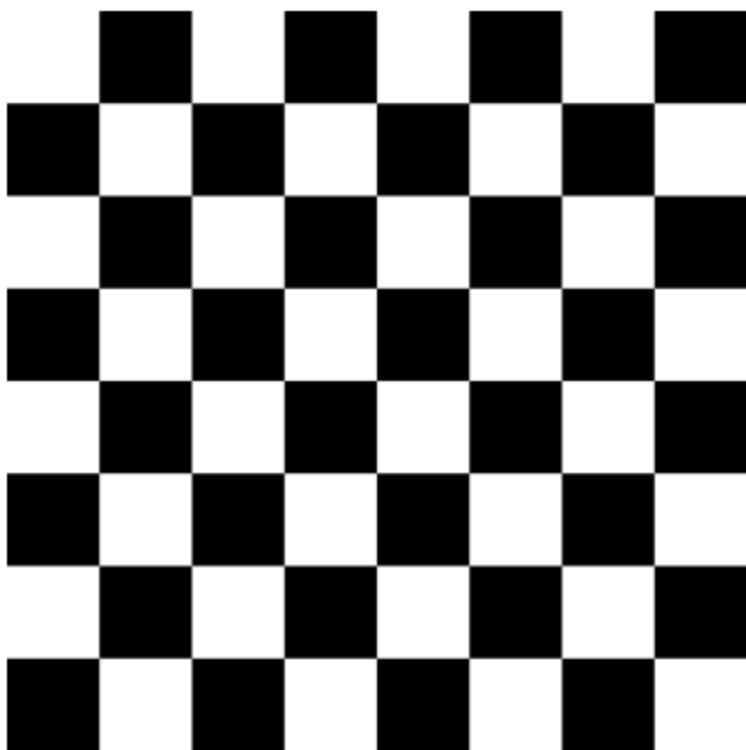
```
In [41]: img3 = np.zeros((256, 256), dtype=np.uint8)
block = 32

for i in range(0, 256, block):
    for j in range(0, 256, block):
        if (i//block + j//block) % 2 == 0:
            img3[i:i+block, j:j+block] = 255

plt.imshow(img3, cmap='gray')
plt.title("Chessboard Pattern")
plt.axis("off")
```

```
Out[41]: (np.float64(-0.5), np.float64(255.5), np.float64(255.5), np.float64(-0.5))
```

Chessboard Pattern



Algorithm: Piecewise Linear Transformation

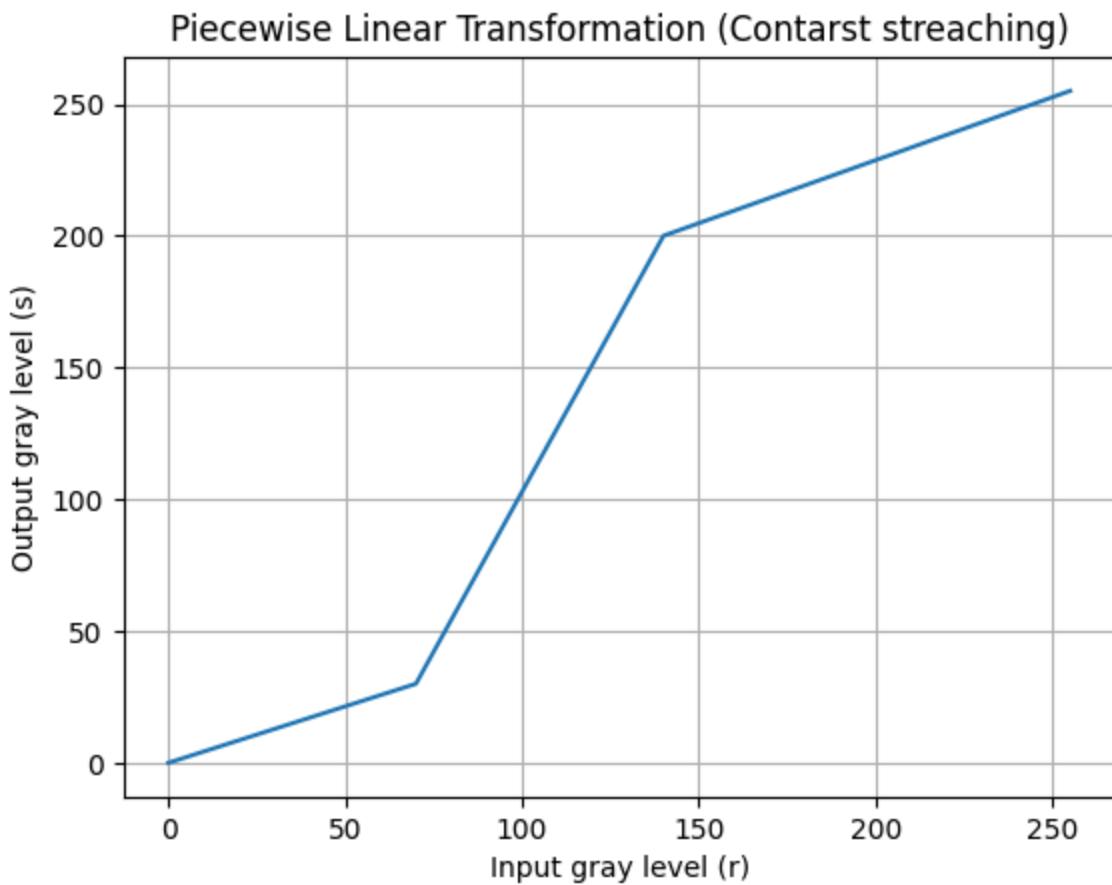
```
In [67]: import numpy as np  
import matplotlib.pyplot as plt
```

```
In [68]: r = np.arange(256)  
s = np.zeros(256)
```

```
In [69]: r1 , r2 = 70, 140  
s1 , s2 = 30 , 200
```

```
In [70]: for i in range(256):  
    if r[i] < r1:  
        s[i] = (s1/r1) * r[i]  
    elif r[i] <= r2:  
        s[i] = ((s2 - s1)/(r2 - r1)) * (r[i] - r1) + s1  
    else:  
        s[i] = ((255 - s2)/(255 - r2)) * (r[i] - r2) + s2
```

```
In [71]: plt.plot(r, s)  
plt.xlabel("Input gray level (r)")  
plt.ylabel("Output gray level (s)")  
plt.title("Piecewise Linear Transformation (Contrast stretching)")  
plt.grid()  
plt.show()
```



```
In [72]: import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
In [ ]: img = cv2.imread(
    r"D:\MIT WPU B.Tech Data\3rd Year Data\IPPR\puppy-with-gray-scale-noise.we
    b
)
```

Piecewise Linear Transformation (ON IMAGE)

```
In [83]: r_vals = np.arange(256)
s_vals = np.zeros(256)

r1, r2 = 70, 140
s1, s2 = 30, 200

for i in range(256):
    if r_vals[i] < r1:
        s_vals[i] = (s1 / r1) * r_vals[i]
    elif r_vals[i] <= r2:
```

```

        s_vals[i] = ((s2 - s1) / (r2 - r1)) * (r_vals[i] - r1) + s1
    else:
        s_vals[i] = ((255 - s2) / (255 - r2)) * (r_vals[i] - r2) + s2

plt.figure(figsize=(15, 4))

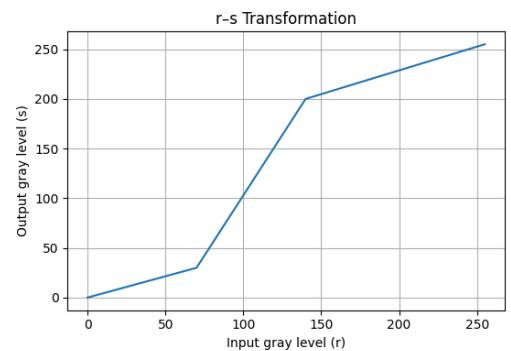
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(piecewise, cmap='gray')
plt.title("Piecewise Linear Output")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("Input gray level (r)")
plt.ylabel("Output gray level (s)")
plt.title("r-s Transformation")
plt.grid()

plt.tight_layout()
plt.show()

```



Gray Level Slicing WITHOUT Background

```
In [85]: r_vals = np.arange(256)
s_vals = np.zeros(256)

r1, r2 = 100, 150

for i in range(256):
```

```

if r1 <= r_vals[i] <= r2:
    s_vals[i] = 255
else:
    s_vals[i] = 0
plt.figure(figsize=(15, 4))

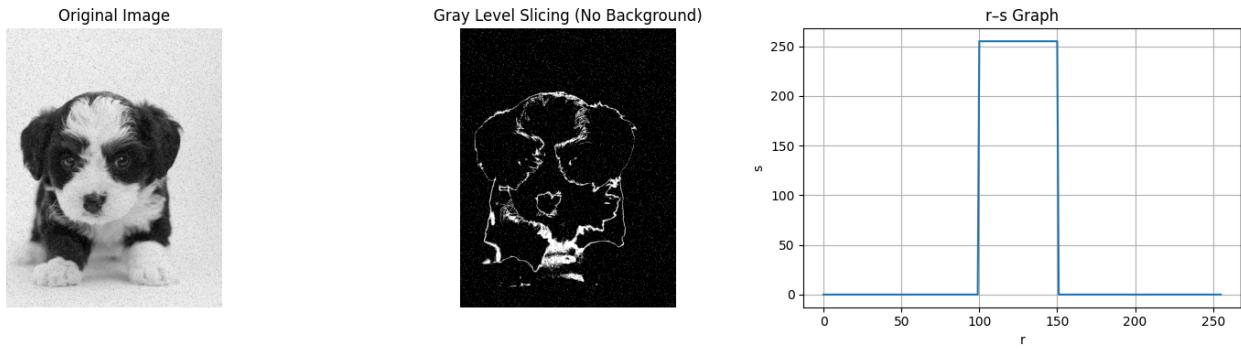
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(slice_no_bg, cmap='gray')
plt.title("Gray Level Slicing (No Background)")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("r")
plt.ylabel("s")
plt.title("r-s Graph")
plt.grid()

plt.tight_layout()
plt.show()

```



Gray Level Slicing WITH Background

```

In [86]: r_vals = np.arange(256)
s_vals = np.zeros(256)

for i in range(256):
    if r1 <= r_vals[i] <= r2:
        s_vals[i] = 255
    else:
        s_vals[i] = r_vals[i]
plt.figure(figsize=(15, 4))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')

```

```

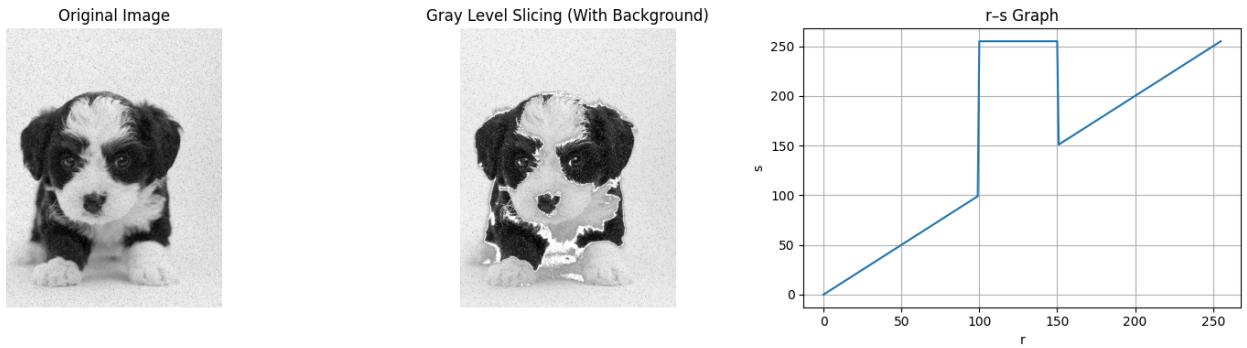
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(slice_with_bg, cmap='gray')
plt.title("Gray Level Slicing (With Background)")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("r")
plt.ylabel("s")
plt.title("r-s Graph")
plt.grid()

plt.tight_layout()
plt.show()

```



Contrast Stretching

```

In [87]: r_vals = np.arange(256)
s_vals = ((r_vals - r_min) / (r_max - r_min)) * 255
plt.figure(figsize=(15, 4))

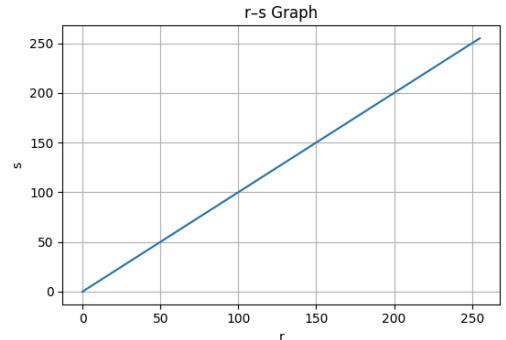
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(contrast_stretch, cmap='gray')
plt.title("Contrast Stretched Image")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("r")
plt.ylabel("s")
plt.title("r-s Graph")
plt.grid()

```

```
plt.tight_layout()  
plt.show()
```



Thresholding

```
In [88]: T = 120  
r_vals = np.arange(256)  
s_vals = np.where(r_vals >= T, 255, 0)  
plt.figure(figsize=(15, 4))  
  
plt.subplot(1, 3, 1)  
plt.imshow(img, cmap='gray')  
plt.title("Original Image")  
plt.axis("off")  
  
plt.subplot(1, 3, 2)  
plt.imshow(threshold_img, cmap='gray')  
plt.title("Thresholded Image")  
plt.axis("off")  
  
plt.subplot(1, 3, 3)  
plt.plot(r_vals, s_vals)  
plt.xlabel("r")  
plt.ylabel("s")  
plt.title("r-s Graph")  
plt.grid()  
  
plt.tight_layout()  
plt.show()
```

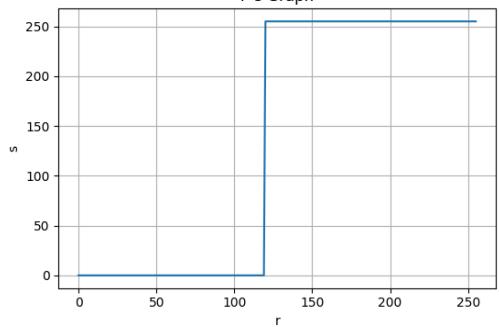
Original Image

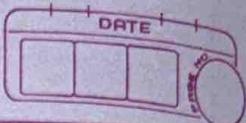


Thresholded Image



r-s Graph





IPPR

- * Exp-1
- * Post lab Questions.

1) What do you mean by gray level?
→ In image processing, a grey level refers to the intensity value of a pixel in a grayscale image. It tells you how bright or dark pixel is.

2) Write the expression to find the number of bits to store a digital image.
→ The expression to store digital image using number of bits is.

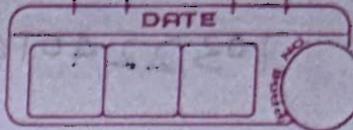
$$\text{Num of bits} = M \times N \times K$$

M - number of rows.
N - number of columns.
K - number of bits per pixel (bpp)

3) Name types of resolutions w.r.t a digital image.

→ The main types of resolution w.r.t a digital image are.

- 1) Spatial resolution
- 2) Grey level (intensity) Resolution
- 3) Temporal resolution.
- 4) Physical level resolution.



4) Specify the elements of the DIP system.

→ A typical Digital Image processing system includes.

1) Image Acquisition

2) Image Storage

3) Image Processing

4) Image Display

5) Image Transmission

6) Image Interpretation

5) Write any four applications of DIP

→ Common application of DIP are.

1) Medical Imaging (e.g., CT, MRI)

2) Remote sensing (satellite)

3) Robotics & Machine vision

4) Bio metrics (Face, fingerprint).

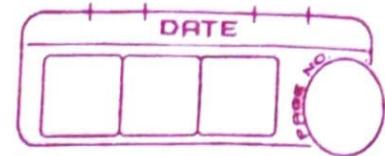


Image
Display

Computer

Mass
storage

Hard
copy
Device

Image processing
hardware

Image
processing
software

Image
Sensor

Problem - Domain.