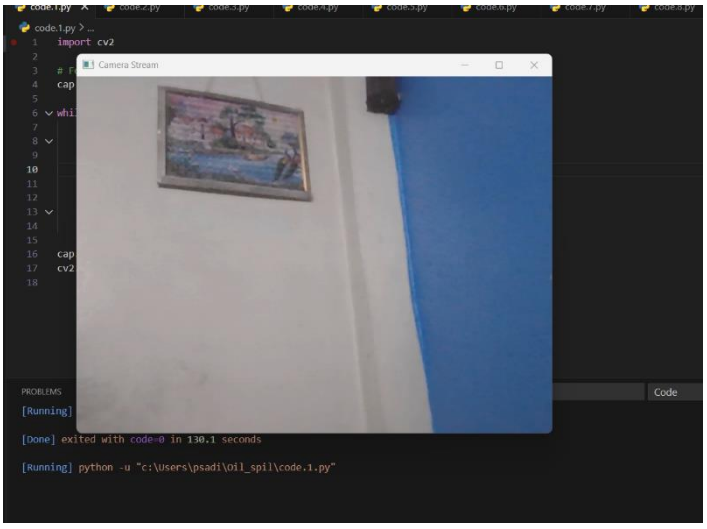
CODE-1

```
import cv2
# For USB webcam (index 0 = first camera)
cap = cv2.VideoCapture(0)
while True:
ret, frame = cap.read()
if not ret:
break
cv2.imshow("Camera Stream", frame)
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:



Purpose of the Code

This script captures live video from your USB webcam and displays it in a window. It's mainly used to test if your camera is working and to get a live feed for further computer vision processing.

LOGIC

1. **Import OpenCV** We use cv2 library to work with images and videos.

2. **Start the Webcam** cv2.VideoCapture(0) opens the first camera so we can capture video frames.

3. **Keep Reading Frames** We use a loop to read one frame at a time from the camera continuously.

4. **Check if Frame is Captured** If a frame is not read properly, we stop the loop.

5. **Show the Video** Each frame is displayed in a window so we can see the live video.

6. **Stop the Video** If we press "q", the loop ends and the video stops.

7. **Clean Up** Close the camera and all OpenCV windows properly.

Output:

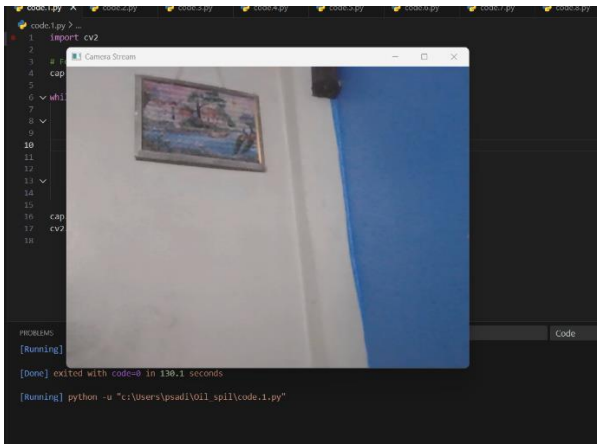
A window opens showing live video from the webcam, and it stops when "q" is pressed.

CODE-2

```
import os

# Start webcam

cap = cv2.VideoCapture(0)

# Create output folder

os.makedirs("frames", exist_ok=True)

frame_count = 0

while True:

    ret, frame = cap.read()  # Read frame from camera

    if not ret:

        break

    # Show live video

    cv2.imshow("Camera Stream", frame)

    # Save each frame as an image

    filename = f"frames/frame_{frame_count:06d}.jpg"

    cv2.imwrite(filename, frame)

    frame_count += 1

    # Stop when 'q' is pressed
```

```
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
# Release camera and close windows
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:



Logic Explanation :

1. **Imports the** os **module** to handle folder creation and file management.

2. **Starts the webcam** using cv2.VideoCapture(0) — 0 refers to the default camera on the system.

3. **Creates an output folder** named "frames" using os.makedirs("frames", exist_ok=True) so that each captured frame can be saved there.

4. Initializes a **frame counter** (frame_count = 0) to keep track of saved image numbers.

5. Enters a **continuous loop** to:

    ***Read frames** from the webcam (cap.read()).

    ***Display the live video feed** in a window titled *"Camera Stream"*.

    ***Save each captured frame** to the "frames" folder with filenames like frame_000001.jpg, frame_000002.jpg, etc.

    Increment the frame counter after saving each image.

The loop **runs continuously** until the user presses the **'q' key**, which breaks the loop.

Releases the webcam and **closes all OpenCV windows** using cap.release() and cv2.destroyAllWindows() to free system resources.

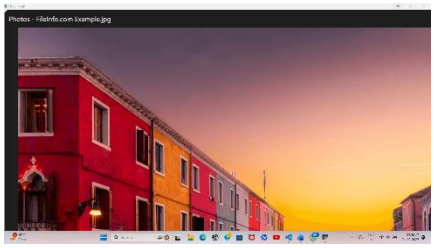
Purpose:

To **capture live video frames from the webcam** and **save each frame as an image** in a specified folder.This program is useful for **collecting real-time image data**, which can later be used for **image processing, object detection, or machine learning** applications.

CODE 3:

```
import cv2
# Read an image
img = cv2.imread  (r"C:\Users\psadi\Downloads\jpg_44-2.jpg")
# replace with your file path

# Check if image loaded successfully
if img is None:
    print("Error: Could not read image.")
else:
    # Show the image in a window
    cv2.imshow("My Image", img)
    # Wait until a key is pressed, then close
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

OUTPUT:



Logic Explanation :

1. Imports the OpenCV library (cv2) for image processing.

2. Reads an image from the specified file path and stores it as a NumPy array.

3. Checks if the image was successfully loaded; prints an error message if not.

4. Opens a window to display the image.

5. Waits indefinitely until a key is pressed.

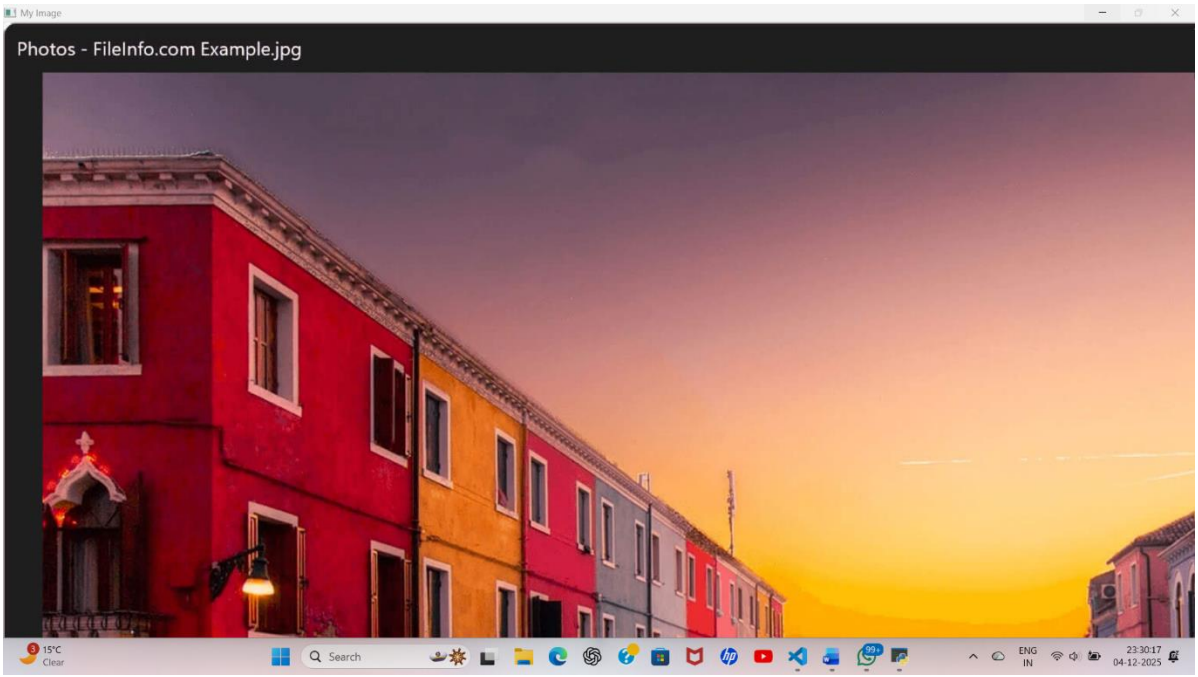
6. Closes the display window and releases any resources used by OpenCV.

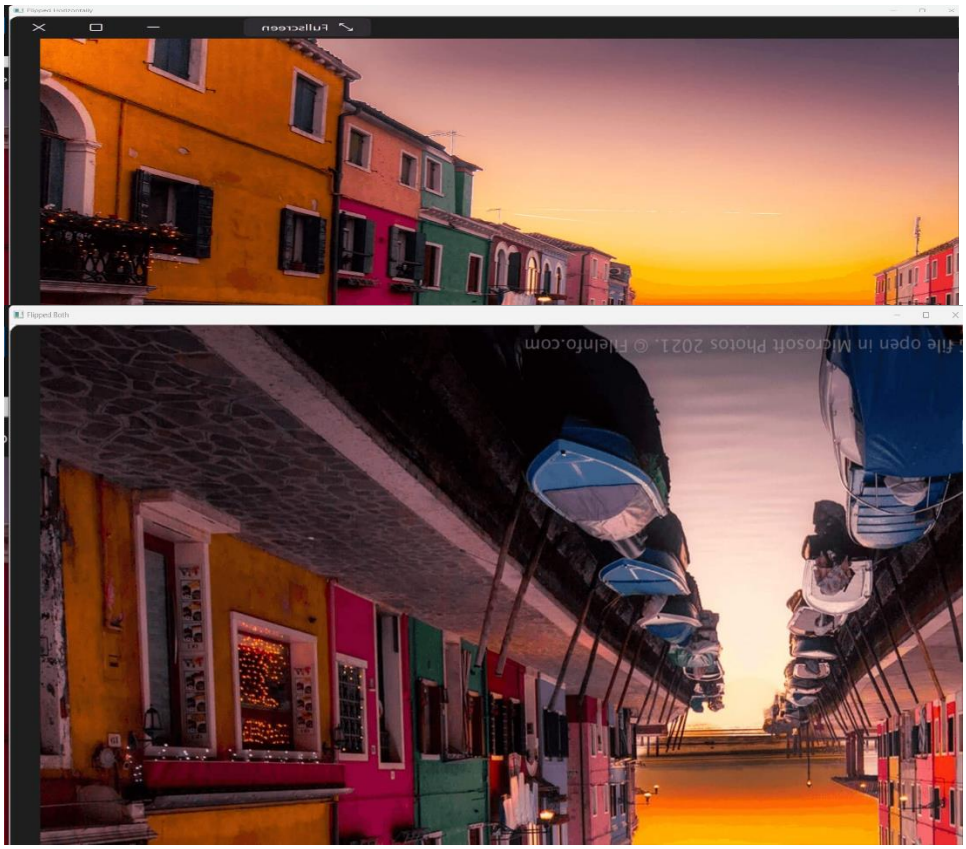
Purpose:
To load an image from disk and display it in a window, allowing users to visualize images programmatically and verify that the file was correctly read.

CODE 4:

```
import cv2
# Read image
img = cv2.imread (r"C:\Users\psadi\Downloads\jpg_44-2.jpg")
# Check if image loaded successfully
if img is None:
    print("Error: Could not read image.")  # <-- Indented
else:
    # Flip vertically (0), horizontally (1), or both (-1)
    flip_vertical = cv2.flip(img, 0)    # 0 is for vertical flip
    flip_horizontal = cv2.flip(img, 1)  # 1 for horizontal flip
    flip_both = cv2.flip(img, -1)       # -1 for both
    # Show results
    cv2.imshow("Original", img)
    cv2.imshow("Flipped Vertically", flip_vertical)
    cv2.imshow("Flipped Horizontally", flip_horizontal)
    cv2.imshow("Flipped Both", flip_both)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

INPUT :





Code Logic: Image Flipping using OpenCV

1. Imports the OpenCV library (cv2) for image processing.
2. Reads an image from the specified file path and stores it as a NumPy array.
3. Checks if the image was successfully loaded; prints an error message if not.
4. Creates three flipped versions of the image using cv2.flip:

    *Vertically (flip_vertical)

    *Horizontally (flip_horizontal)

    *Both vertically and horizontally (flip_both)

5. Displays the original image along with the three flipped versions in separate windows.
6. Waits indefinitely until a key is pressed.
7. Closes all display windows and releases OpenCV resources.

Purpose:
To demonstrate how to flip an image in different orientations using OpenCV,

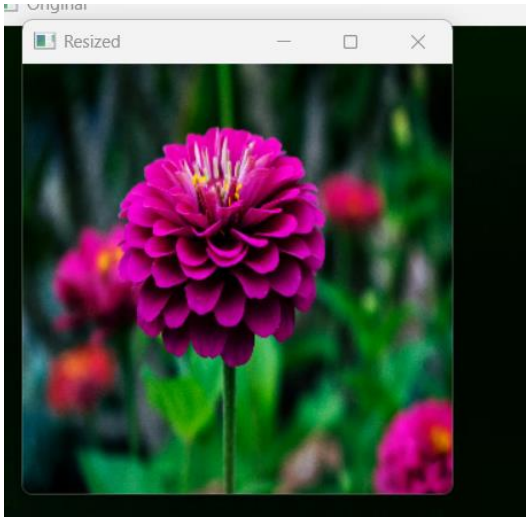
enabling visualization of the original and transformed images for tasks like data augmentation or image analysis.

CODE 5:

```
import cv2
# Load an image
img = cv2.imread (r"C:\Users\psadi\Downloads\nature-flowers-garden-863963.jpg")
# Replace with your file path
# Check if the image loaded correctly
if img is None:
    print("Error: Could not read image.")
    exit()
# Resize image (width=300, height=300)
resized = cv2.resize(img, (300, 300))
# Show both images
cv2.imshow("Original", img)
cv2.imshow("Resized", resized)
# Wait for a key press, then close windows
cv2.waitKey(0)
cv2.destroyAllWindows()
# Optional: Save the resized image
cv2.imwrite("resized_output.jpg", resized)
```

INPUT:

OUTPUT:



Code Logic: Image resized using OpenCV

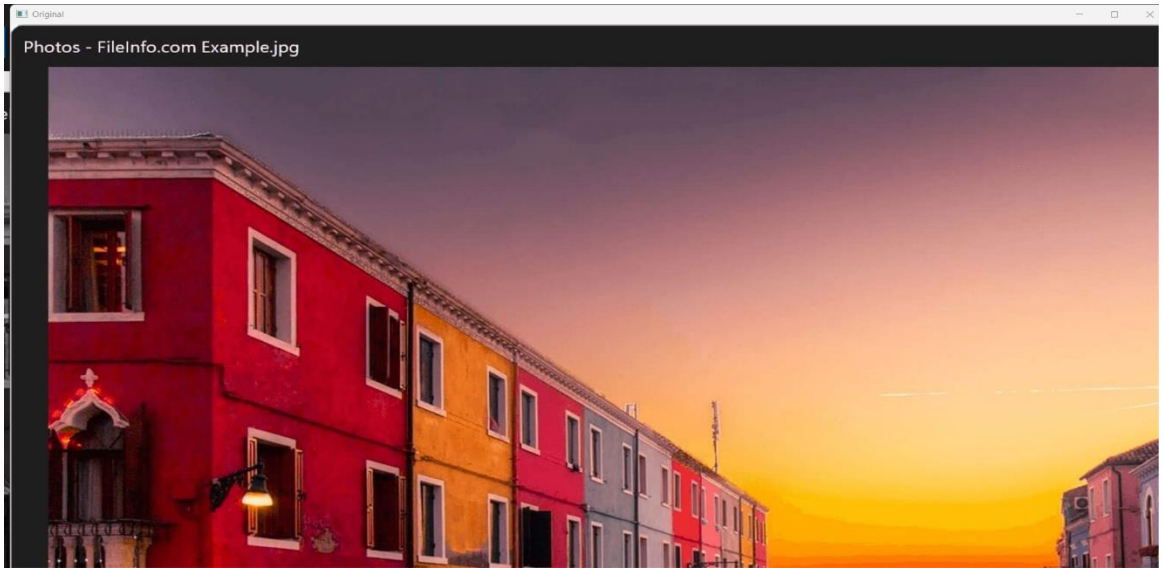
1. Imports the OpenCV library (cv2) for image processing.
2. Reads an image from the specified file path and stores it as a NumPy array.
3. Checks if the image was successfully loaded; prints an error message and exits if not.
4. Resizes the image to a fixed dimension of 300×300 pixels using cv2.resize.
5. Displays both the original and resized images in separate windows.
6. Waits indefinitely until a key is pressed.
7. Closes all display windows and releases OpenCV resources.
8. Optionally saves the resized image to disk using cv2.imwrite.

Purpose:
To demonstrate how to resize an image using OpenCV while allowing comparison between the original and transformed images, which is useful for preprocessing images for machine learning or standardizing image dimensions for display

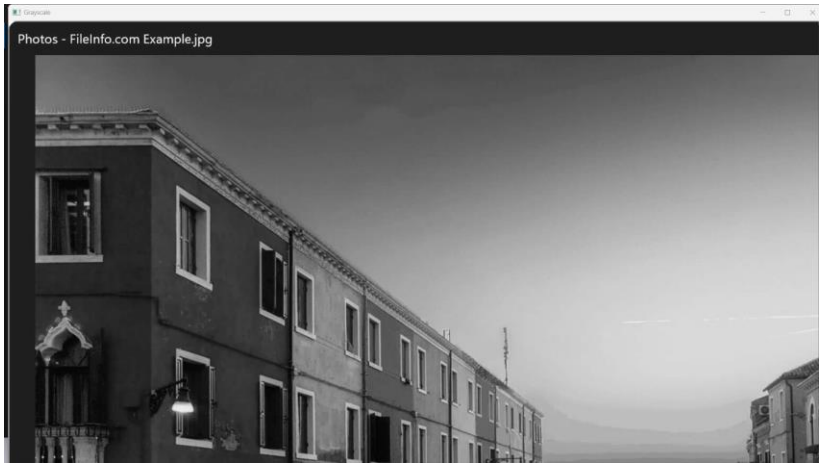
CODE 6:

```
import cv2
# Load an image
img = cv2.imread  (r"C:\Users\psadi\Downloads\jpg_44-2.jpg")
# Replace with your file path
# Check if image loaded correctly
if img is None:
    print("Error: Could not read image.")
    exit()
# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Show both images
cv2.imshow("Original", img)
cv2.imshow("Grayscale", gray)
# Wait until a key is pressed
cv2.waitKey(0)
cv2.destroyAllWindows()
# Optional: Save grayscale image
cv2.imwrite("grayscale_output.jpg", gray)
```

INPUt



OUTPUT:



Code Logic: Image grayscale using OpenCV

1. Imports the OpenCV library (cv2) for image processing.

2. Reads an image from the specified file path and stores it as a NumPy array.

3. Checks if the image was successfully loaded; prints an error message and exits if not.

4. Converts the color image to grayscale using cv2.cvtColor with cv2.COLOR_BGR2GRAY.

5. Displays both the original color image and the grayscale image in separate windows.

6. Waits indefinitely until a key is pressed.

7. Closes all display windows and releases OpenCV resources.

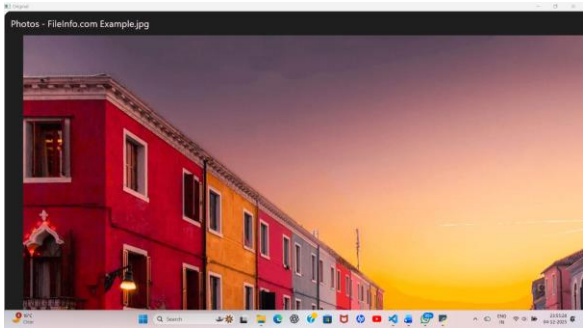
8. Optionally saves the grayscale image to disk using cv2.imwrite.

Purpose:
To demonstrate how to convert a color image to grayscale using OpenCV, which is useful for simplifying image analysis, reducing computational complexity, or preparing images for computer vision tasks such as edge detection or thresholding.

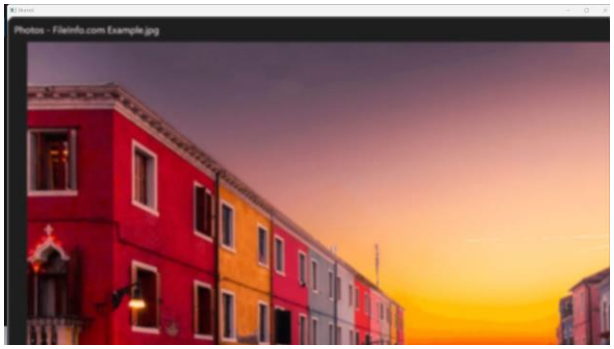
CODE 7:

```
import cv2
# Load an image
img = cv2.imread  (r"C:\Users\psadi\Downloads\jpg_44-2.jpg")
# Replace with your file path
# Check if image loaded correctly
if img is None:
    print("Error: Could not read image.")
    exit()
# Apply Gaussian Blur (15x15 kernel)
# Larger kernel size = stronger blur, smaller = softer blur
blur = cv2.GaussianBlur(img, (15, 15), 0)
# Show both images
cv2.imshow("Original", img)
cv2.imshow("Blurred", blur)
# Wait for key press
cv2.waitKey(0)
cv2.destroyAllWindows()
# Optional: Save the blurred image
cv2.imwrite("blurred_output.jpg", blur)
```

INPUT:



OUTPUT



Code Logic: GaussianBlur Image using OpenCV

1. Imports the OpenCV library (cv2) for image processing.

2. Reads an image from the specified file path and stores it as a NumPy array.

3. Checks if the image was successfully loaded; prints an error message and exits if not.

4. Applies Gaussian blur to the image using cv2.GaussianBlur with a kernel size of 15×15.

    *A larger kernel results in stronger blurring, while a smaller kernel produces a softer blur.

5. Displays both the original and blurred images in separate windows.

6. Waits indefinitely until a key is pressed.

7. Closes all display windows and releases OpenCV resources.

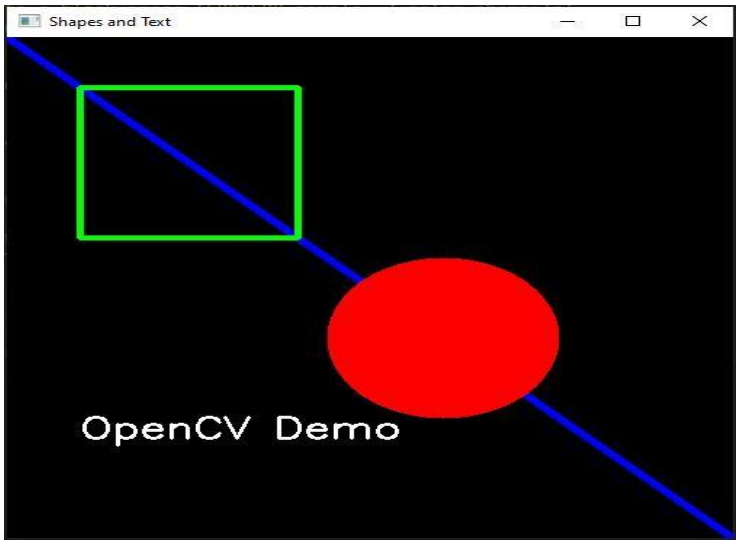
8. Optionally saves the blurred image to disk using cv2.imwrite.

Purpose:
To demonstrate how to apply Gaussian blur to an image using OpenCV, which is useful for noise reduction, smoothing, or preprocessing images for further computer vision tasks such as edge detection or segmentation.

CODE 8:

```
import cv2
import numpy as np
# Create a black image (500x500 pixels, 3 color channels)
img = np.zeros((500, 500, 3), dtype="uint8")
# Draw a blue line
cv2.line(img, (0, 0), (500, 500), (255, 0, 0), 5)
# Draw a green rectangle
cv2.rectangle(img, (50, 50), (200, 200), (0, 255, 0), 3)
# Draw a red filled circle
cv2.circle(img, (300, 300), 80, (0, 0, 255), -1)
# Add white text
cv2.putText(img, "OpenCV Demo", (50, 400), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 255, 255), 2)
# Display the image
cv2.imshow("Shapes and Text", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Code Logic:Draw shapes text using OpenCV

1. Imports the OpenCV library (cv2) for image processing and NumPy (np) for array manipulation.

2. Creates a black image of size 500×500 pixels with 3 color channels (RGB) using np.zeros.

3. Draws various shapes on the image:

   *A **blue line** from the top-left corner to the bottom-right corner with thickness 5.

   *A **green rectangle** from coordinates (50, 50) to (200, 200) with thickness 3.

   *A **red filled circle** centered at (300, 300) with radius 80 (-1 indicates filled).

4. Adds **white text** “OpenCV Demo” at position (50, 400) using cv2.putText.

5. Displays the resulting image in a window titled "Shapes and Text".

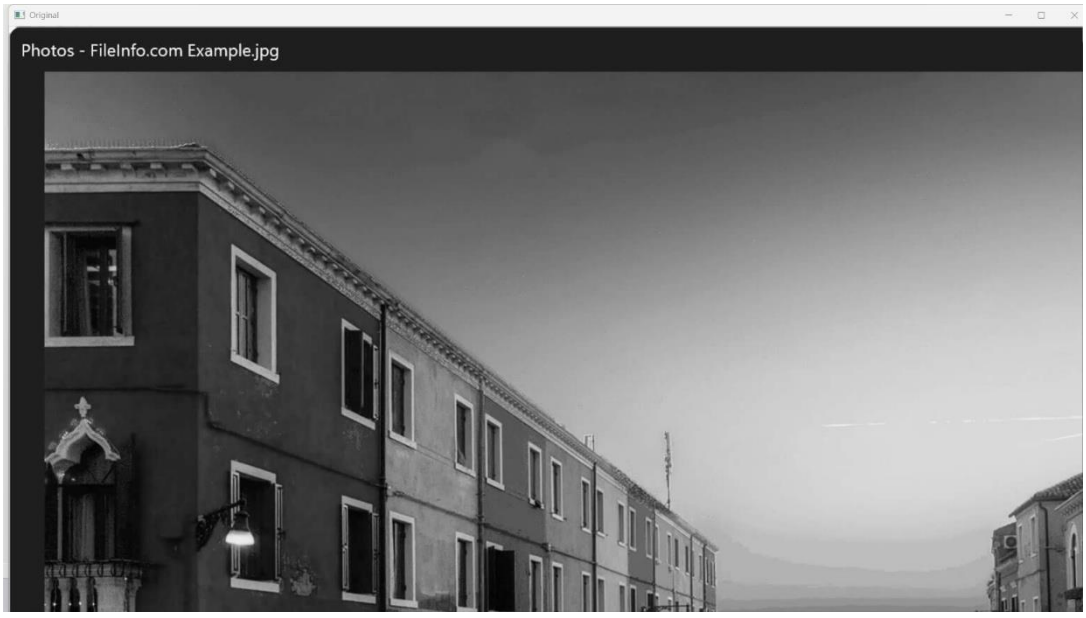
6. Waits indefinitely until a key is pressed.

Purpose:
To demonstrate how to create an image from scratch and draw basic shapes and text using OpenCV, which is useful for creating visual annotations, graphics, or custom illustrations for image processing projects.

CODE 9:

```
import cv2
# Load the image in grayscale mode
img = cv2.imread (r"C:\Users\psadi\Downloads\jpg_44-2.jpg", 0)
# Replace with your correct file path
# Apply thresholding
_, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
# Display the original and thresholded images
cv2.imshow("Original", img)
cv2.imshow("Thresholded", thresh)
# Wait for a key press and close windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

1INPUT:



OUTPUT:

Code Logic:binary thresholded Image using OpenCV

1. Imports the OpenCV library (cv2) for image processing.

2. Loads an image in **grayscale mode** by passing 0 as the second argument to cv2.imread.

3. Applies **binary thresholding** using cv2.threshold:

    *Pixels with intensity >127 are set to 255 (white).

    *Pixels with intensity ≤127 are set to 0 (black).

4. Displays both the original grayscale image and the thresholded binary image in separate windows.

5. Waits indefinitely until a key is pressed.

6. Closes all display windows and releases OpenCV resources.

Purpose:
To demonstrate how to convert a grayscale image into a binary image using thresholding, which is useful for separating foreground from background, simplifying image analysis, and preparing images for tasks like contour detection or segmentation.

CODE 10:

```
import cv2 img = cv2.imread (r"C:\Users\psadi\Downloads\jpg_44-2.jpg", 0)

edges = cv2.Canny(img, 100, 200)
cv2.imshow("Edges", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Logic:

1. Imports the OpenCV library (cv2) for image processing.

2. Loads an image in **grayscale mode** by passing 0 as the second argument to cv2.imread.

3. Applies the **Canny edge detection** algorithm using cv2.Canny with thresholds 100 and 200:

    *Detects edges in the image by finding areas with strong intensity gradients.

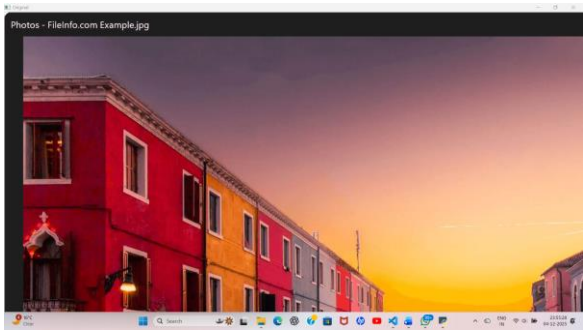
4. Displays the resulting edge-detected image in a window titled "Edges".

5. Waits indefinitely until a key is pressed.

6. Closes all display windows and releases OpenCV resources.

Purpose:
To demonstrate how to detect edges in an image using OpenCV, which is useful for feature extraction, object detection, and simplifying image analysis for computer vision tasks.

INPUT:



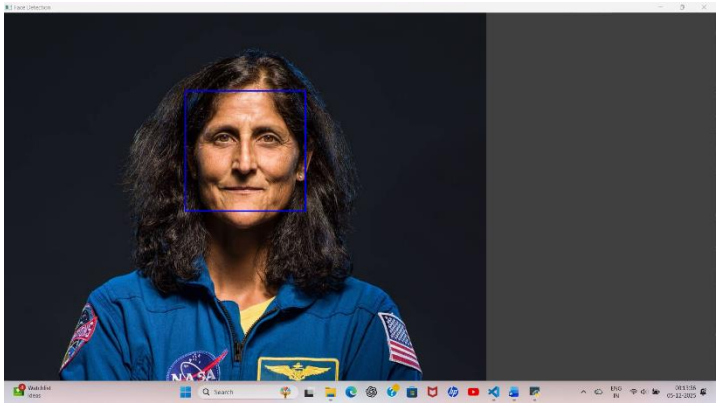
OUTPUT:

CODE 11:

```
import cv2
# Load pre-trained classifier
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
                                     "haarcascade_frontalface_default.xml")
# Read the image (use absolute path)
img = cv2.imread(r"C:\Users\HP\OneDrive\Desktop\infosys
internship\faceimage1.webp")
if img is None:
    print("Error: Could not read image. Check the file path and name.")
    exit()
# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Detect faces
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
# Draw rectangles around faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
# Display the image
cv2.imshow("Face Detection", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:



Logic:

1. Imports the OpenCV library (cv2) for image processing.

2. Loads a **pre-trained Haar Cascade classifier** for frontal face detection using cv2.CascadeClassifier.

3. Reads the image from the specified file path and checks if it was loaded successfully.

4. Converts the color image to **grayscale** using cv2.cvtColor because Haar cascades work on grayscale images.

5. Detects faces in the grayscale image using detectMultiScale:

    *1.1 is the scale factor for image resizing during detection.

    *4 is the minimum number of neighbors to retain a detection.

6. Draws **rectangles** around all detected faces on the original image.

7. Displays the resulting image with detected faces in a window titled "Face Detection".

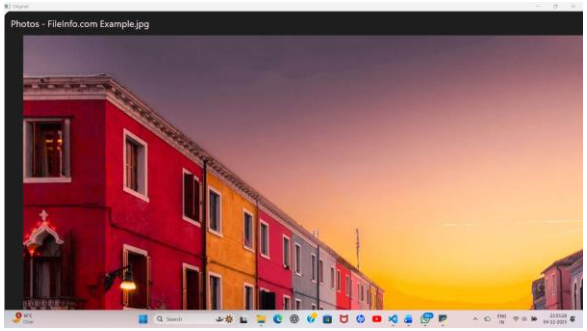
8. Waits indefinitely until a key is pressed.

Purpose:
To demonstrate face detection in images using OpenCV's Haar Cascade classifier, which is useful for applications like security systems, attendance tracking, or preprocessing images for facial recognition.

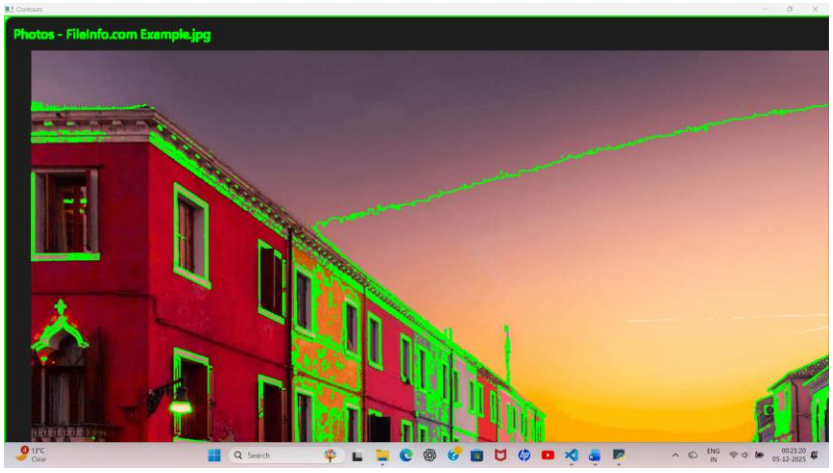
CODE 12:

```
import cv2
img = cv2.imread(r"C:\Users\psadi\Downloads\jpg_44-2.jpg")

if img is None:
    print("Error: Could not read image. Check the file path and name.")
    exit()
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0, 255, 0), 2)
cv2.imshow("Contours", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:



OUTPUT:



Logic:

1. Imports the OpenCV library (cv2) for image processing.
2. Reads the image from the specified file path and checks if it was loaded successfully.
3. Converts the image to **grayscale** using cv2.cvtColor.
4. Applies **binary thresholding** using cv2.threshold to separate shapes from the background.
5. Finds **contours** in the thresholded image using cv2.findContours:

    *cv2.RETR_TREE retrieves all contours and reconstructs the hierarchy.

    *cv2.CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments into endpoints.

6. Draws all detected contours on the original image using cv2.drawContours with green color and thickness 2.

7. Displays the resulting image with contours in a window titled "Contours".

8. Waits indefinitely until a key is pressed.

9. Closes all display windows and releases OpenCV resources.

Purpose:

The program reads an image, converts it to grayscale, thresholds it to create a binary version, detects object boundaries (contours), and displays the image with those contours highlighted in green.This technique is commonly used in shape detection, object recognition, and image analysis tasks.

CODE 13:

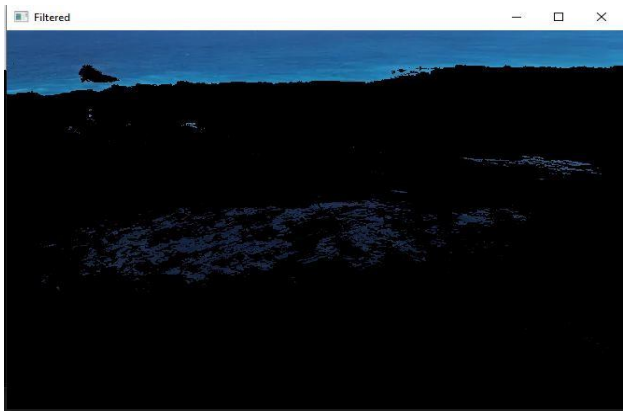
```
import cv2
# Read the image
Img=cv2.imread(r"C:\Users\psadi\Downloads\image.png")


# Check if image loaded properly
if img is None:
    print("Error: Could not read image. Check the file path and name.")
    exit()
# Convert to HSV color space
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# Define blue color range
lower_blue = (100, 150, 0)
upper_blue = (140, 255, 255)
# Create a mask for blue color
mask = cv2.inRange(hsv, lower_blue, upper_blue)
# Apply mask on original image
result = cv2.bitwise_and(img, img, mask=mask)
# Display results
cv2.imshow("Original", img)
cv2.imshow("Mask", mask)
cv2.imshow("Filtered", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

MASK:

FILTERED:



Logic:

1. Imports the OpenCV library (cv2) for image processing.

2. Reads the image from the specified file path and checks if it was loaded successfully.

3. Converts the image from **BGR to HSV color space** using cv2.cvtColor because HSV simplifies color-based segmentation.

4. Defines the **blue color range** with lower_blue and upper_blue values.

5. Creates a **mask** using cv2.inRange to isolate pixels within the blue range.

6. Applies the mask on the original image using cv2.bitwise_and to extract only the blue regions.

7. Displays:

    *The original image

*The mask showing detected blue areas

*The result image showing only blue regions filtered from the original

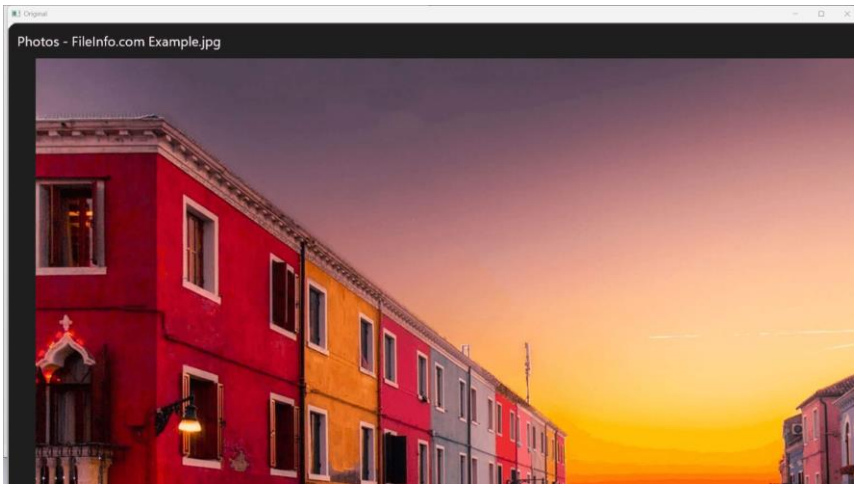
Purpose:

The program reads an image, converts it to HSV color space, isolates the blue color range, and displays only the blue-colored regions.This technique is widely used in **object tracking, color-based segmentation, and robotics** (for example, tracking a colored ball).

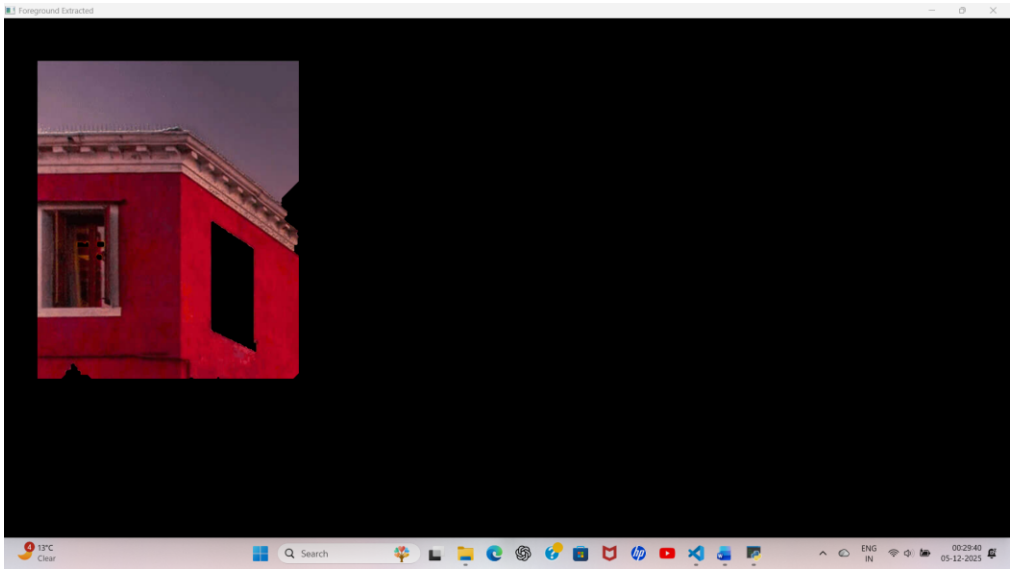
CODE 14:

```
import cv2
import numpy as np
img = cv2.imread(r"C:\Users\HP\OneDrive\Desktop\infosys
internship\oilspill5.webp")
if img is None:
    print("Error: Could not read image. Check the file path and name.")exit()
# Create mask and models
mask = np.zeros(img.shape[:2], np.uint8)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
# Define rectangle for the object (adjust coordinates if needed)
rect = (50, 50, 400, 500)
# Apply GrabCut algorithm
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)
# Create final mask
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
# Extract the foreground
result = img * mask2[:, :, np.newaxis]
cv2.imshow("Original", img)
cv2.imshow("Foreground Extracted", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:



OUTPUT:



Logic:

This code extracts the **foreground object (like a person)** from an image using OpenCV's **GrabCut algorithm**, which separates the foreground and background automatically.

1.cv2.imread("person.jpg")
Reads the image named person.jpg in color mode and stores it in the variable img.

2.mask = np.zeros(img.shape[:2], np.uint8)
Creates a **mask** (same height and width as the image) initialized with zeros.

*This mask helps OpenCV know which pixels belong to the foreground and background.

3.bgdModel = np.zeros((1, 65), np.float64)
Creates a temporary array used internally by the algorithm to model the **background**.

4.fgdModel = np.zeros((1, 65), np.float64)
Creates a temporary array used to model the **foreground**.

5.rect = (50, 50, 400, 500)
Defines a rectangular **Region of Interest (ROI)** that roughly contains the object (the person).

*(x, y, width, height) → top-left corner and size of rectangle.

6.cv2.grabCut(img, mask, rect, bgdModel, fgdModel, cv2.GC_INIT_WITH_RECT)
Applies the **GrabCut algorithm**:

*It uses the rectangle to identify probable background and foreground regions.

*Iterates 5 times to refine the segmentation.

*Updates the mask, background, and foreground models automatically.

7.mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype("uint8")
Converts the multi-class mask to a binary mask:

*Pixels labeled as background (0 or 2) → set to 0.

*Pixels labeled as foreground (1 or 3) → set to 1.

8.result = img * mask2[:, :, np.newaxis]
Multiplies the original image with the binary mask to **keep only the foreground** (the person) and remove the background.

9.cv2.imshow("Original", img) Displays the original input image.

10.cv2.imshow("Foreground Extracted", result)
Displays the output image where the background is removed and only the person is visible

Purpose:

The program uses OpenCV's **GrabCut algorithm** to automatically segment and extract the main object (foreground) from an image.This is useful in applications like **background removal, image editing, portrait isolation, and object recognition**.

CODE 15:

```
import cv2
# Open webcam (0 = default camera)
cap = cv2.VideoCapture(0)
while True:
   ret, frame = cap.read()
   if not ret:
      print("Error: Unable to access the webcam.")
      break
   # Convert frame to HSV color space
   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
   # Define blue color range
   lower_blue = (100, 150, 0)
   upper_blue = (140, 255, 255)
   # Create mask for blue color
   mask = cv2.inRange(hsv, lower_blue, upper_blue)
   # Apply mask to original frame
   result = cv2.bitwise_and(frame, frame, mask=mask)
   # Display windows
```

```
    cv2.imshow("Frame", frame)
    cv2.imshow("Mask", mask)
    cv2.imshow("Tracked", result)
    # Press 's' to save the current frame
    if cv2.waitKey(1) & 0xFF == ord('s'):
        cv2.imwrite("captured_frame.jpg", frame)
        print("    Image saved as 'captured_frame.jpg'")
    # Press 'q' to quit
    elif cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release resources
cap.release()
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

LOGIC

1. Opens your **webcam** and reads frames continuously.

2. Converts each frame from **BGR to HSV** color space.

3. Creates a **mask** to isolate blue regions.

4. Displays:

    *Frame → Original camera feed

    *Mask → White areas where blue is detected

    *Tracked → Only blue areas shown

5. Press **'s'** → Saves the current frame as captured_frame.jpg

6. Press **'q'** → Exits the window safely

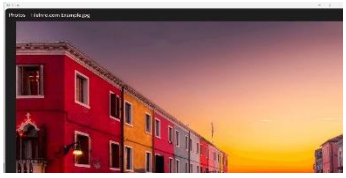
Purpose

This program detects and tracks blue-colored objects in real-time using your webcam. You can capture and save images during the live stream — useful for object tracking, gesture recognition, or color-based control systems.

CODE 15:

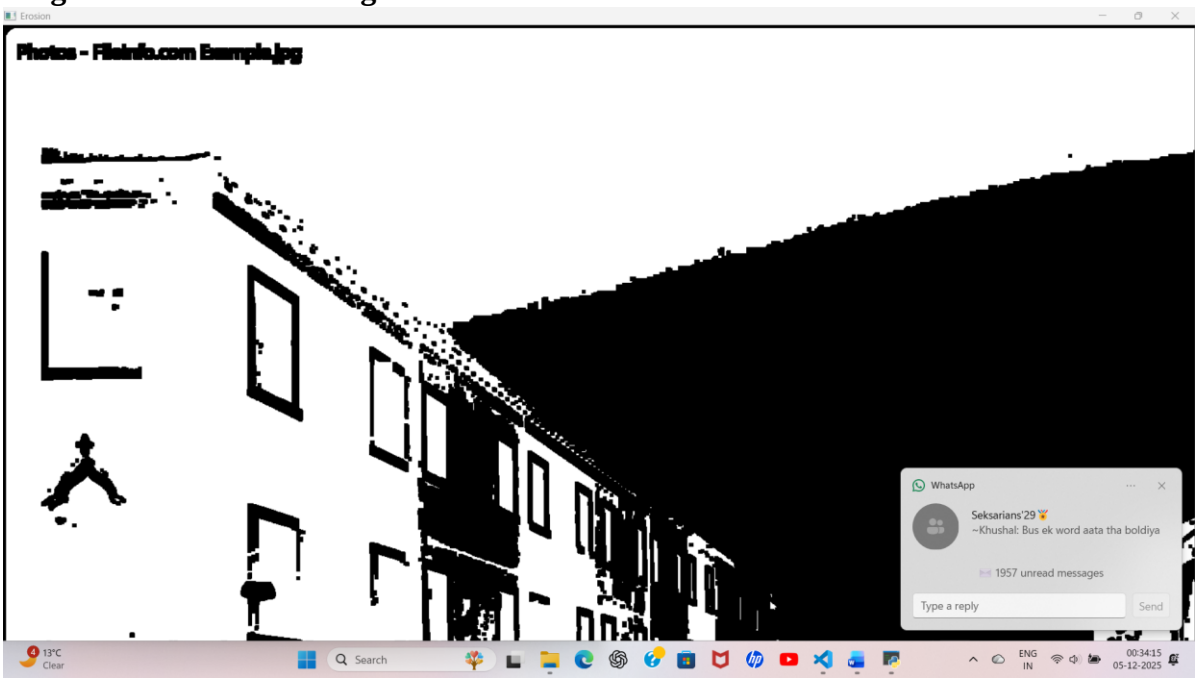
```
import cv2
import numpy as np
# Step 1: Read the image in grayscale
img =
cv2.imread(r"C:\Users\HP\OneDrive\Desktop\infosysinternship\oilspill5jpg.jpg", 0)
# Replace with your image path
if img is None:
    print("Error: Could not read image.")
    exit()
# Step 2: Apply binary inverse thresholding
_, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
# Step 3: Define a 5x5 kernel for morphological operations
kernel = np.ones((5, 5), np.uint8)
# Step 4: Apply erosion
erosion = cv2.erode(thresh, kernel, iterations=1)
# Step 5: Apply dilation
dilation = cv2.dilate(thresh, kernel, iterations=1)
# Step 6: Display results
cv2.imshow("Original Thresholded Image", thresh)
cv2.imshow("Erosion", erosion)
cv2.imshow("Dilation", dilation)
# Step 7: Wait for a key press and close windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:



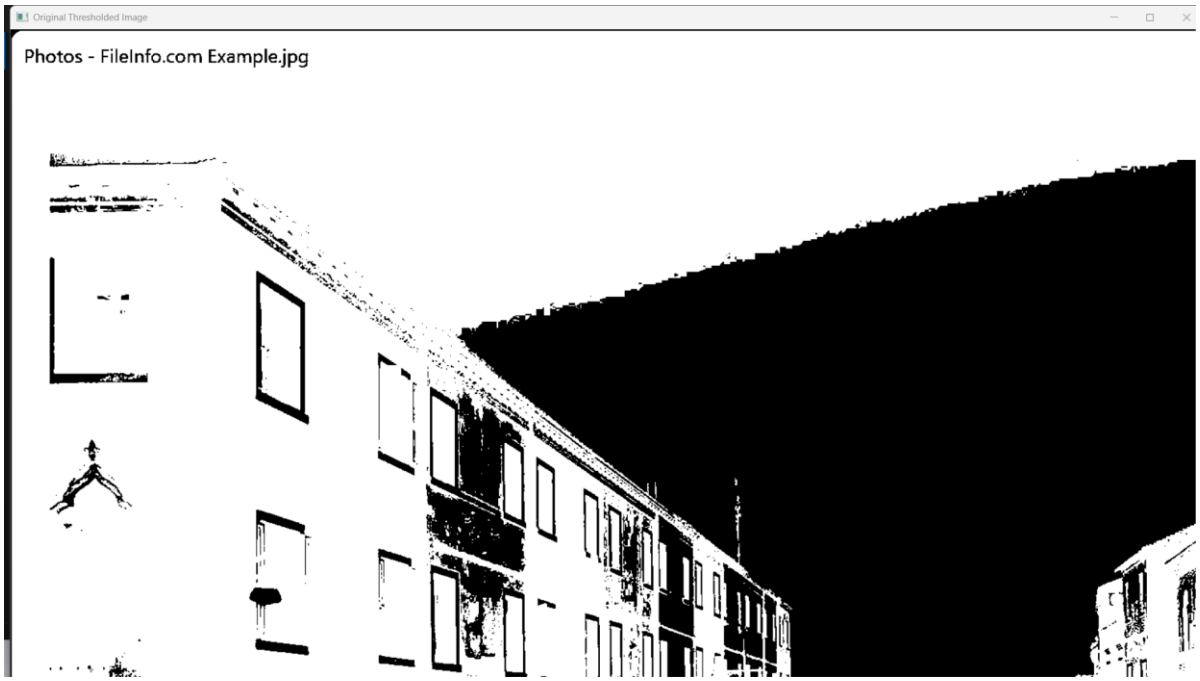
OUTPUT:

Original Thresholded Image



Erosion

Dilation



Logic:

This code demonstrates **morphological operations** (Erosion and Dilation) in image processing using OpenCV. These operations are commonly used for noise removal, shape refinement, and feature extraction from binary images.

1. cv2.imread("text.png", 0) Reads the image named text.png in **grayscale mode**. The argument 0 converts it into a single-channel image for easier binary processing.

2. cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
Converts the grayscale image into a **binary image** using thresholding.

*Pixel values above 127 become 0 (black).

*Pixel values below 127 become 255 (white), because of THRESH_BINARY_INV.This inverts the colors so that text or objects appear white on a black background.

3.kernel = np.ones((5, 5), np.uint8)
Creates a **5×5 matrix (kernel)** filled with ones, used for morphological operations.

*The kernel defines the area size over which erosion or dilation acts.

4. cv2.erode(thresh, kernel, iterations=1) Performs **erosion** on the binary image.

*Erosion **shrinks** the white regions (foreground).

*Useful for removing small white noise or separating connected objects.

5. cv2.dilate(thresh, kernel, iterations=1) Performs **dilation** on the binary image.

*Dilation **expands** the white regions.

*Useful for filling small holes or connecting broken parts of objects.

6. cv2.imshow("Original", thresh) Displays the binary thresholded image.

7. cv2.imshow("Erosion", erosion) Displays the result after applying erosion (thinner text or objects).

8. cv2.imshow("Dilation", dilation) Displays the result after applying dilation (thicker text or objects).

Purpose:

The program reads an image, converts it into a binary form, and demonstrates **Erosion** and **Dilation**—two key morphological operations.These techniques are useful for tasks such as **noise reduction, boundary detection, and preparing images for OCR (Optical Character Recognition)**.

CODE 17:

```
import re
from typing import List
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
import spacy
nlp = spacy.load("en_core_web_sm") # small, fast English model
def basic_clean(text: str) -> str:
 # lower, strip urls/emails/@mentions/hashtags, keep
letters/numbers/space/apostrophe
 text = text.lower()
 text = re.sub(r"(http\S+|www\.\S+)", " ", text)
 text = re.sub(r"\S+@\S+", " ", text)
 text = re.sub(r"[@#]\w+", " ", text)
 text = re.sub(r"[^a-z0-9\s']", " ", text)
 text = re.sub(r"\s+", " ", text).strip()
 return text
def tokenize_stop_lemma(text: str) -> List[str]:
 doc = nlp(text)
 out = []
 for tok in doc:
 if tok.is_space or tok.is_punct:
 continue
 lemma = tok.lemma_.lower().strip()
 if len(lemma) < 3: # drop very short tokens
 continue
 if lemma in ENGLISH_STOP_WORDS: # sklearn's built-in stoplist
 continue
 out.append(lemma)
 return out
def preprocess(text: str) -> List[str]:
```

```
return tokenize_stop_lemma(basic_clean(text))
if __name___== "__main__":
 s = "Emails like help@site.com are filtered. I’m LOVING NLP!!! Visit https://x.y."
 print(preprocess(s))
```

INPUT:
s = "Emails like help@site.com are filtered. I’m LOVING NLP!!! Visit “https://x.y."

OUTPUT:

```
['email', 'like', 'filter', 'love', 'nlp', 'visit']
```

Logic:

1. basic_clean() → Converts text to lowercase, removes URLs, emails, mentions, hashtags,and special characters, keeping only letters, numbers, spaces, and apostrophes.

2. tokenize_stop_lemma() →Uses spaCy to tokenize the cleaned text.
Converts words to their lemmas (base forms).Removes stop words, punctuation, and short tokens (less than 3 characters).

3. preprocess() → Combines cleaning and tokenization steps to return a final list of meaningful words.

4. When run, it processes a sample sentence and prints the cleaned, lemmatized word list.Purpose: To prepare raw text for NLP or machine learning by cleaning, tokenizing, lemmatizing, and removing unnecessary words**.**

CODE 18:

```
# file: 2_classify_tfidf.py
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
# Tiny demo dataset (positive/negative sentiment)
texts = [
    "I loved this movie, fantastic acting and great story",
    "This film was terrible and boring",
    "Absolutely wonderful experience, highly recommend",
    "Worst acting ever, do not watch",
    "It was okay, some parts were fun",
    "I hated the plot, very disappointing",
    "Brilliant direction and superb cast",
    "Not good, waste of time",
    "Enjoyable and engaging from start to finish",
    "Awful soundtrack and weak story"
```

```
]
labels = np.array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0])  # 1=positive, 0=negative
# Step 1: Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    texts, labels, test_size=0.3, random_state=42, stratify=labels
)
# Step 2: Create a pipeline with TF-IDF and Logistic Regression
pipe = Pipeline([
    ("tfidf", TfidfVectorizer(ngram_range=(1, 2), min_df=1)),
    ("clf", LogisticRegression(max_iter=1000))
])
# Step 3: Train the model
pipe.fit(X_train, y_train)

# Step 4: Evaluate the model
y_pred = pipe.predict(X_test)
print("Classification report:\n", classification_report(y_test, y_pred, digits=4))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))

# Step 5: Try predictions on new samples
samples = ["pretty good but slow in places", "utterly awful, I want my time back"]
print("Predictions:", pipe.predict(samples))
print("Class probabilities:\n", pipe.predict_proba(samples))
```

LOGIC:

1. Creates a small dataset of **movie reviews** labeled as **positive (1)** or **negative (0)**.

2. **Splits** the dataset into **training (70%)** and **testing (30%)** sets to evaluate model performance.

3. Builds a **Pipeline** that includes:

    ***TfidfVectorizer** → Converts text data into numerical features based on **word importance (TF-IDF)**.

    ***LogisticRegression** → Learns patterns to classify reviews as positive or negative.

4. **Trains (fit)** the pipeline on the training data so the model learns from the examples.

5. **Predicts** sentiment labels for the test data and prints:

    ***Classification report** → Shows precision, recall, F1-score, and accuracy.

    ***Confusion matrix** → Displays counts of correct and incorrect predictions.

6. **Tests** the model on new unseen sample reviews and prints:

***Predicted sentiment** (positive or negative).

***Class probabilities** (confidence of predictions)

Purpose:

To **automatically classify text reviews as positive or negative** using **machine learning**, demonstrating a simple **text sentiment analysis** pipeline based on **TF-IDF feature extraction** and **Logistic Regression classification**.

OUTPUT:

```
Classification report:
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000         2
           1     0.3333    1.0000    0.5000         1

    accuracy                         0.3333         3
   macro avg     0.1667    0.5000    0.2500         3
weighted avg     0.1111    0.3333    0.1667         3

Confusion matrix:
 [[0 2]
 [0 1]]
Predictions: [1 1]
Class probabilities:
 [[0.42652933 0.57347067]
 [0.46719711 0.53280289]]
```

CODE 19:

```
# file: 3_tune_grid.py
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
import numpy as np
texts = [
"excellent movie with great acting",
"terrible plot and awful pacing",
"loved every moment, fantastic!",
"boring and predictable",
"superb cinematography and direction",
"weak script and bad acting",
"what a masterpiece",
"not good at all",
"brilliant experience overall",
"do not recommend"]
y = np.array([1,0,1,0,1,0,1,0,1,0])
```

```
pipe = Pipeline([
("tfidf", TfidfVectorizer()),
("clf", LogisticRegression(max_iter=1000))])
param_grid = {
"tfidf__ngram_range": [(1,1),(1,2)],
"tfidf__min_df": [1,2],
"tfidf__analyzer": ["word", "char_wb"],
"clf__C": [0.25, 1.0, 4.0] # regularization strength
}
search = GridSearchCV(pipe, param_grid, cv=3, n_jobs=-1, scoring="f1")
search.fit(texts, y)
print("Best params:", search.best_params_)
print("Best CV score (f1):", search.best_score_)
best_model = search.best_estimator_
print("Sample prediction:", best_model.predict(["not a great movie but had
  moments"]))
```

OUTPUT:

```
Best params: {'clf__C': 4.0, 'tfidf__analyzer': 'char_wb', 'tfidf__min_df': 1, 'tfidf__ngram_range': (1, 2)}
Best CV score (f1): 0.7222222222222222
Sample prediction: [1]
```

Logic:

1. **Creates a small dataset** of short movie reviews labeled as **positive (1)** or **negative (0)**.

2. Builds a **Pipeline** consisting of:

   ***TfidfVectorizer** → Converts text into numerical TF-IDF feature vectors.

   ***LogisticRegression** → Classifies the reviews as positive or negative.

3. Defines a **parameter grid (**param_grid**)** to test multiple settings:

   *tfidf__ngram_range → Uses single words (1,1) or word pairs (1,2).

   *tfidf__min_df → Ignores rare words appearing fewer than 1 or 2 times.

   *tfidf__analyzer → Chooses between word-level or character-level analysis.

   *clf__C → Adjusts model regularization (controls overfitting).

4. Uses **GridSearchCV** with **3-fold cross-validation (cv=3)** to automatically test all parameter combinations and find the best configuration.

5. **Fits (**search.fit**)** the grid search on the dataset, evaluating each model using **F1-score** as the performance metric.

6. **Prints** the following outputs:

***Best parameters** found by the grid search.

***Best cross-validation F1-score** achieved.

***Sample prediction** from the best-performing model on a new review.

Purpose:

To **optimize model performance** by automatically tuning **TF-IDF** and **Logistic Regression** parameters using **Grid Search**.This ensures the classifier achieves the **best possible accuracy and generalization** for text sentiment classification tasks.

CODE 20:

```
# file: 4_tfidf_demo.py
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
# Sample documents
docs = [
"machine learning is fun",
"deep learning advances machine intelligence",
"artificial intelligence and machine learning"
]
# Create TF-IDF model
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(docs)
# Convert to DataFrame
tfidf_df = pd.DataFrame(X.toarray(), columns=tfidf.get_feature_names_out())
print("Vocabulary:", tfidf.get_feature_names_out())
print("\nTF-IDF Matrix:")
print(tfidf_df.round(3))
```

OUTPUT:

```
Vocabulary: ['advances' 'and' 'artificial' 'deep' 'fun' 'intelligence' 'is' 'learning'
 'machine']

TF-IDF Matrix:
   advances    and  artificial   deep    fun  intelligence     is  learning  \
0     0.000  0.000       0.000  0.000  0.609          0.00  0.609     0.360
1     0.552  0.000       0.000  0.552  0.000          0.42  0.000     0.326
2     0.000  0.552       0.552  0.000  0.000          0.42  0.000     0.326

   machine
0    0.360
1    0.326
2    0.326
```

LOGIC:

1. **Creates a small list of text documents** related to machine learning and artificial intelligence.

2. Initializes the TfidfVectorizer() from scikit-learn to convert text into numerical vectors based on **TF-IDF (Term Frequency–Inverse Document Frequency)**.

3. **Fits and transforms** the documents using fit_transform() to produce a **TF-IDF matrix**, where each row represents a document and each column represents a word.

4. Converts the resulting **sparse matrix** into a **pandas DataFrame** using pd.DataFrame() for easier viewing and analysis.

5. **Displays**:

   *The **vocabulary** (unique words extracted from all documents).

   *The **TF-IDF matrix**, showing the importance scores of each word in each document.

Purpose:

To **demonstrate how TF-IDF converts text into numerical features**, capturing the **importance of each word** relative to a document and the entire collection.
This forms the **foundation for text mining, NLP, and machine learning tasks**, such as classification, clustering, and keyword extraction.

CODE 21:

```
# file: 5_spacy_ner_pos.py
import spacy
from pprint import pprint
nlp = spacy.load("en_core_web_sm")
text = ("Apple is opening a new office in Bengaluru next quarter. "
"Tim Cook met Karnataka officials on September 3, 2025 to discuss expansion.")
doc = nlp(text)
print("\nNamed Entities (text, label):")
for ent in doc.ents:
    print(f"{ent.text:<25} -> {ent.label_}")
print("\nPart-of-Speech & Lemmas:")
for token in doc:
    if not token.is_space:
        print(f"{token.text:<15} POS={token.pos_:<5} Lemma={token.lemma_}")
```

```
print("\nNoun chunks (base NPs):")
pprint([chunk.text for chunk in doc.noun_chunks])
```

OUTPUT:

```
Named Entities (text, label):
Apple                     -> ORG
Bengaluru                 -> GPE
next quarter              -> DATE
Tim Cook                  -> PERSON
Karnataka                 -> GPE
September 3, 2025         -> DATE

Part-of-Speech & Lemmas:
Apple           POS=PROPN Lemma=Apple
is              POS=AUX   Lemma=be
opening         POS=VERB  Lemma=open
a               POS=DET   Lemma=a
new             POS=ADJ   Lemma=new
office          POS=NOUN  Lemma=office
in              POS=ADP   Lemma=in
Bengaluru       POS=PROPN Lemma=Bengaluru
next            POS=ADJ   Lemma=next
quarter         POS=NOUN  Lemma=quarter
.               POS=PUNCT Lemma=.
Tim             POS=PROPN Lemma=Tim
Cook            POS=PROPN Lemma=Cook
met             POS=VERB  Lemma=meet
Karnataka       POS=PROPN Lemma=Karnataka
officials       POS=NOUN  Lemma=official
on              POS=ADP   Lemma=on
September       POS=PROPN Lemma=September
3               POS=NUM   Lemma=3
,               POS=PUNCT Lemma=,
2025            POS=NUM   Lemma=2025
to              POS=PART  Lemma=to
discuss         POS=VERB  Lemma=discuss
expansion       POS=NOUN  Lemma=expansion
.               POS=PUNCT Lemma=.

Noun chunks (base NPs):
['Apple',
 'a new office',
 'Bengaluru',
 'Tim Cook',
 'Karnataka officials',
 'September',
 'expansion']
```

Logic:

1. **Imports the SpaCy library** and loads the **English language model** en_core_web_sm.

2. Defines a **sample text** mentioning *Apple*, *Bengaluru*, *Tim Cook*, and *Karnataka* for NLP processing.

3. Passes the text through the SpaCy model using nlp(text) to create a **Doc object**, which stores all linguistic annotations (tokens, entities, POS tags, etc.).

4. **Named Entity Recognition (NER):**

*Iterates through doc.ents to extract **named entities** such as organizations, people, dates, and locations.

*Prints each entity along with its **label** (e.g., ORG, GPE, DATE).

5. **Part-of-Speech (POS) Tagging and Lemmatization:**

Iterates through each token in doc to display:

*The **word (token)**

*Its **part of speech (POS)** (e.g., NOUN, VERB, PROPN)

*Its **lemma** (base form of the word).

6. **Noun Chunk Extraction:**

*Uses doc.noun_chunks to extract **noun phrases** (base noun groups like “a new office” or “Karnataka officials”).

*Prints the noun chunks using pprint() for better readability.

Purpose:

1. To demonstrate **Natural Language Processing (NLP)** capabilities using **SpaCy**, including:

Named Entity Recognition (NER) for identifying people, places, organizations, and dates.

Part-of-Speech (POS) tagging and **Lemmatization** for grammatical and linguistic analysis.

Noun phrase extraction to identify key subjects or objects in text.

2. This program showcases how SpaCy can analyze text structure and meaning a key step in building intelligent **language understanding systems**.

CODE 22:

```
# file: 6_nb_classify.py
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
# Tiny sentiment dataset
texts = [
"I love this movie",
"This film was awful",
"Amazing performance and great story",
"Boring and too long",
"Fantastic acting",
```

```
"Terrible direction"]
labels = [1, 0, 1, 0, 1, 0] # 1=positive, 0=negative
# Split data
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3,
random_state=42)
# Bag of Words + Naive Bayes
vectorizer = CountVectorizer()
X_train_bow = vectorizer.fit_transform(X_train)
X_test_bow = vectorizer.transform(X_test)
clf = MultinomialNB()
clf.fit(X_train_bow, y_train)
y_pred = clf.predict(X_test_bow)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:

```
Classification Report:
                precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

Logic:

1. **Creates a small sentiment dataset** containing short movie reviews labeled as **positive (1)** or **negative (0)**.

Splits the dataset into **training (70%)** and **testing (30%)** sets using train_test_split() to evaluate model performance on unseen data.

Initializes a **CountVectorizer** to convert text into a **Bag-of-Words (BoW)** representation — each word becomes a feature, and its frequency in each review is counted.

Transforms the training and testing data using fit_transform() and transform() to create BoW feature matrices.

Initializes a **Multinomial Naive Bayes (MultinomialNB)** classifier, which is well-suited for text classification tasks involving word counts.

Trains (fits) the Naive Bayes model on the training data (X_train_bow, y_train).

Predicts sentiment labels for the test data and prints a **classification report**, which includes metrics such as **precision, recall, F1-score, and accuracy**.

Purpose:

To demonstrate **text classification using the Naive Bayes algorithm** on a simple movie review dataset.
This program shows how to:

Convert raw text into numerical features using **Bag-of-Words**, and

Use **Multinomial Naive Bayes** to automatically classify reviews as **positive or negative**, providing a basic yet effective approach for **sentiment analysis**.

CODE 23:

```
# file: 7_similarity_demo.py
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
# Example documents
docs = [
"I love machine learning and NLP",
"NLP and machine learning are amazing",
"Cooking recipes are fun to try",
]
# TF-IDF representation
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(docs)
# Compute cosine similarity
sim_matrix = cosine_similarity(X)
print("Cosine Similarity Matrix:\n", sim_matrix)
```

OUTPUT:

```
Cosine Similarity Matrix:
 [[1.         0.64424596 0.        ]
 [0.64424596 1.         0.12413287]
 [0.         0.12413287 1.        ]]
```

Logic:

1. **Creates a small set of example text documents**, two related to *machine learning/NLP* and one about *cooking*.

2. Initializes a **TfidfVectorizer()** to convert the text documents into **TF-IDF feature vectors**, capturing the importance of each word in relation to all documents.

3. Uses fit_transform() to produce the **TF-IDF matrix**, where each document is represented numerically by its term weights.

4. Calculates the **cosine similarity** between every pair of documents using cosine_similarity(X).

5. Prints the **cosine similarity matrix**, where:

*Values close to **1** indicate high similarity between documents.

*Values close to **0** indicate low or no similarity.

Purpose:

To demonstrate how to measure **text similarity** using **TF-IDF vectors** and **cosine similarity.**This technique helps in identifying how closely two documents are related in meaning a foundational step in applications such as **document clustering**, **recommendation systems**, and **semantic search engines**.

CODE 24:

```
# file: 8_topic_modeling.py
from gensim import corpora, models
# Example dataset
docs = [
"I love deep learning and natural language processing",
"Artificial intelligence is the future",
"Cooking and baking are my hobbies",
"I enjoy trying new recipes in the kitchen",
"Machine learning and AI are closely related"]
# Tokenize
texts = [doc.lower().split() for doc in docs]
# Create dictionary & corpus
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
# Train LDA model (2 topics)
lda_model = models.LdaModel(corpus, num_topics=2, id2word=dictionary,
  passes=10)
# Show topics
for idx, topic in lda_model.print_topics(-1):
    print(f"Topic {idx}: {topic}")
```

OUTPUT:

```
Topic 0: 0.100*"and" + 0.071*"are" + 0.071*"learning" + 0.043*"closely" + 0.043*"ai" + 0.043*"machine" + 0.043*"related" + 0.043*"processing" + 0.043*"hobbies" + 0.043*"natural"
Topic 1: 0.092*"the" + 0.056*"i" + 0.055*"new" + 0.055*"recipes" + 0.055*"in" + 0.055*"enjoy" + 0.055*"trying" + 0.055*"kitchen" + 0.055*"artificial" + 0.055*"future"
```

LOGIC:

1. **Install the required library** using the command:

```
%pip install gensim
```

2. This installs the Gensim library, which is used for topic modeling and text analysis.

3. **Creates a small dataset** containing text documents from two different themes *AI/ML* and *cooking*.

4. **Tokenizes** each document by converting it to lowercase and splitting it into individual words using split().

5. Builds a **dictionary** using corpora.Dictionary(texts), which assigns a unique ID to every distinct word in the dataset.

6. Converts each document into a **Bag-of-Words (BoW)** representation using doc2bow(), producing a list of (word ID, word frequency) pairs to form the **corpus**.

7. Trains a **Latent Dirichlet Allocation (LDA)** model using models.LdaModel() with the following parameters:

    *num_topics=2 → identifies two main topics in the dataset.

    *passes=10 → iterates 10 times over the corpus for better topic estimation.

    *id2word=dictionary → maps word IDs back to actual words for readability.

Prints the discovered topics using print_topics(), displaying the top words contributing to each topic and their relative importance.

Purpose:

To demonstrate how to perform **Topic Modeling** using **Latent Dirichlet Allocation (LDA)** with the **Gensim** library.This program helps **automatically discover hidden themes** within a collection of text documents by grouping related words into coherent topics.It is widely used in **text mining, document organization, and content analysis** to understand large text datasets without manual labeling.