TITLE:	DATE: PAGE NO:

AIM: Familiarization of the tool used for computer vision.

CODE:

1. pip list

```
Package         Version
-------------------------------------------
absl-py1.4.0
accelerate      1.3.0
aiohappyeyeballs        2.4.6
aiohttp 3.11.12
aiosignal       1.3.2
alabaster       1.0.0
albucore        0.0.23
albumentations          2.0.4
ale-py  0.10.2
altair  5.5.0
annotated-types         0.7.0
anyio   3.7.1
argon2-cffi     23.1.0
argon2-cffi-bindings    21.2.0
array_record    0.6.0
arviz   0.20.0
astropy7.0.1
astropy-iers-data       0.2025.2.17.0.34.13
astunparse      1.6.3
atpublic        4.1.0
attrs   25.1.0
audioread       3.0.1
autograd        1.7.0
...
yellowbrick     1.5
yfinance        0.2.54
zipp    3.21.0
zstandard       0.23.0
```

!pip install opencv-python

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python) (1.26.4)
```

2. !pip install tensorflow

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
```

TITLE:	DATE: PAGE NO:

/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages ...
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,>=2.18->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist- packages (from rich->keras>=3.5.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

3. !pip install matplotlib

Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)

4. !pip install numpy

Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.26.4)

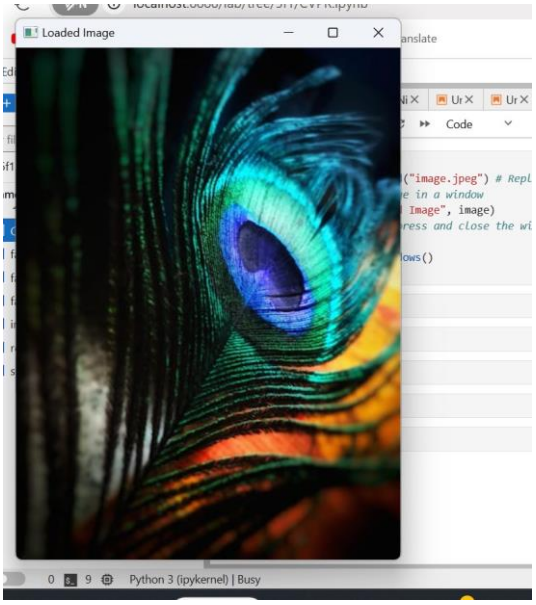
TITLE:	DATE: PAGE NO:

AIM: Write a Program to read and display an image.

CODE:

```
import cv2
# Load an image from file
image = cv2.imread("s1.jpg") # Display the image in a window
cv2.imshow("Loaded Image", image)
# Wait for a key press and close the window
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



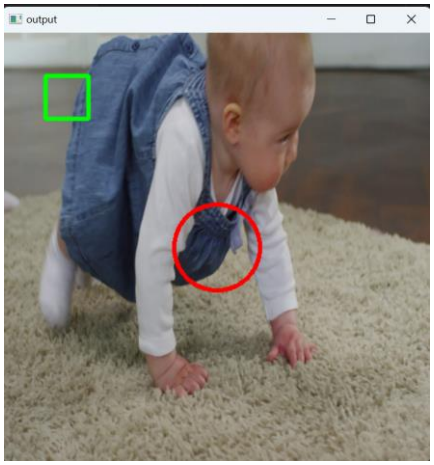
TITLE:	DATE: PAGE NO:

AIM: Write a Program to read and write video files.

CODE:

```
import cv2
cap = cv2.VideoCapture("input_video.mp4")
if not cap.isOpened():
    print("Error: Could not open the video file.")
    exit()
output = cv2.VideoWriter(
    "output.avi", cv2.VideoWriter_fourcc(*'XVID'), 10, (500, 500))
while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Failed to capture frame.")
        break
    frame = cv2.resize(frame, (500, 500))
    cv2.rectangle(frame, (50, 50), (100, 100), (0, 255, 0), 3)
    center = (250, 250)
    radius = 50
    cv2.circle(frame, center, radius, (0, 0, 255), 3)
    output.write(frame)
    cv2.imshow("output", frame)
    if cv2.waitKey(1) & 0xFF == ord('s'):
        break
cv2.destroyAllWindows()
output.release()
cap.release()
```

OUTPUT:



TITLE:	DATE: PAGE NO:

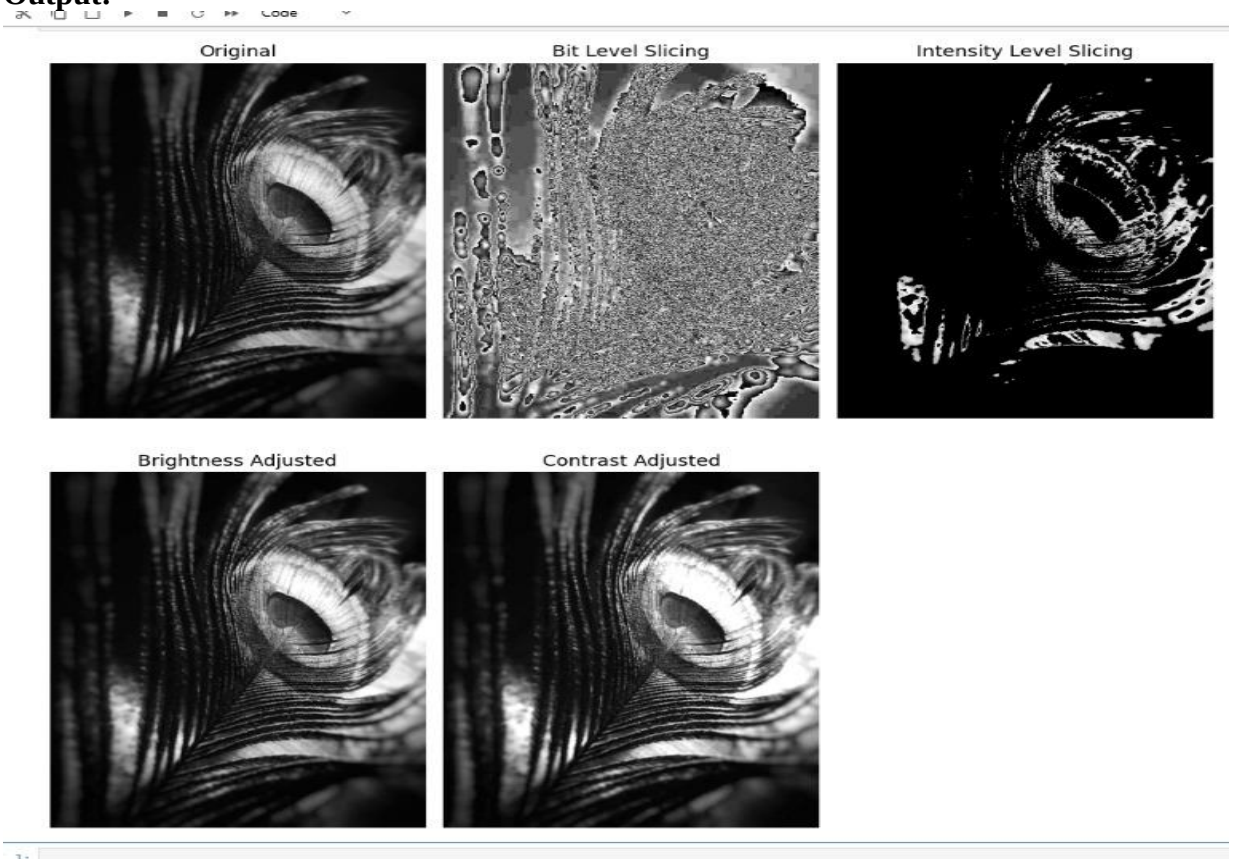
AIM: Write a Program to enhance an image using basic functions.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the image in grayscale
image = cv2.imread('mango.jpeg', cv2.IMREAD_GRAYSCALE)
# Function for Bit Level Slicing
def bit_level_slicing(image, bit_position):
    max_bit = 1 << bit_position  # Extracting the specific bit plane
    sliced_image = np.bitwise_and(image, max_bit)
    return sliced_image
# Function for Intensity Level Slicing
def intensity_level_slicing(image, low, high):
    sliced_image = np.zeros_like(image)
    sliced_image[(image >= low) & (image <= high)] = image[(image >= low) & (image <= high)]
    return sliced_image
# Function for Brightness Adjustment
def adjust_brightness(image, brightness_value):
    brightness_adjusted = cv2.add(image, np.full_like(image, brightness_value, dtype=np.uint8))
    return brightness_adjusted
# Function for Contrast Adjustment
def adjust_contrast(image, contrast_value):
    contrast_adjusted = cv2.convertScaleAbs(image, alpha=contrast_value, beta=0)
    return contrast_adjusted
# Function to Display Images
def display_images(images, titles):
    plt.figure(figsize=(10, 10))
    for i, (img, title) in enumerate(zip(images, titles)):
        plt.subplot(2, 3, i + 1)
        plt.imshow(img, cmap='gray')
        plt.title(title)
        plt.axis('off')
    plt.tight_layout()
    plt.show()
# Apply transformations
bit_sliced = bit_level_slicing(image, 4)  # Example: Extracting the 4th bit plane
intensity_sliced = intensity_level_slicing(image, 100, 150)
brightness_adjusted = adjust_brightness(image, 50)
contrast_adjusted = adjust_contrast(image, 1.5)
# Display the results
images = [image, bit_sliced, intensity_sliced, brightness_adjusted, contrast_adjusted]
titles = ['Original', 'Bit Level Slicing', 'Intensity Level Slicing', 'Brightness Adjusted', 'Contrast Adjusted']
display_images(images, titles)
```

TITLE:	DATE: PAGE NO:

Output:



TITLE:	DATE: PAGE NO:

AIM: Write a Program to enhance an image using transformational functions image negative and piecewise linear functions.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the image in grayscale
image = cv2.imread("mango.jpeg", cv2.IMREAD_GRAYSCALE)
# 1. Image Negative Transformation
negative = 255 - image
# 4. Piecewise Linear Transformation
def piecewise_linear(img):
    r1, s1 = 50, 0
    r2, s2 = 150, 255
    img = img.astype(np.float32)  # Convert to float for calculations
    # Apply transformation
    img_transformed = np.piecewise(
        img,
        [img <= r1, (img > r1) & (img <= r2), img > r2],
        [
            lambda r: (s1 / r1) * r,
            lambda r: ((s2 - s1) / (r2 - r1)) * (r - r1) + s1,
            lambda r: ((255 - s2) / (255 - r2)) * (r - r2) + s2,
        ]
    )
    return np.uint8(img_transformed)
# Apply Piecewise Linear Transformation
piecewise_transformed = piecewise_linear(image)
# Display all images
titles = ['Original', 'Negative', 'Piecewise Linear']
images = [image, negative, piecewise_transformed]
plt.figure(figsize=(10, 6))
for i in range(3):
    plt.subplot(1, 3, i + 1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.tight_layout()
plt.show()
```

TITLE:	DATE: PAGE NO:

OUTPUT:

TITLE:	DATE: PAGE NO:

AIM: Write a Program to enhance an image using log transformation and power law transformation.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the image in grayscale
image = cv2.imread("mango.jpeg", cv2.IMREAD_GRAYSCALE)
# Check if the image is loaded correctly
if image is None:
    print("Error: Image not found or cannot be loaded.")
else:
    # Avoid division by zero by adding a small constant (epsilon)
    epsilon = 1e-5  # Small constant to prevent log(0) issues
    # 2. Log Transformation (s = c * log(1 + r))
    c = 255 / np.log(1 + np.max(image) + epsilon)  # Compute constant safely
    log_transformed = c * np.log(1 + image.astype(np.float32) + epsilon)  # Apply log function
    log_transformed = np.uint8(log_transformed)  # Convert back to uint8
    # 3. Power Law (Gamma) Transformation (s = c * r^γ)
    gamma = 2.2  # Adjust brightness
    gamma_transformed = np.array(255 * ((image / 255 + epsilon) ** gamma), dtype=np.uint8)
    # Display the images
    titles = ['Original', 'Log Transform', 'Power Law (Gamma)']
    images = [image, log_transformed, gamma_transformed]
    plt.figure(figsize=(10, 6))
    for i in range(3):
        plt.subplot(1, 3, i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(titles[i])
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

Output:

TITLE:	DATE: PAGE NO:

AIM: Write a program to implement histogram calculation and equalization for the given image.

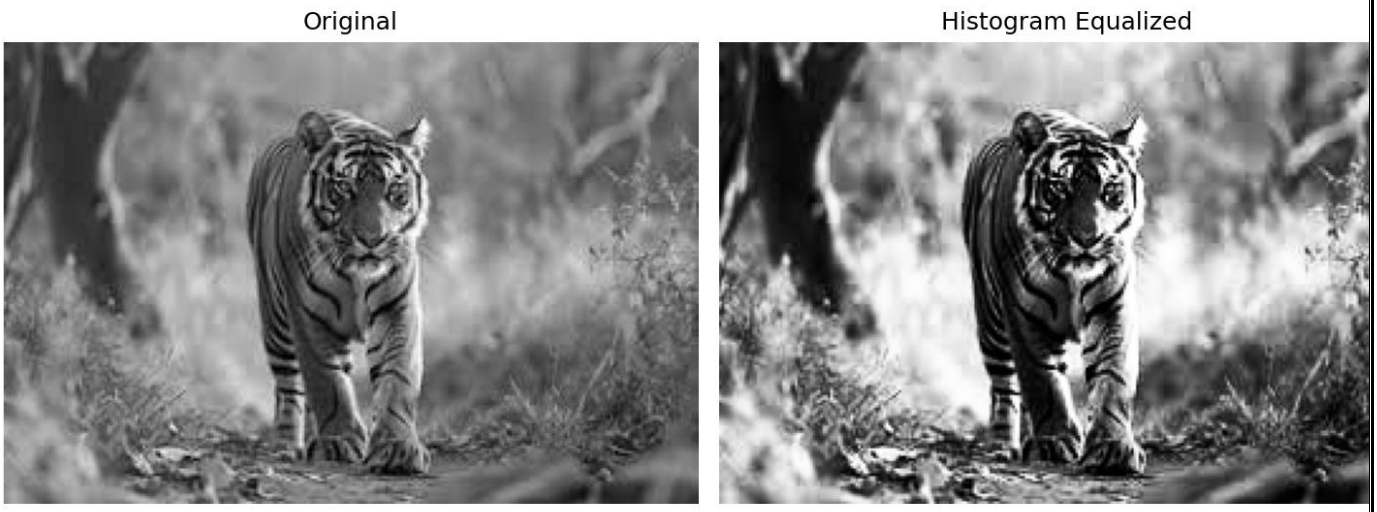
CODE:

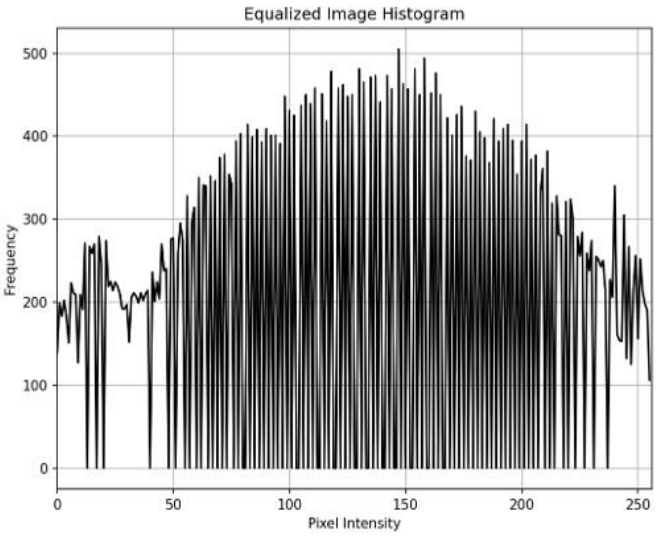
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image in grayscale
image = cv2.imread("mango.jpeg", cv2.IMREAD_GRAYSCALE)

# Check if image is loaded correctly
if image is None:
    print("Error: Image not found or cannot be loaded.")
else:
    # 1. Calculate and plot histogram
    def calculate_histogram(image, title="Histogram"):
        histogram = cv2.calcHist([image], [0], None, [256], [0, 256])
        plt.figure(figsize=(8, 6))
        plt.plot(histogram, color='black')
        plt.title(title)
        plt.xlabel("Pixel Intensity")
        plt.ylabel("Frequency")
        plt.xlim([0, 256])
        plt.grid()
        plt.show()

    # 2. Histogram Equalization
    def histogram_equalization(image):
        return cv2.equalizeHist(image)

    # Apply histogram equalization
    equalized_image = histogram_equalization(image)

    # Display original and equalized images
    titles = ['Original', 'Histogram Equalized']
    images = [image, equalized_image]

    plt.figure(figsize=(10, 6))
    for i in range(len(images)):
        plt.subplot(1, 2, i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(titles[i])
        plt.axis('off')

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

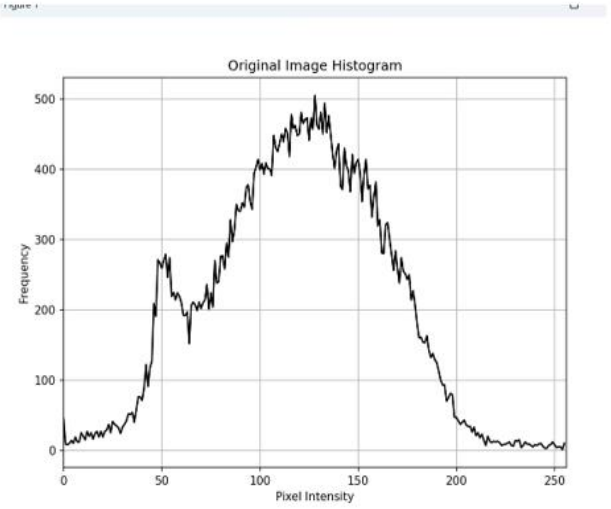
TITLE:	DATE: PAGE NO:

```
# Plot histogram of original and equalized images
calculate_histogram(image, "Original Image Histogram")
calculate_histogram(equalized_image, "Equalized Image Histogram")
```

Output:







TITLE:	DATE: PAGE NO:

AIM: Write a program to implement histogram specification.

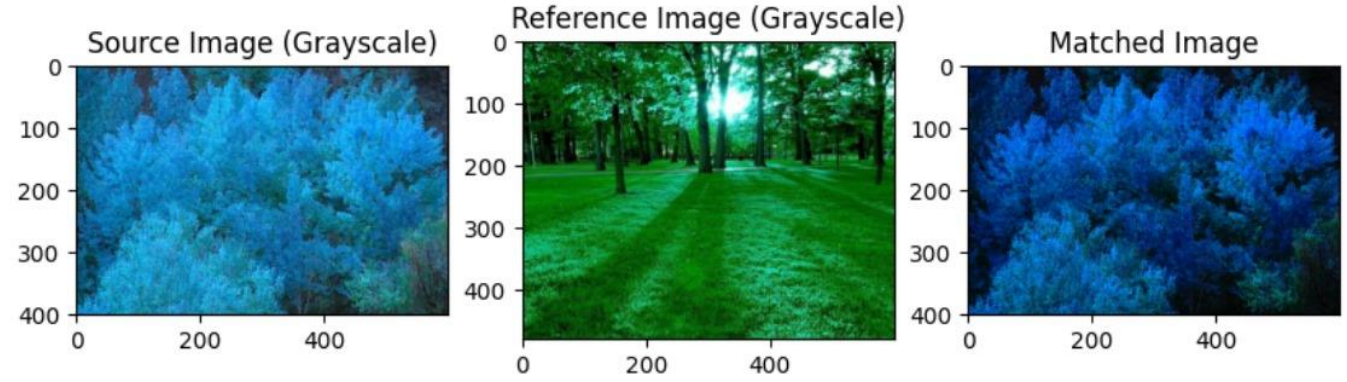
CODE:

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

def hist_match(source, reference):
    """
    Matches the histogram of the source image to that of the reference image.
    Works on each color channel separately (R, G, B).
    """
    matched = source.copy()  # Copy source image to apply changes

    for i in range(3):  # Loop through each color channel (R, G, B)
        # Compute histogram for source and reference images
        src_hist, _ = np.histogram(source[:, :, i].ravel(), 256, [0, 256])
        ref_hist, _ = np.histogram(reference[:, :, i].ravel(), 256, [0, 256])

        # Compute cumulative distribution functions (CDFs)
        src_cdf = np.cumsum(src_hist) / src_hist.sum()
        ref_cdf = np.cumsum(ref_hist) / ref_hist.sum()

        # Create a mapping from source to reference
        mapping = np.interp(src_cdf, ref_cdf, np.arange(256))

        # Apply the mapping to transform the pixel values
        matched[:, :, i] = np.interp(source[:, :, i], np.arange(256), mapping)

    return np.uint8(matched)  # Convert back to uint8 format

# Load source and reference images
src = cv2.imread('a1.webp')  # Read source image
ref = cv2.imread('a2.png')  # Read reference image

# Check if images are loaded correctly
if src is None or ref is None:
    print("Error: Image not found. Check file paths.")
else:
    # Convert images from BGR (OpenCV default) to RGB (for displaying correctly)
    src = cv2.cvtColor(src, cv2.COLOR_BGR2RGB)
    ref = cv2.cvtColor(ref, cv2.COLOR_BGR2RGB)

    # Perform histogram matching
    matched = hist_match(src, ref)

    # Display images side by side
    images = [src, ref, matched]
```

TITLE:	DATE: PAGE NO:

```
titles = ['Source Image', 'Reference Image', 'Histogram Matched Image']

plt.figure(figsize=(12, 4))
for i in range(3):
    plt.subplot(1, 3, i+1)  # Create subplots
    plt.imshow(images[i])     # Show the image
    plt.title(titles[i])      # Set title
    plt.axis('off')           # Hide axes

plt.show()  # Display images
```

Output:



TITLE:	DATE: PAGE NO:

AIM: Write a program to perform Arithmetic operators.

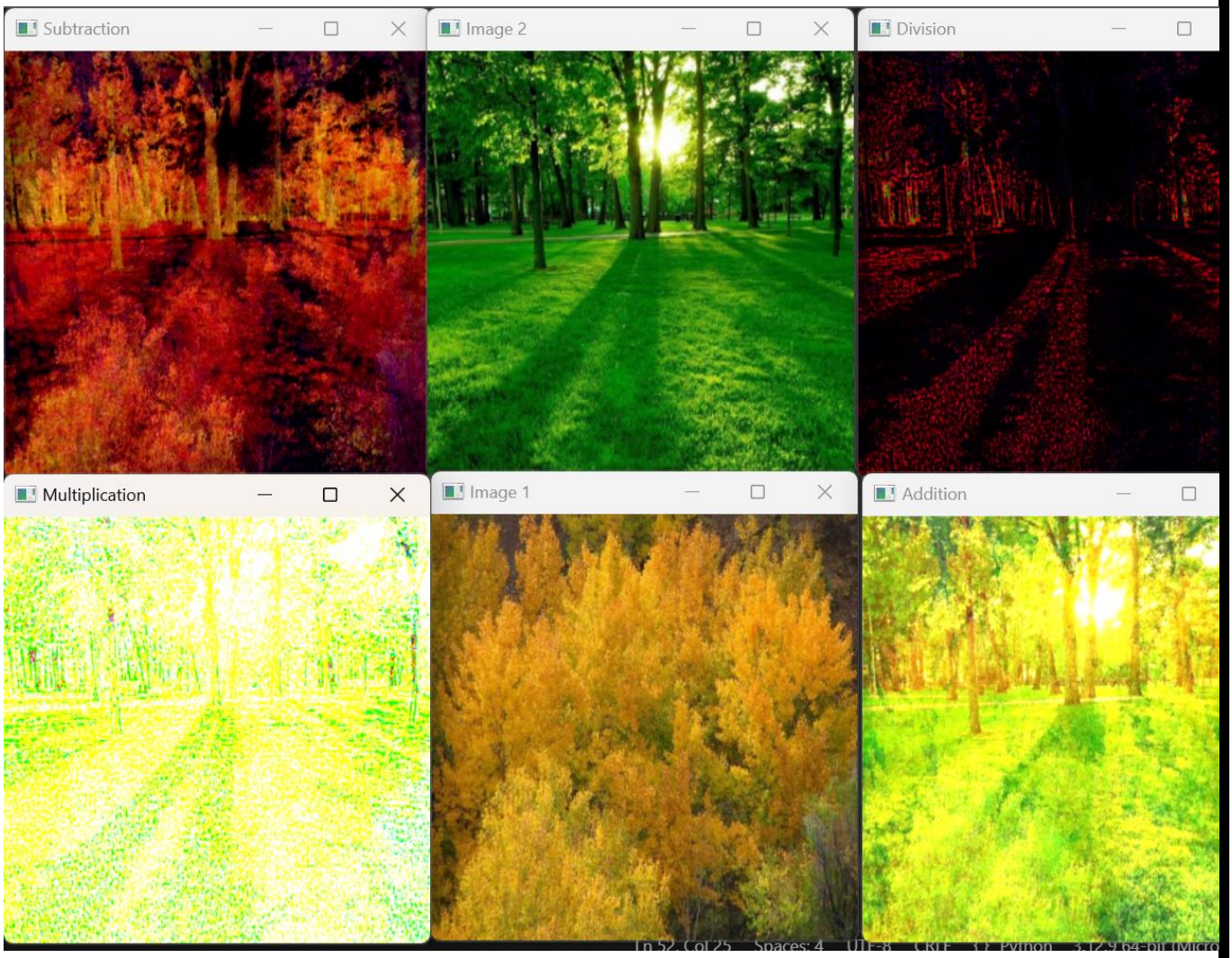
CODE:

```
import cv2
import numpy as np

def arithmetic_operations(image1_path, image2_path):
    # Load images
    image1 = cv2.imread(image1_path)
    image2 = cv2.imread(image2_path)

    # Check if images are loaded correctly
    if image1 is None or image2 is None:
        print("Error: One or both images not found. Check file paths.")
        return

    # Resize image2 to match image1's dimensions
    image2 = cv2.resize(image2, (image1.shape[1], image1.shape[0]))

    # Perform arithmetic operations
    add_result = cv2.add(image1, image2)
    subtract_result = cv2.subtract(image1, image2)
    multiply_result = cv2.multiply(image1, image2)

    # Convert images to float32 for division to avoid divide-by-zero errors
    image1_float = image1.astype(np.float32)
    image2_float = np.where(image2 == 0, 1, image2).astype(np.float32)  # Avoid division by zero
    divide_result = cv2.divide(image1_float, image2_float)

    # Convert division result back to uint8 for display
    divide_result = np.clip(divide_result, 0, 255).astype(np.uint8)

    # Resize results for better display
    display_size = (300, 300)
    image1_resized = cv2.resize(image1, display_size)
    image2_resized = cv2.resize(image2, display_size)
    add_resized = cv2.resize(add_result, display_size)
    subtract_resized = cv2.resize(subtract_result, display_size)
    multiply_resized = cv2.resize(multiply_result, display_size)
    divide_resized = cv2.resize(divide_result, display_size)
    # Show results
    cv2.imshow('Image 1', image1_resized)
    cv2.imshow('Image 2', image2_resized)
    cv2.imshow('Addition', add_resized)
    cv2.imshow('Subtraction', subtract_resized)
    cv2.imshow('Multiplication', multiply_resized)
    cv2.imshow('Division', divide_resized)
    cv2.waitKey(0)
```

TITLE:	DATE: PAGE NO:

```
    cv2.destroyAllWindows()

# Paths to images (Update with actual file paths)
image1_path = 'm1.jpeg'  # Replace with your first image file
image2_path = 'm2.jpg'   # Replace with your second image file

# Run the function
arithmetic_operations(image1_path, image2_path)
```

Output:



TITLE:	DATE: PAGE NO:

AIM: Write a program to perform Logical operators.

CODE:

```
import cv2
import numpy as np
def logical_operations(image1_path, image2_path):
    # Load images in grayscale mode
    image1 = cv2.imread(image1_path, cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imread(image2_path, cv2.IMREAD_GRAYSCALE)

    # Check if images are loaded correctly
    if image1 is None or image2 is None:
        print("Error: One or both images not found. Check file paths.")
        return
    # Resize images to 200x200 for consistency
    image1 = cv2.resize(image1, (200, 200))
    image2 = cv2.resize(image2, (200, 200))
    # Perform logical operations
    and_result = cv2.bitwise_and(image1, image2)  # AND operation
    or_result = cv2.bitwise_or(image1, image2)    # OR operation
    xor_result = cv2.bitwise_xor(image1, image2)  # XOR operation
    not_result1 = cv2.bitwise_not(image1)         # NOT operation on image1
    not_result2 = cv2.bitwise_not(image2)         # NOT operation on image2
    # Show original images
    cv2.imshow('Image 1', image1)
    cv2.imshow('Image 2', image2)
    # Show results of logical operations
    cv2.imshow('AND Operation', and_result)
    cv2.imshow('OR Operation', or_result)
    cv2.imshow('XOR Operation', xor_result)
    cv2.imshow('NOT Image 1', not_result1)
    cv2.imshow('NOT Image 2', not_result2)
    # Wait for user to press a key and close windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()
# Paths to images (Update with actual file paths)
image1_path = 'm1.jpeg'  # Replace with your first image file
image2_path = 'm2.jpg'  # Replace with your second image file
# Run the function
logical_operations(image1_path, image2_path)
```

TITLE:	DATE: PAGE NO:

output: