Programming for Artificial Intelligence

SUPERIOR UNIVERSITY

Name:

Mahak Farhan

Roll no.:

068

Class:

BSAI

Section:

4B

Subject:

Pogramming for Artificial Intelligence

Submitted to:

Sir Rasikh Ali

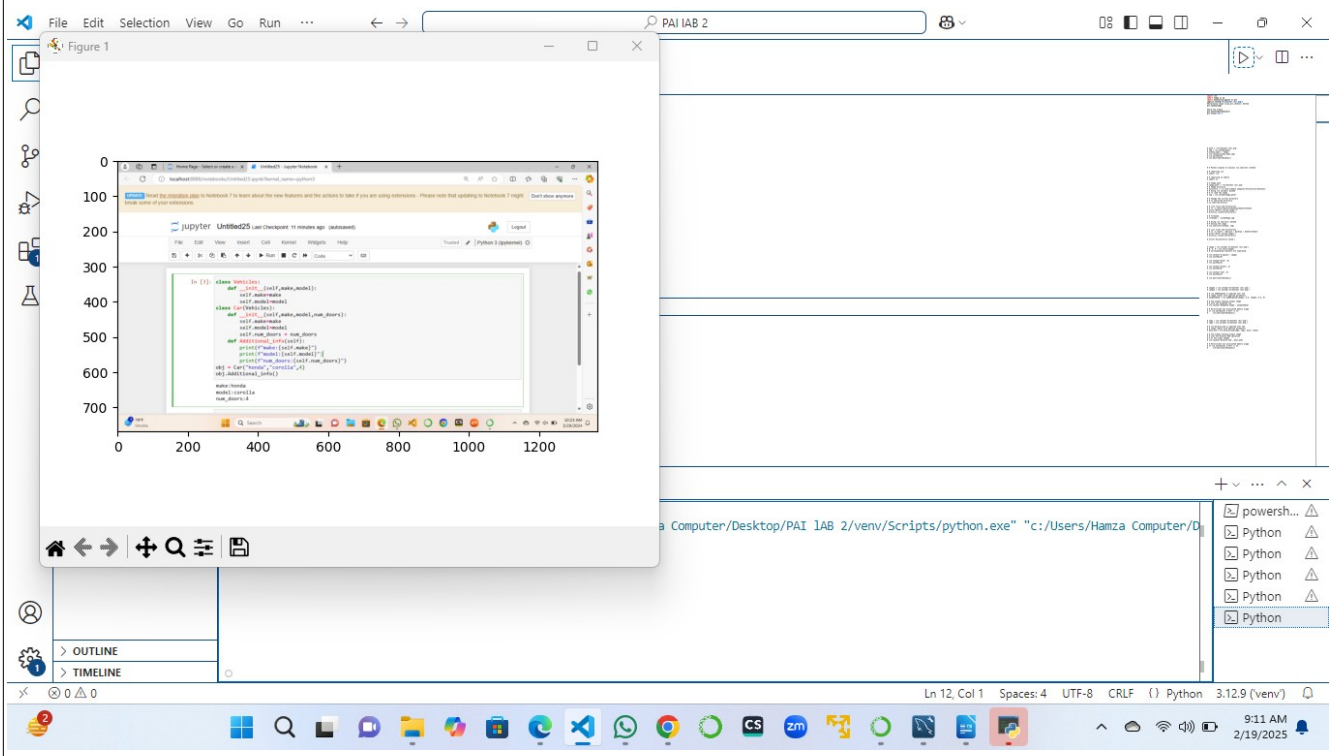
Programming for Artificial Intelligence

Lab 5

Image Processing

Reading an image:

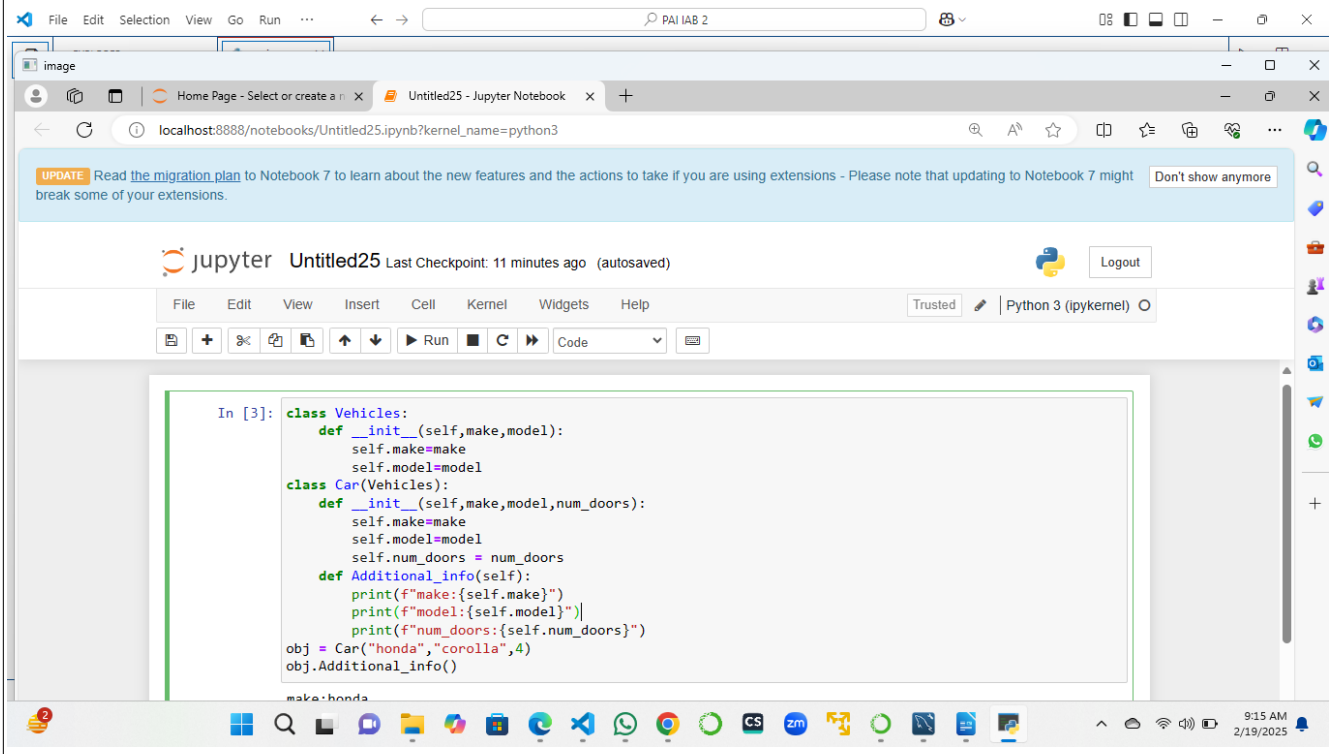
```
import cv2

import numpy as np
import matplotlib.pyplot as plt
img=cv2.imread("Screenshot (31).png")
#Displaying image using plt.imshow() method
plt.imshow(img)

#hold the window
plt.waitforbuttonpress()
plt.close('all')
```



Programming for Artificial Intelligence

Display an image:

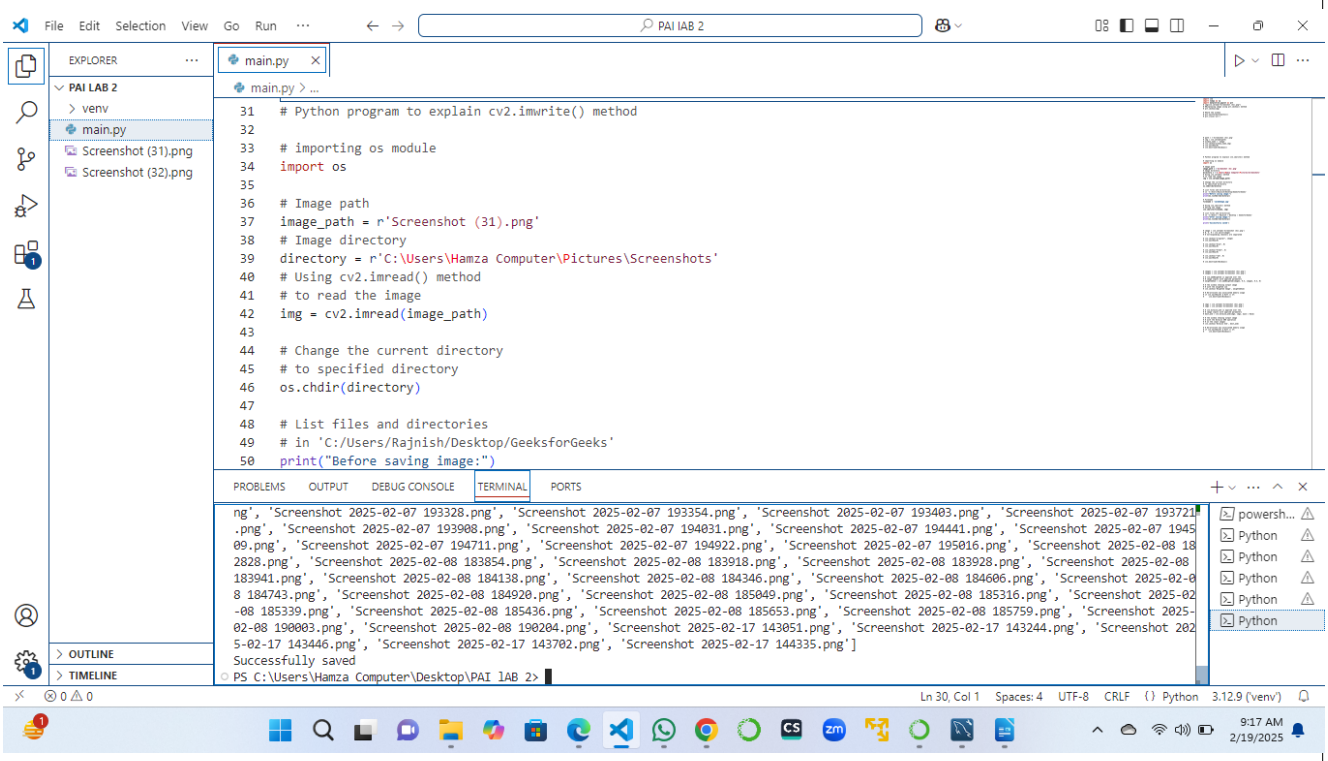
```
path = r'Screenshot (31).png'

img = cv2.imread(path)
window_name = 'image'
cv2.imshow(window_name,img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Writing an image:

```
import os


# Image path
image_path = r'Screenshot (31).png'
# Image directory
directory = r'C:\Users\Hamza Computer\Pictures\Screenshots'
# Using cv2.imread() method
# to read the image
img = cv2.imread(image_path)

# Change the current directory
```

```
# to specified directory
os.chdir(directory)

# List files and directories
# in 'C:/Users/Rajnish/Desktop/GeeksforGeeks'
print("Before saving image:")
print(os.listdir(directory))

# Filename
filename = 'savedImage.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, img)

# List files and directories
# in 'C:/Users / Rajnish / Desktop / GeeksforGeeks'
print("After saving image:")
print(os.listdir(directory))

print('Successfully saved')
```

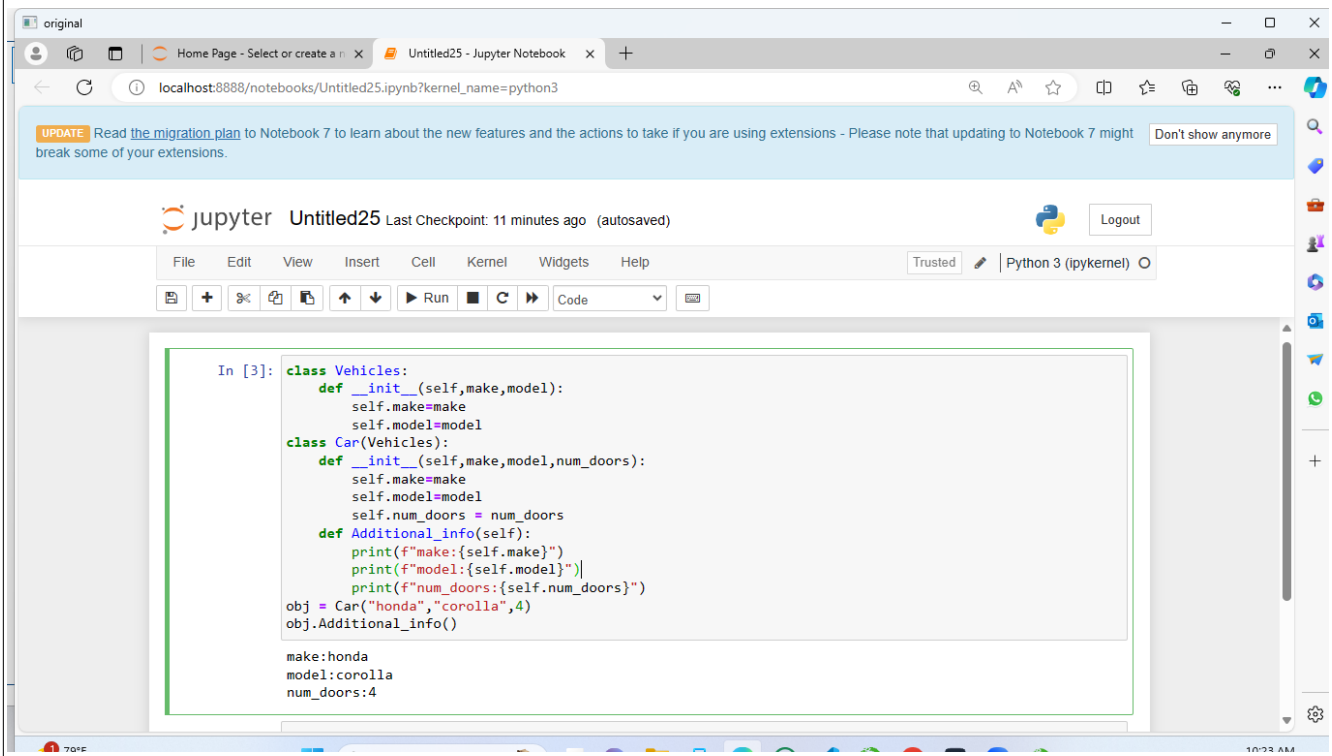


Programming for Artificial Intelligence

Color spacing:

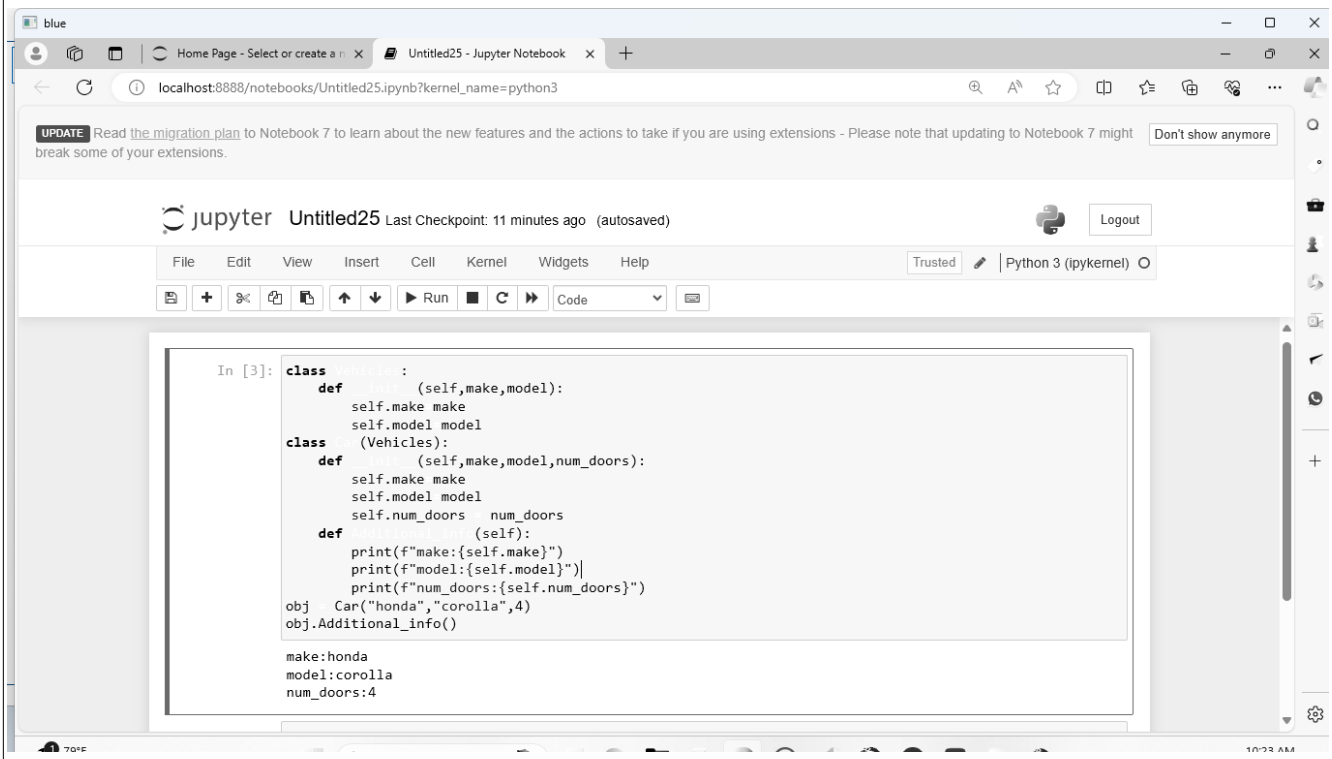
```
image = cv2.imread('Screenshot (31).png')
B, G, R = cv2.split(image)
# Corresponding channels are separated

cv2.imshow("original", image)
cv2.waitKey(0)

cv2.imshow("blue", B)
cv2.waitKey(0)

cv2.imshow("Green", G)
cv2.waitKey(0)

cv2.imshow("red", R)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Original:



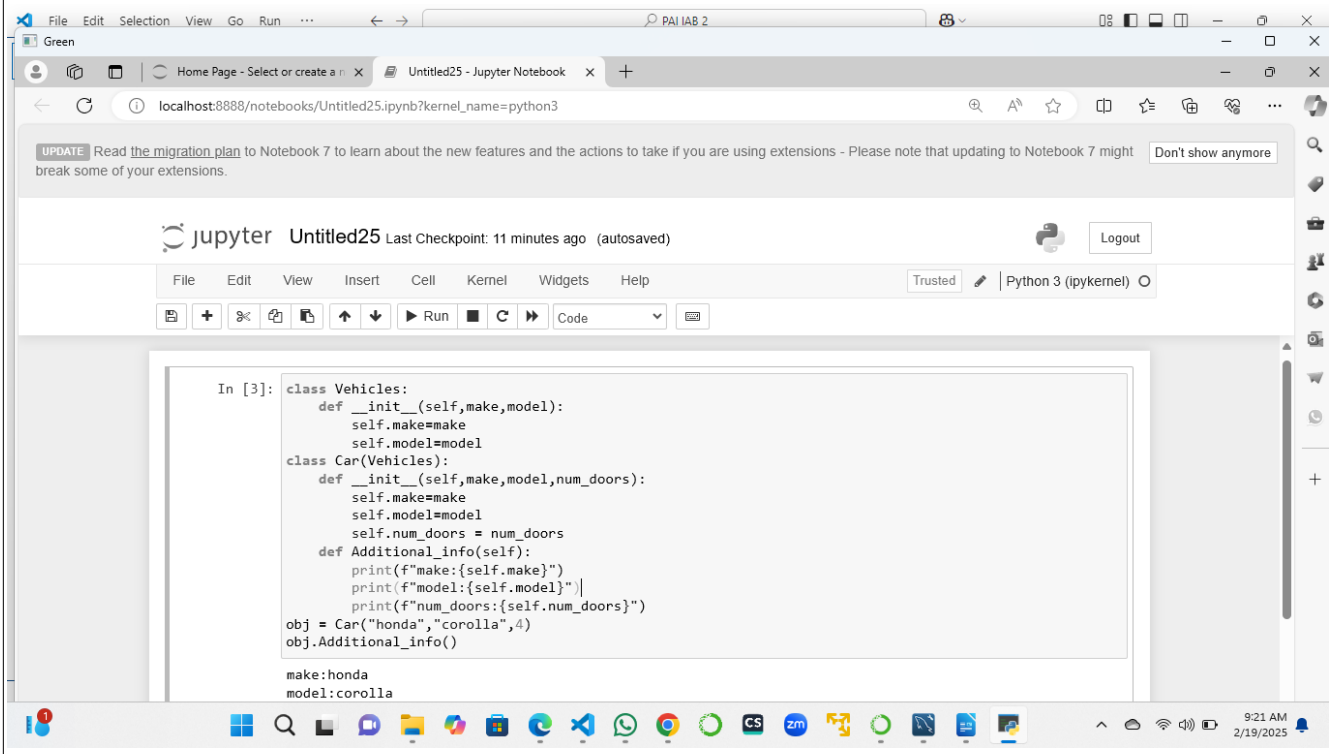
Programming for Artificial Intelligence

Blue:



```
In [3]:
class Vehicles:
    def __init__(self,make,model):
        self.make make
        self.model model
class Car(Vehicles):
    def __init__(self,make,model,num_doors):
        self.make make
        self.model model
        self.num_doors = num_doors
    def Additional_info(self):
        print(f"make:{self.make}")
        print(f"model:{self.model}")
        print(f"num_doors:{self.num_doors}")
obj = Car("honda","corolla",4)
obj.Additional_info()

make:honda
model:corolla
num_doors:4
```

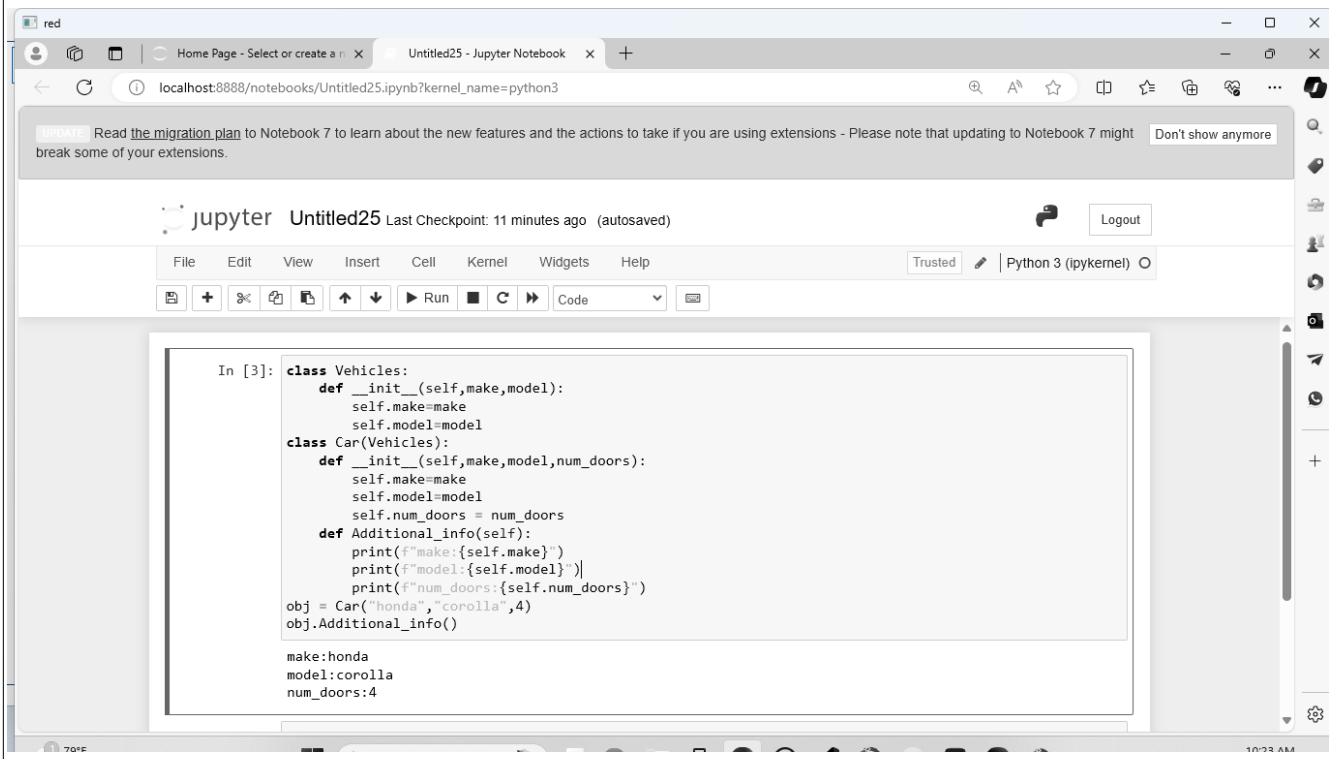
Green:



```
In [3]:
class Vehicles:
    def __init__(self,make,model):
        self.make=make
        self.model=model
class Car(Vehicles):
    def __init__(self,make,model,num_doors):
        self.make=make
        self.model=model
        self.num_doors = num_doors
    def Additional_info(self):
        print(f"make:{self.make}")
        print(f"model:{self.model}")
        print(f"num_doors:{self.num_doors}")
obj = Car("honda","corolla",4)
obj.Additional_info()

make:honda
model:corolla
```

Programming for Artificial Intelligence

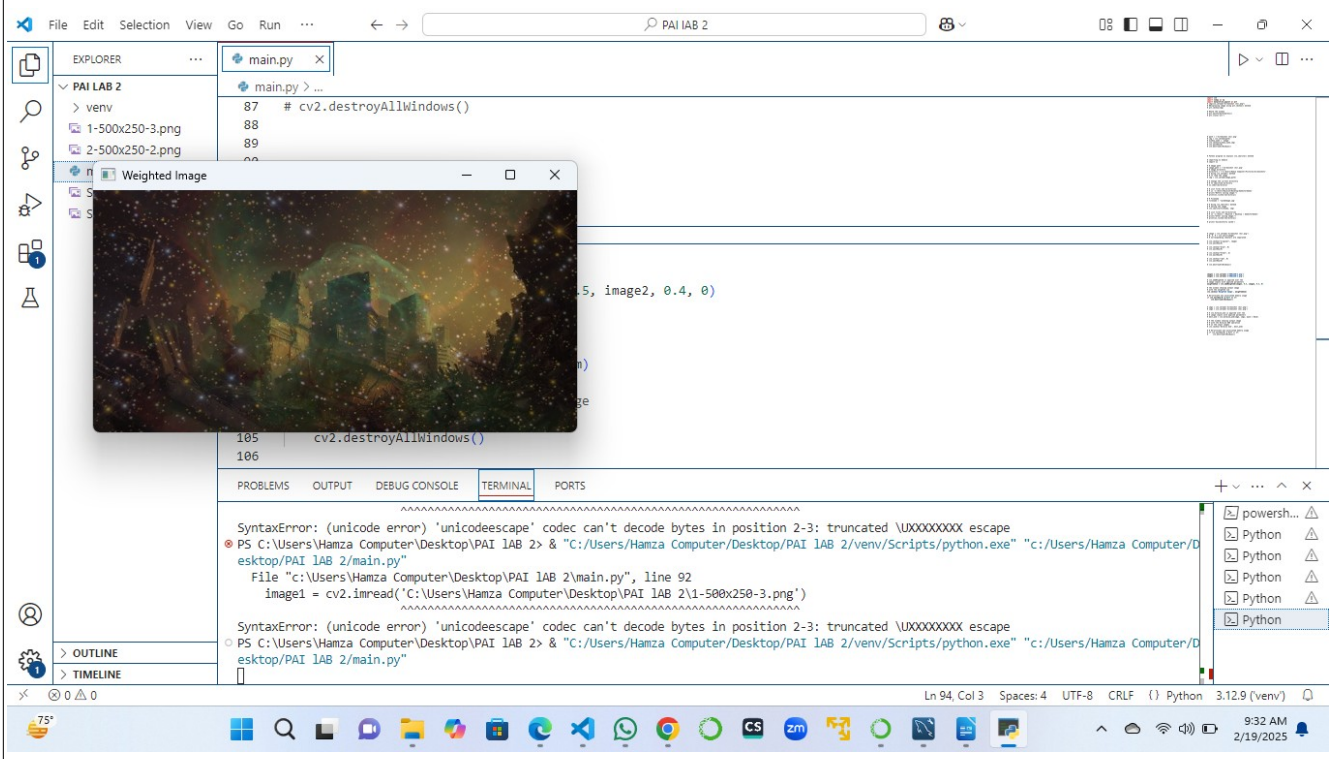
Red:



Arithmetic operations on images:

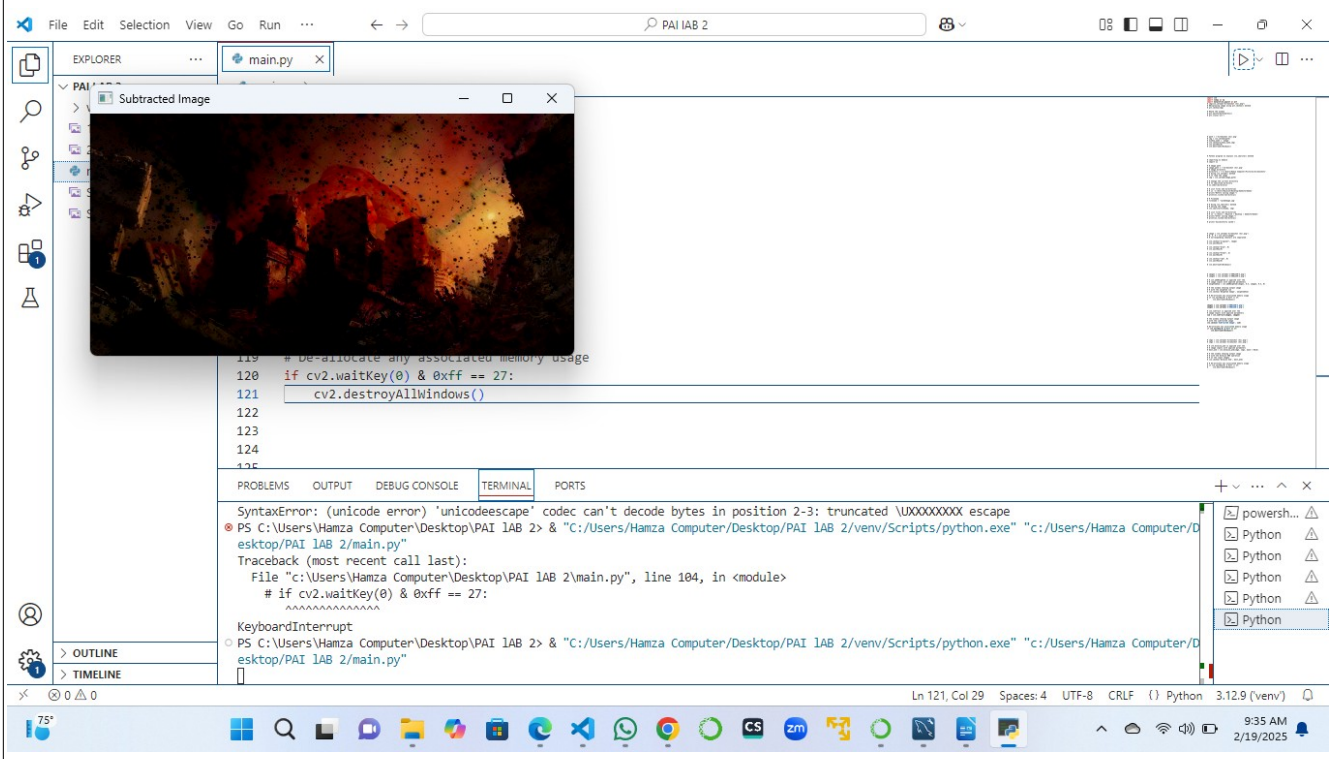
Adding images:

```
image1 = cv2.imread('1-500x250-3.png')

image2 = cv2.imread('2-500x250-2.png')

# cv2.addWeighted is applied over the
# image inputs with applied parameters
weightedSum = cv2.addWeighted(image1, 0.5, image2, 0.4, 0)

# the window showing output image
# with the weighted sum
cv2.imshow('Weighted Image', weightedSum)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```



Subtracting an image:

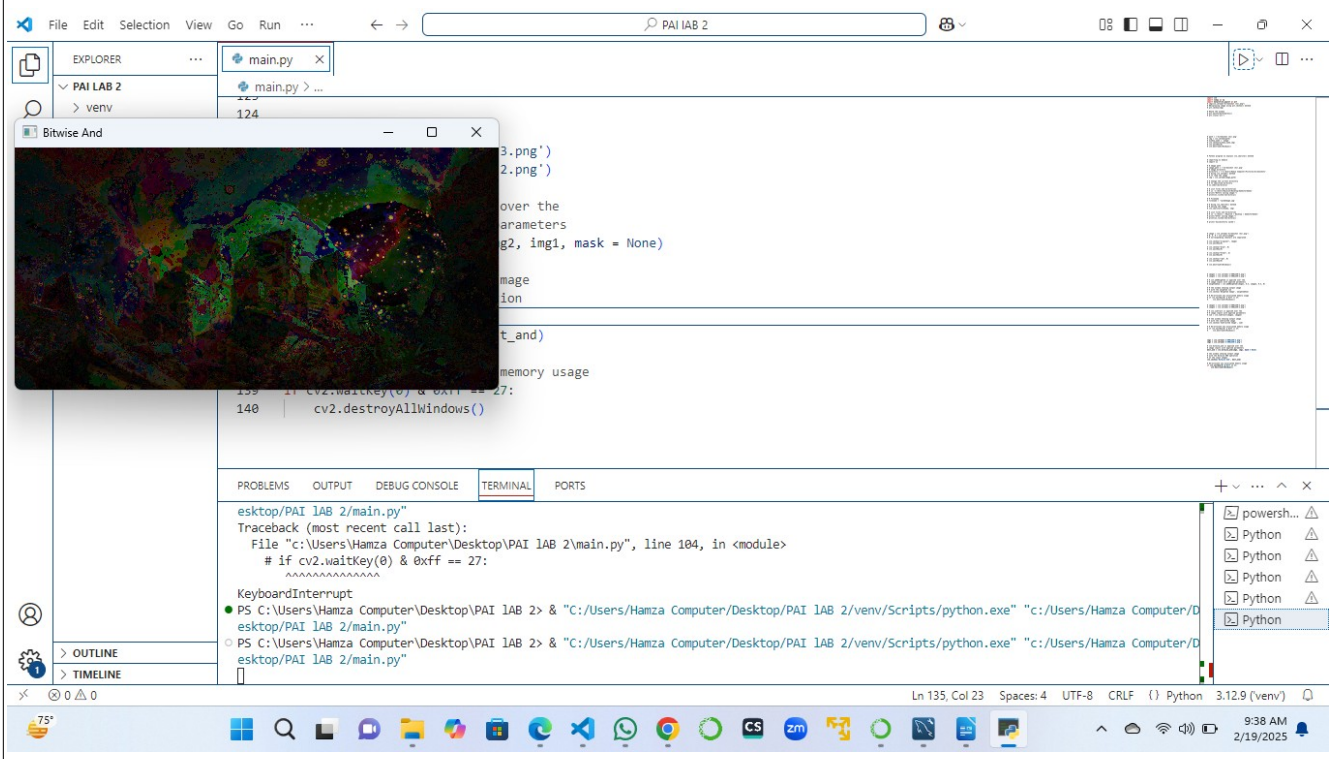
```
image1 = cv2.imread('1-500x250-3.png')
image2 = cv2.imread('2-500x250-2.png')

# cv2.subtract is applied over the
# image inputs with applied parameters
sub = cv2.subtract(image1, image2)

# the window showing output image
# with the subtracted image
cv2.imshow('Subtracted Image', sub)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```



Bitwise operations:

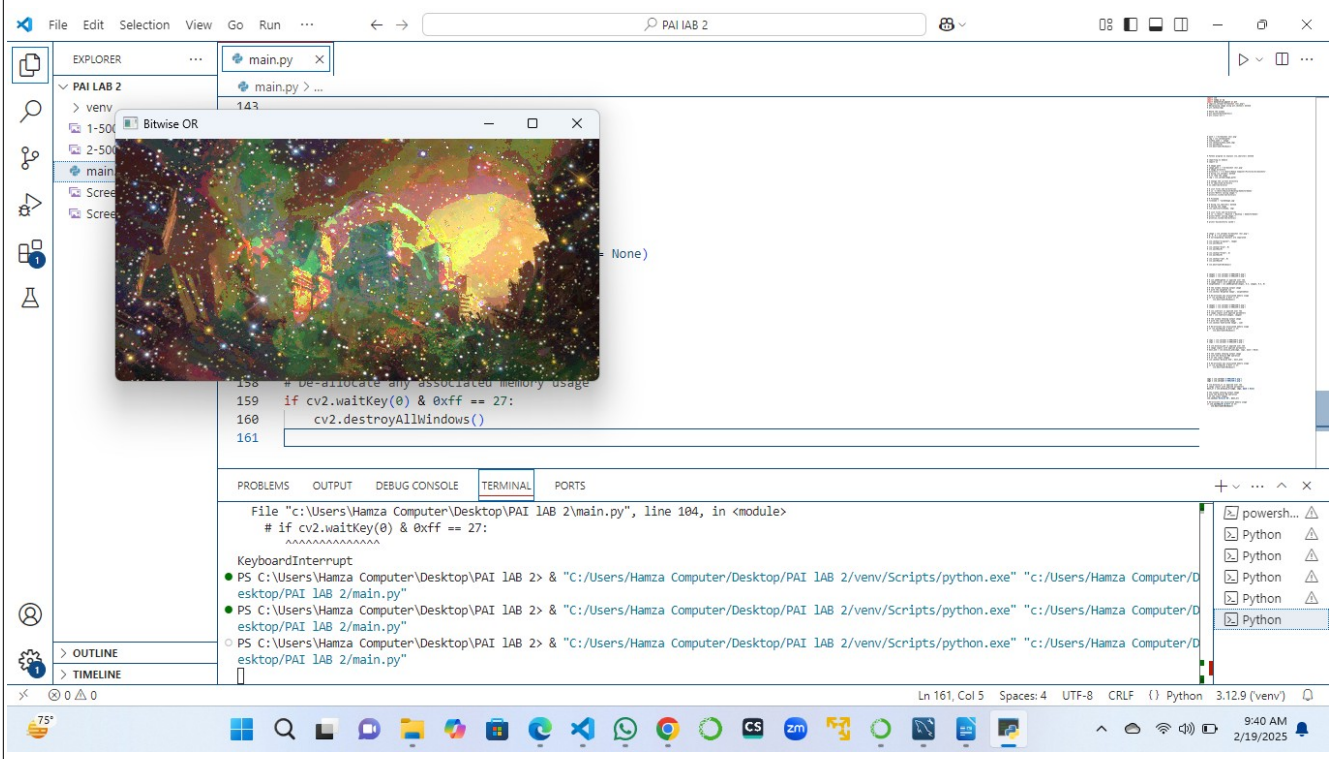
And:

```
img1 = cv2.imread('1-500x250-3.png')

img2 = cv2.imread('2-500x250-2.png')

# cv2.bitwise_and is applied over the
# image inputs with applied parameters
dest_and = cv2.bitwise_and(img2, img1, mask = None)

# the window showing output image
# with the Bitwise AND operation
# on the input images
cv2.imshow('Bitwise And', dest_and)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```



OR:

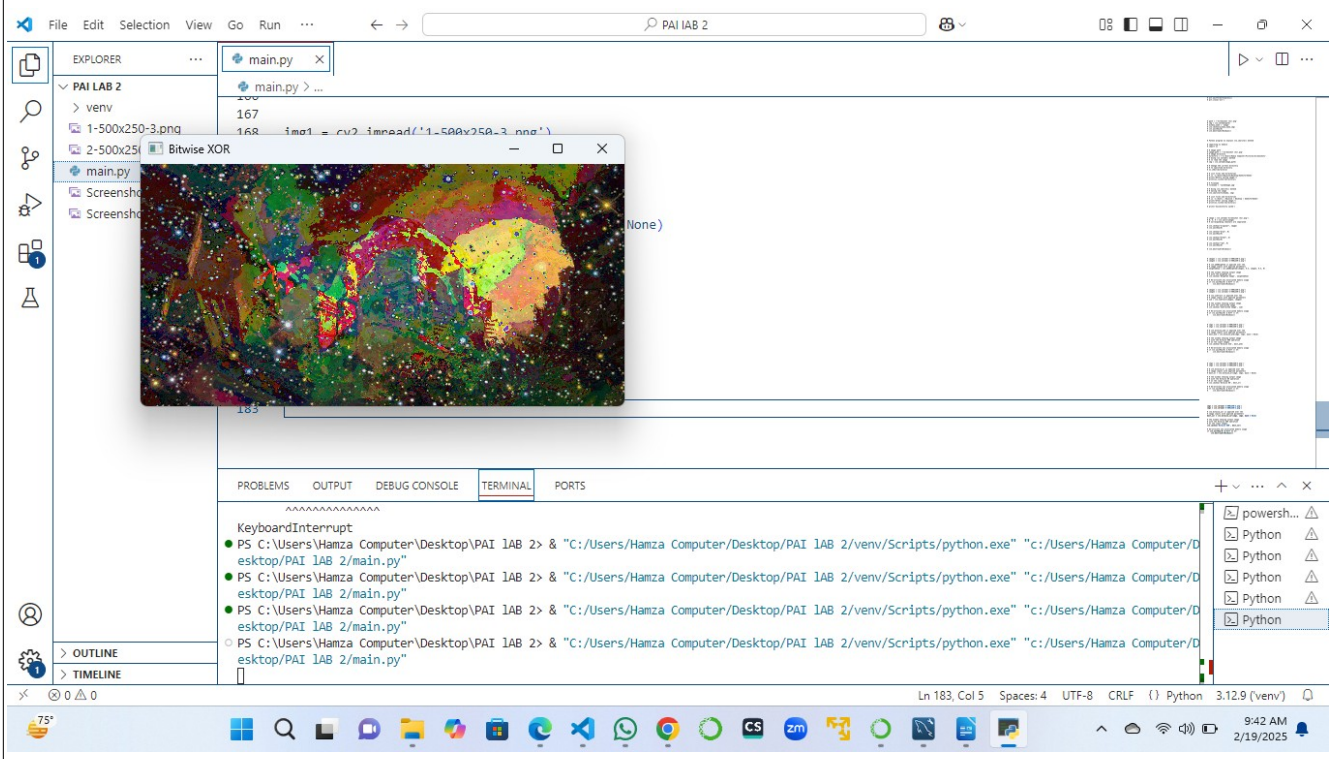
```
img1 = cv2.imread('1-500x250-3.png')
img2 = cv2.imread('2-500x250-2.png')

# cv2.bitwise_or is applied over the
# image inputs with applied parameters
dest_or = cv2.bitwise_or(img2, img1, mask = None)

# the window showing output image
# with the Bitwise OR operation
# on the input images
cv2.imshow('Bitwise OR', dest_or)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```



XOR:

```
img1 = cv2.imread('1-500x250-3.png')

img2 = cv2.imread('2-500x250-2.png')

# cv2.bitwise_xor is applied over the
# image inputs with applied parameters
dest_xor = cv2.bitwise_xor(img1, img2, mask = None)

# the window showing output image
# with the Bitwise XOR operation
# on the input images
cv2.imshow('Bitwise XOR', dest_xor)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Programming for Artificial Intelligence



NOT:

```
img1 = cv2.imread('1-500x250-3.png')

img2 = cv2.imread('2-500x250-2.png')

# cv2.bitwise_not is applied over the
# image input with applied parameters
dest_not1 = cv2.bitwise_not(img1, mask = None)
dest_not2 = cv2.bitwise_not(img2, mask = None)

# the windows showing output image
# with the Bitwise NOT operation
# on the 1st and 2nd input image
cv2.imshow('Bitwise NOT on image 1', dest_not1)
cv2.imshow('Bitwise NOT on image 2', dest_not2)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Programming for Artificial Intelligence

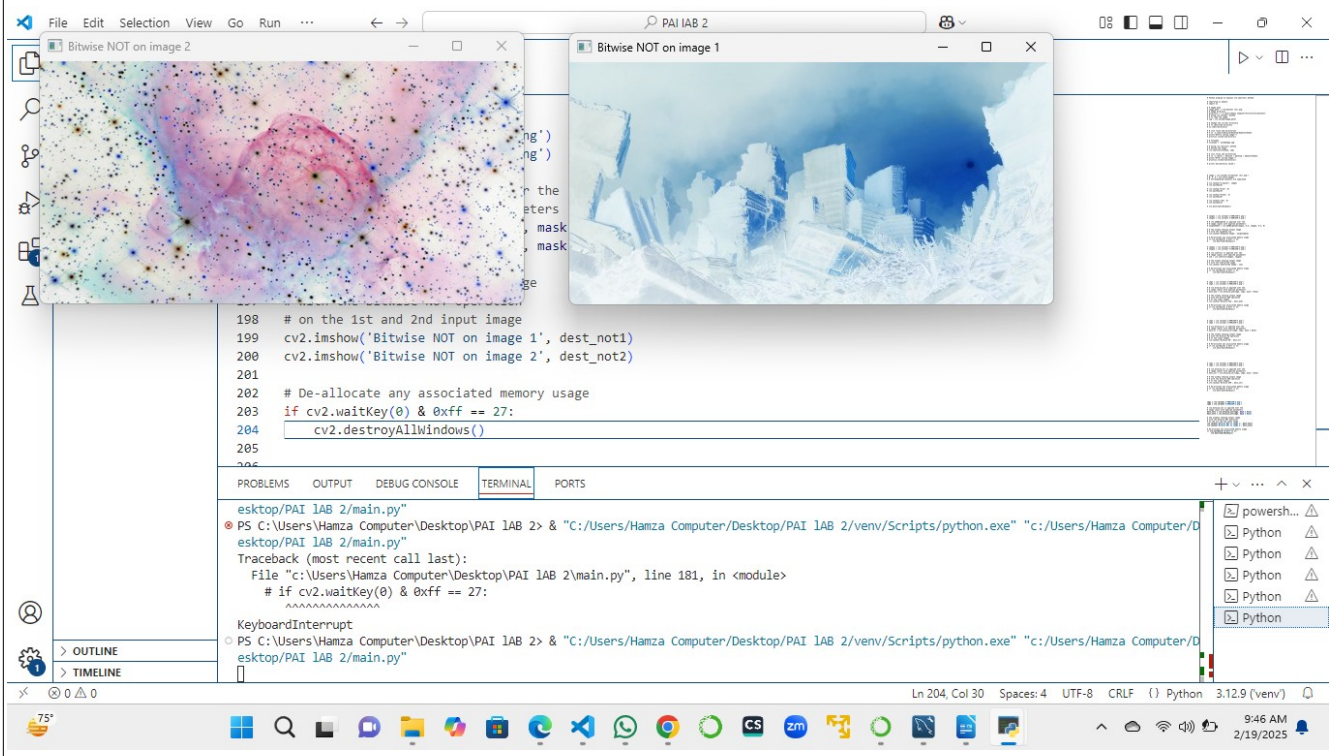
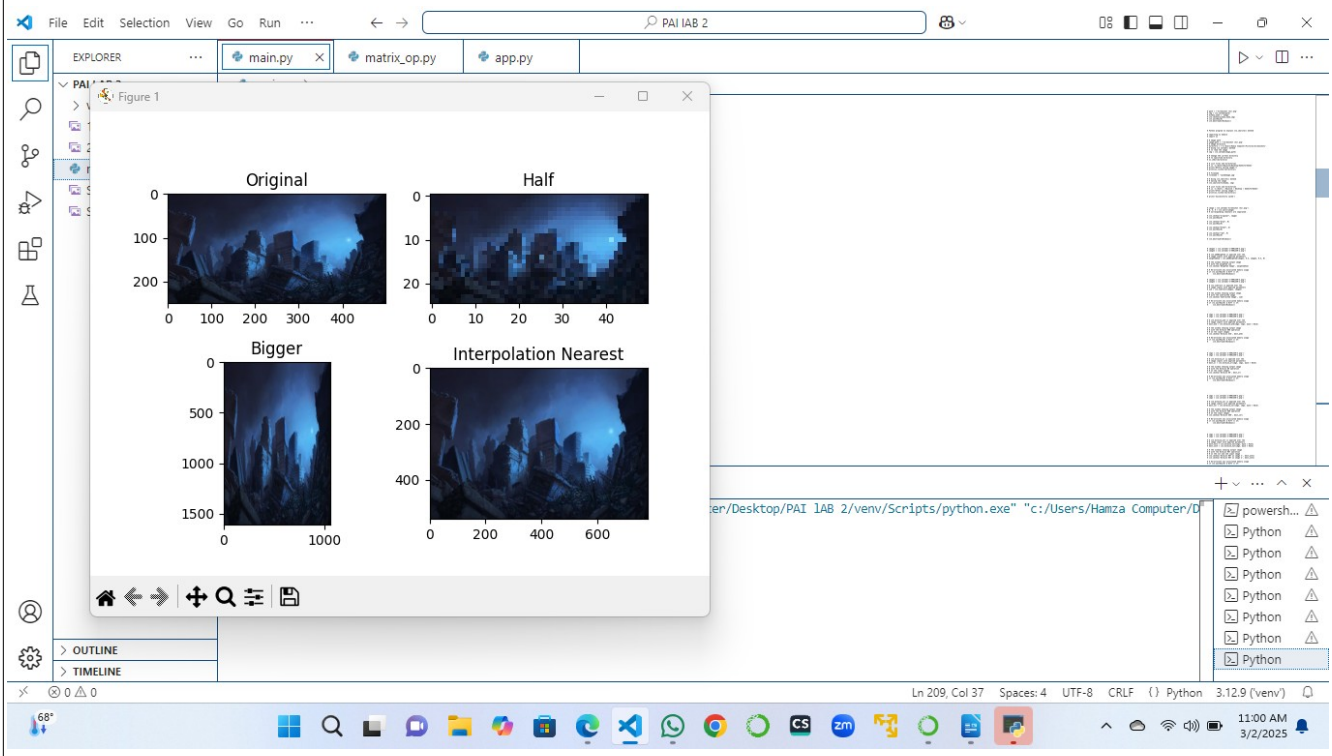


Image resizing:

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('1-500x250-3.png', 1)
# Loading the image

half = cv2.resize(image, (0, 0), fx = 0.1, fy = 0.1)
bigger = cv2.resize(image, (1050, 1610))

stretch_near = cv2.resize(image, (780, 540),
               interpolation = cv2.INTER_LINEAR)


Titles =["Original", "Half", "Bigger", "Interpolation Nearest"]
images =[image, half, bigger, stretch_near]
count = 4

for i in range(count):
    plt.subplot(2, 2, i + 1)
    plt.title(Titles[i])
    plt.imshow(images[i])
```

plt.show()

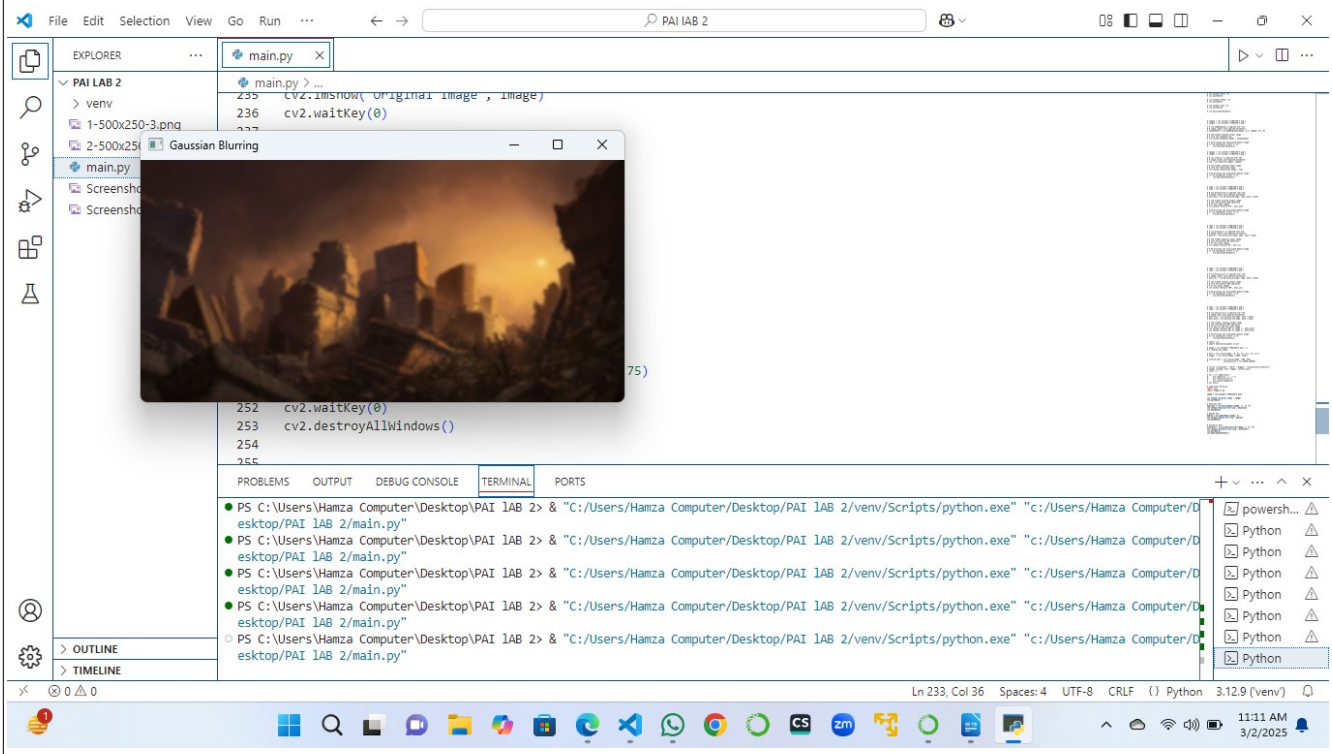


Blurring an image:

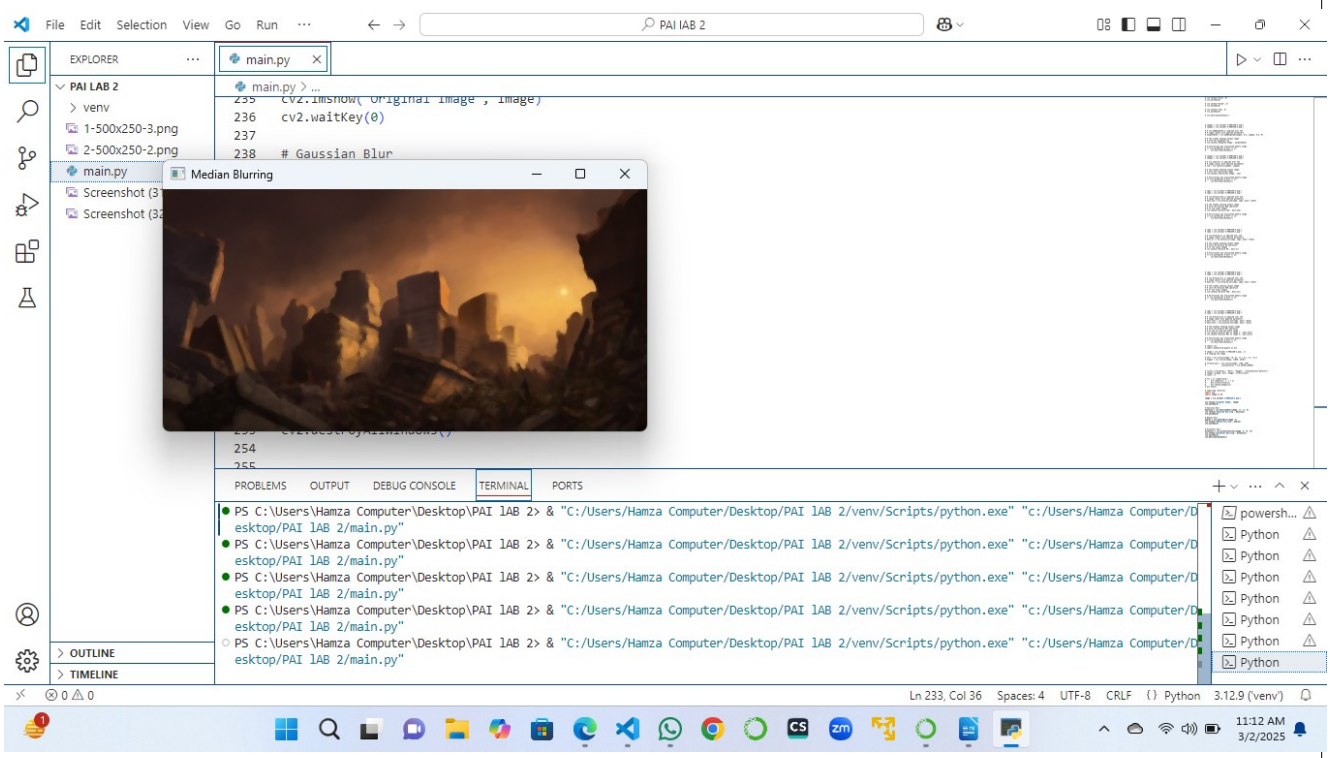
```
import cv2
import numpy as np

image = cv2.imread('1-500x250-3.png')

# Gaussian Blur
Gaussian = cv2.GaussianBlur(image, (7, 7), 0)
cv2.imshow('Gaussian Blurring', Gaussian)
cv2.waitKey(0)

# Median Blur
median = cv2.medianBlur(image, 5)
cv2.imshow('Median Blurring', median)
cv2.waitKey(0)


# Bilateral Blur
bilateral = cv2.bilateralFilter(image, 9, 75, 75)
```

```
cv2.imshow('Bilateral Blurring', bilateral)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

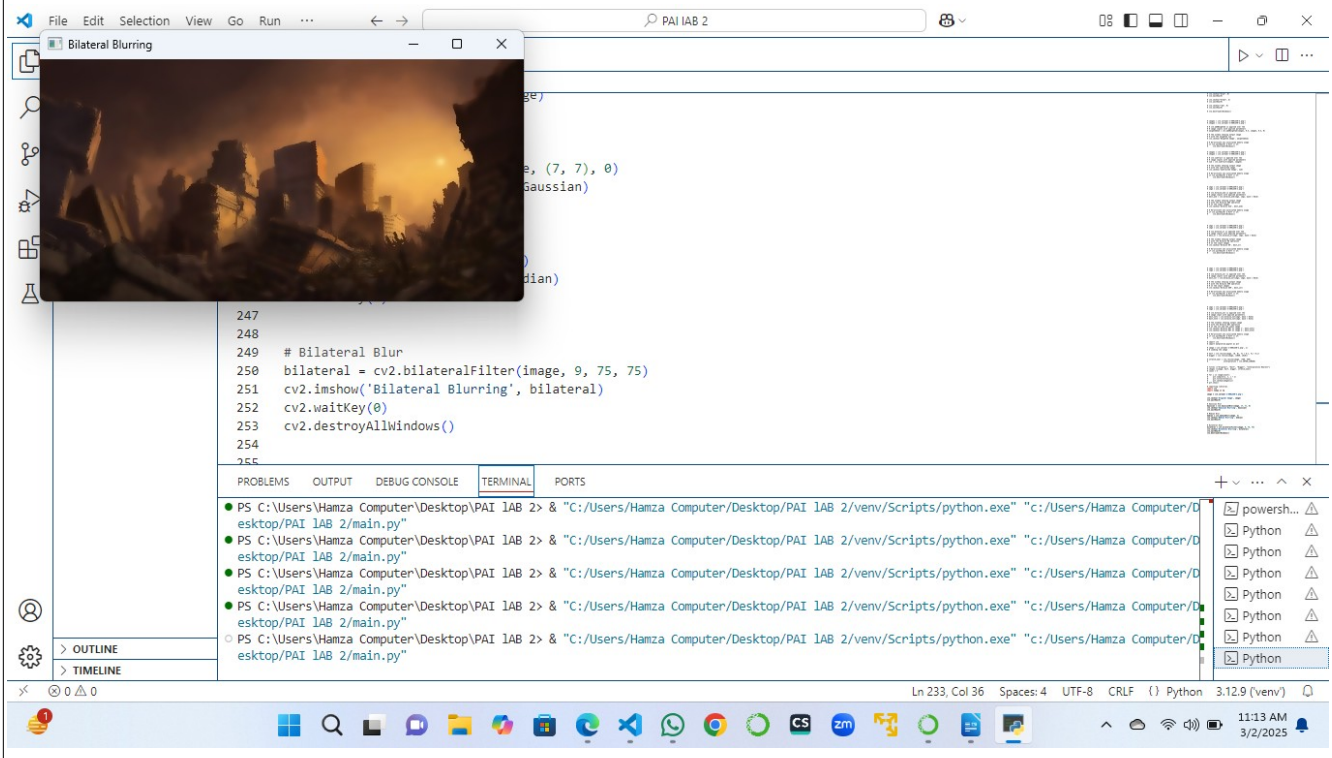
Gaussian blur:



Median blur:

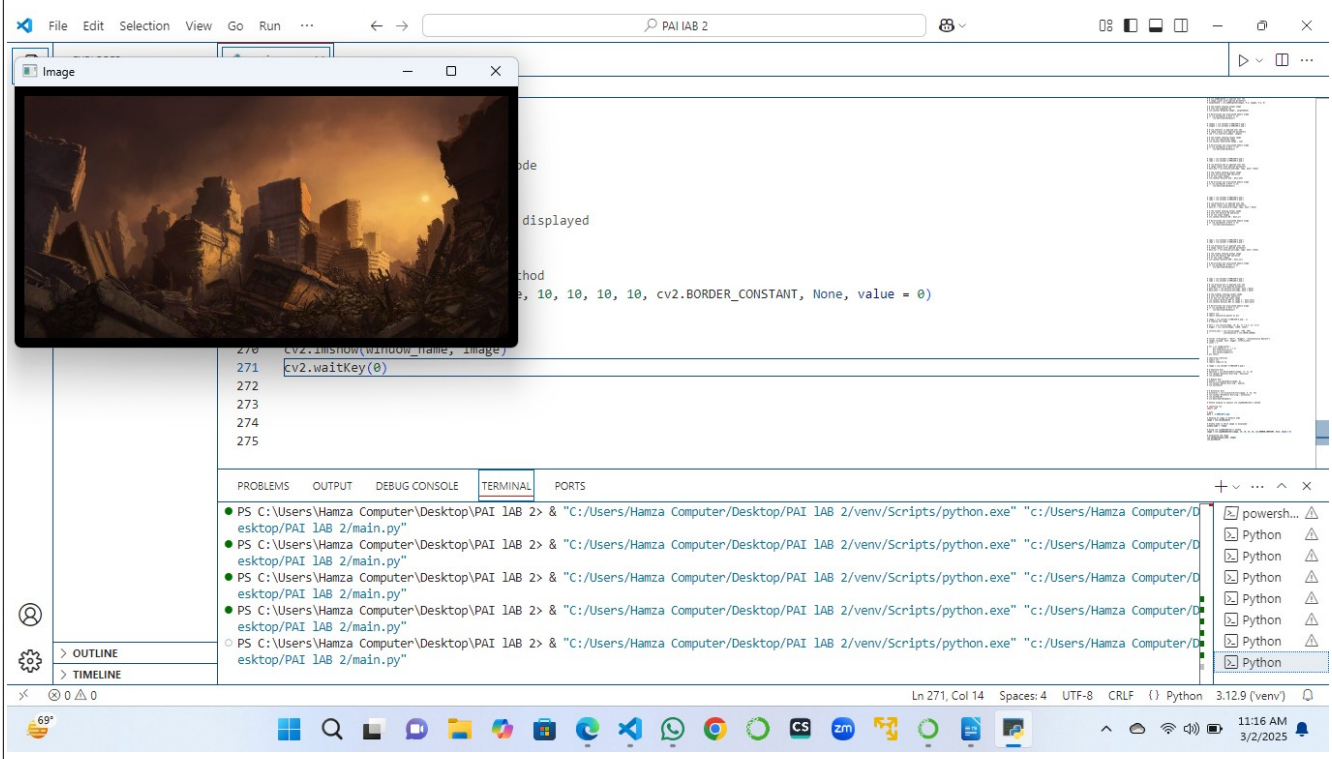


Bilateral blur:



Creating an boarder around image:

```
import cv2

# path
path = '1-500x250-3.png'

# Reading an image in default mode
image = cv2.imread(path)

# Window name in which image is displayed
window_name = 'Image'

# Using cv2.copyMakeBorder() method
image = cv2.copyMakeBorder(image, 10, 10, 10, 10, cv2.BORDER_CONSTANT, None, value = 0)

# Displaying the image
cv2.imshow(window_name, image)
cv2.waitKey(0)
```

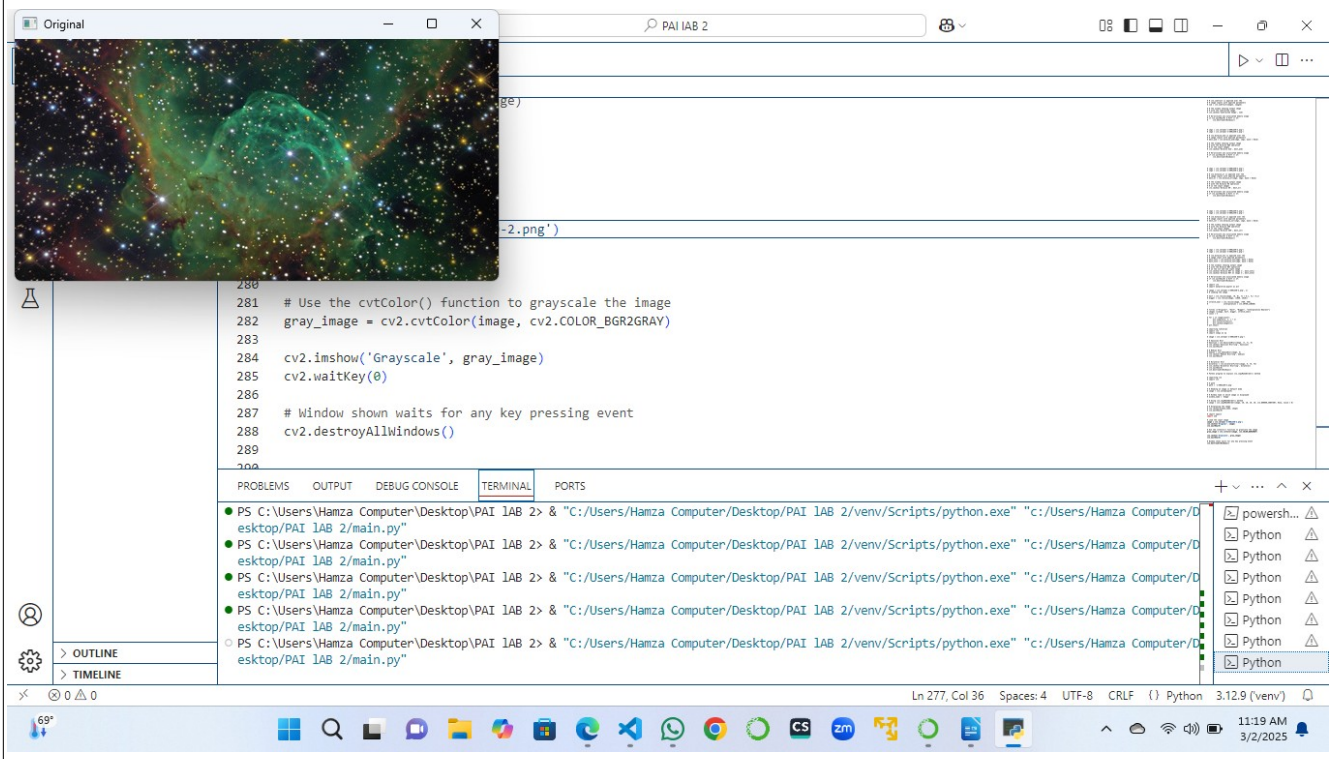


Programming for Artificial Intelligence

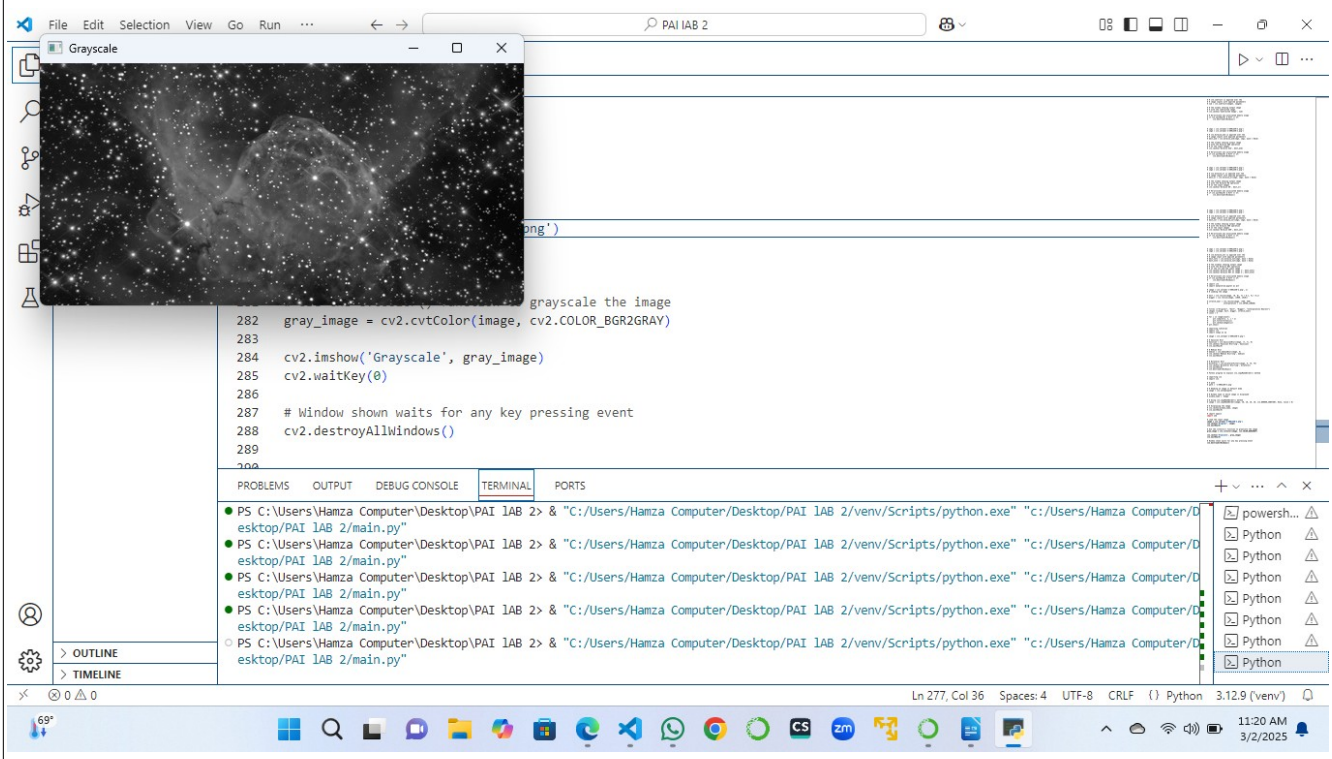
Grayscalling of an image:

```
import cv2

# Load the input image
image = cv2.imread('2-500x250-2.png')
cv2.imshow('Original', image)
cv2.waitKey(0)

# Use the cvtColor() function to grayscale the image
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imshow('Grayscale', gray_image)
cv2.waitKey(0)

# Window shown waits for any key pressing event
cv2.destroyAllWindows()
```

Original:



Grayscale:



Erosion and dilation of an image:

```
import cv2
import numpy as np

# Reading the input image
img = cv2.imread('1-500x250-3.png', 0)

# Taking a matrix of size 5 as the kernel
kernel = np.ones((5, 5), np.uint8)

img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)

cv2.waitKey(0)
```

Programming for Artificial Intelligence

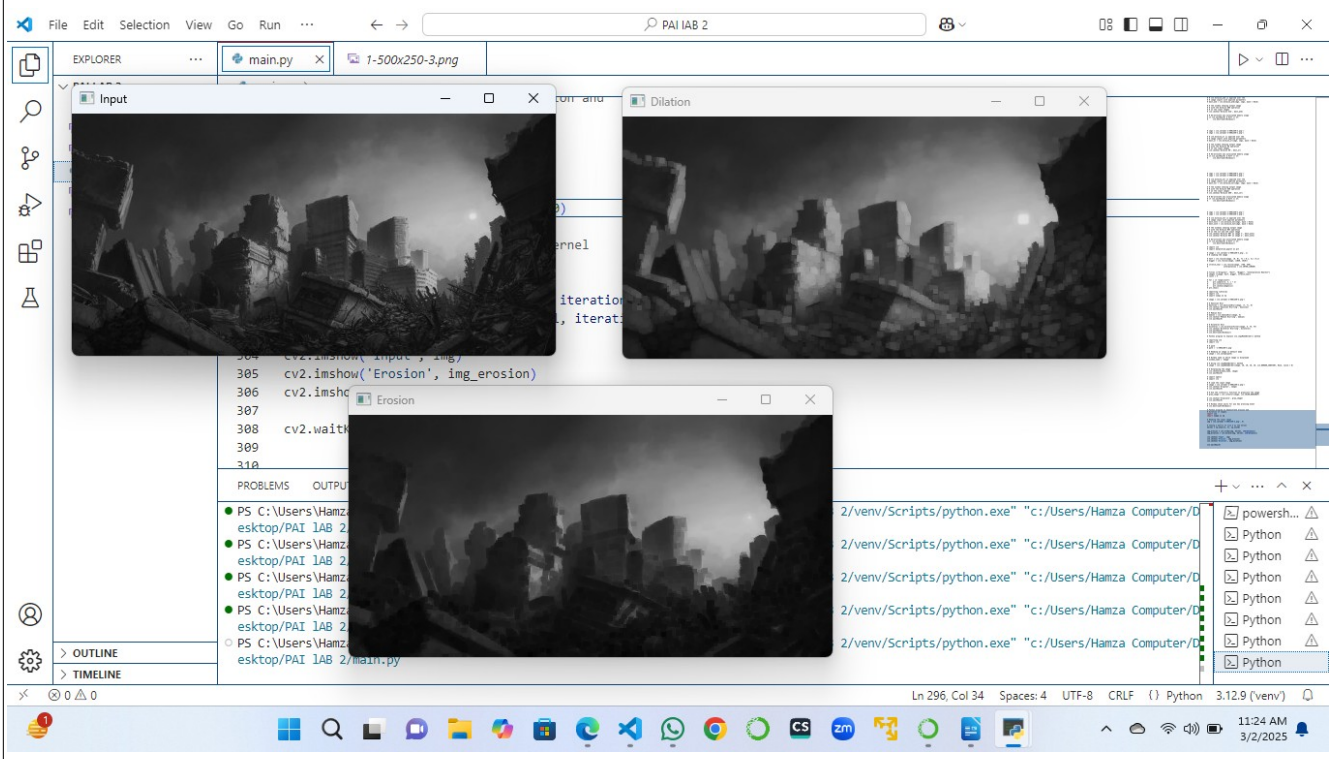


Image rotating:

```
import cv2
import matplotlib.pyplot as plt

# Read image from disk.
img = cv2.imread('1-500x250-3.png')

# Convert BGR image to RGB
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Image rotation parameter
center = (image_rgb.shape[1] // 2, image_rgb.shape[0] // 2)
angle = 30
scale = 1

# getRotationMatrix2D creates a matrix needed for transformation.
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)

# We want matrix for rotation w.r.t center to 30 degree without scaling.
rotated_image = cv2.warpAffine(image_rgb, rotation_matrix, (img.shape[1], img.shape[0]))

# Create subplots
```

```
fig, axs = plt.subplots(1, 2, figsize=(7, 4))

# Plot the original image
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')

# Plot the Rotated image
axs[1].imshow(rotated_image)
axs[1].set_title('Image Rotation')

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

# Display the subplots
plt.tight_layout()
plt.show()
```

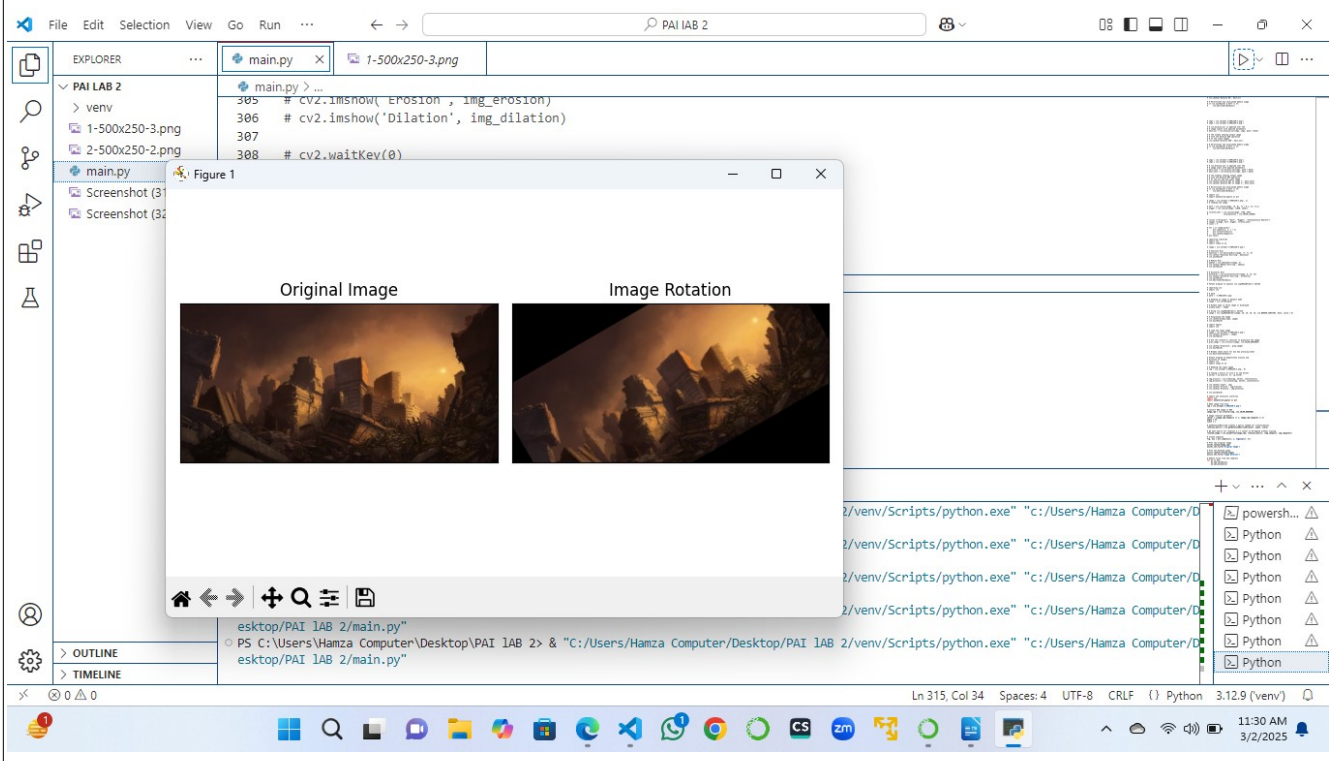
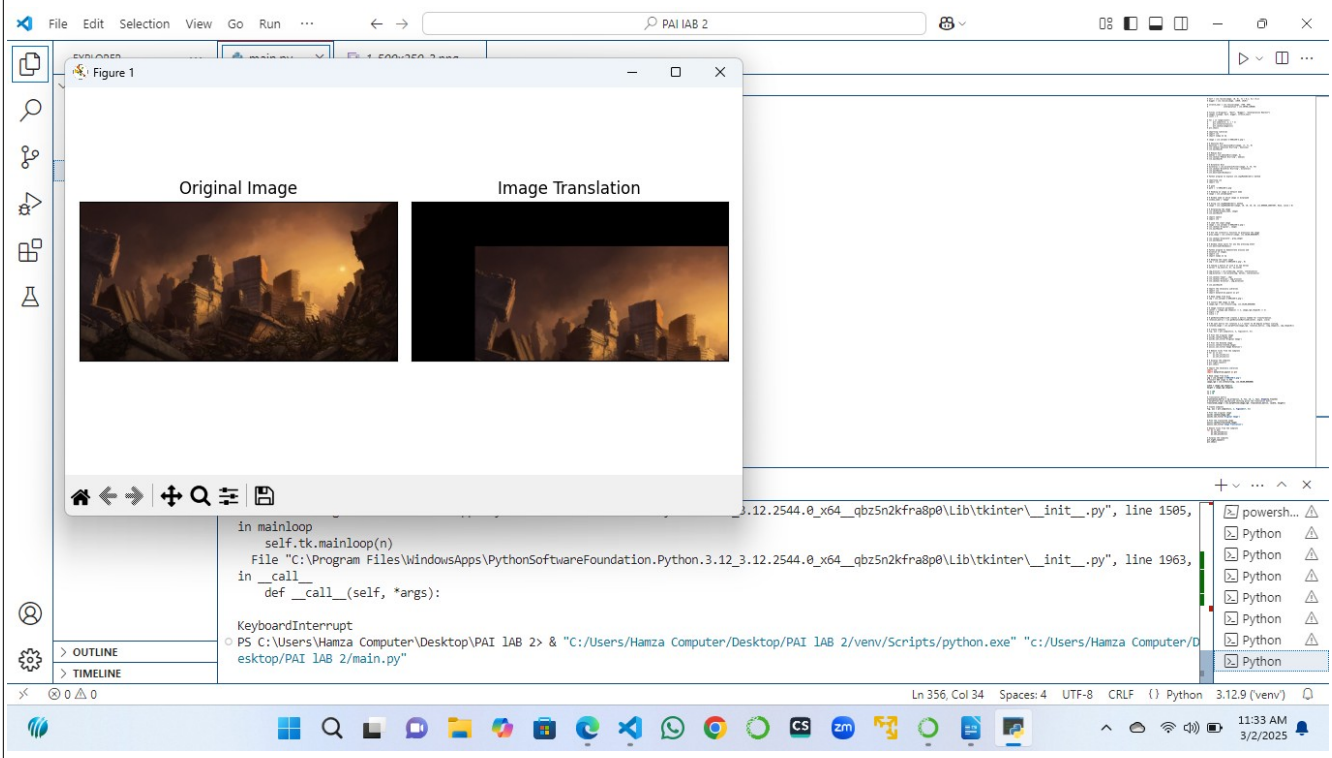


Image Translation:

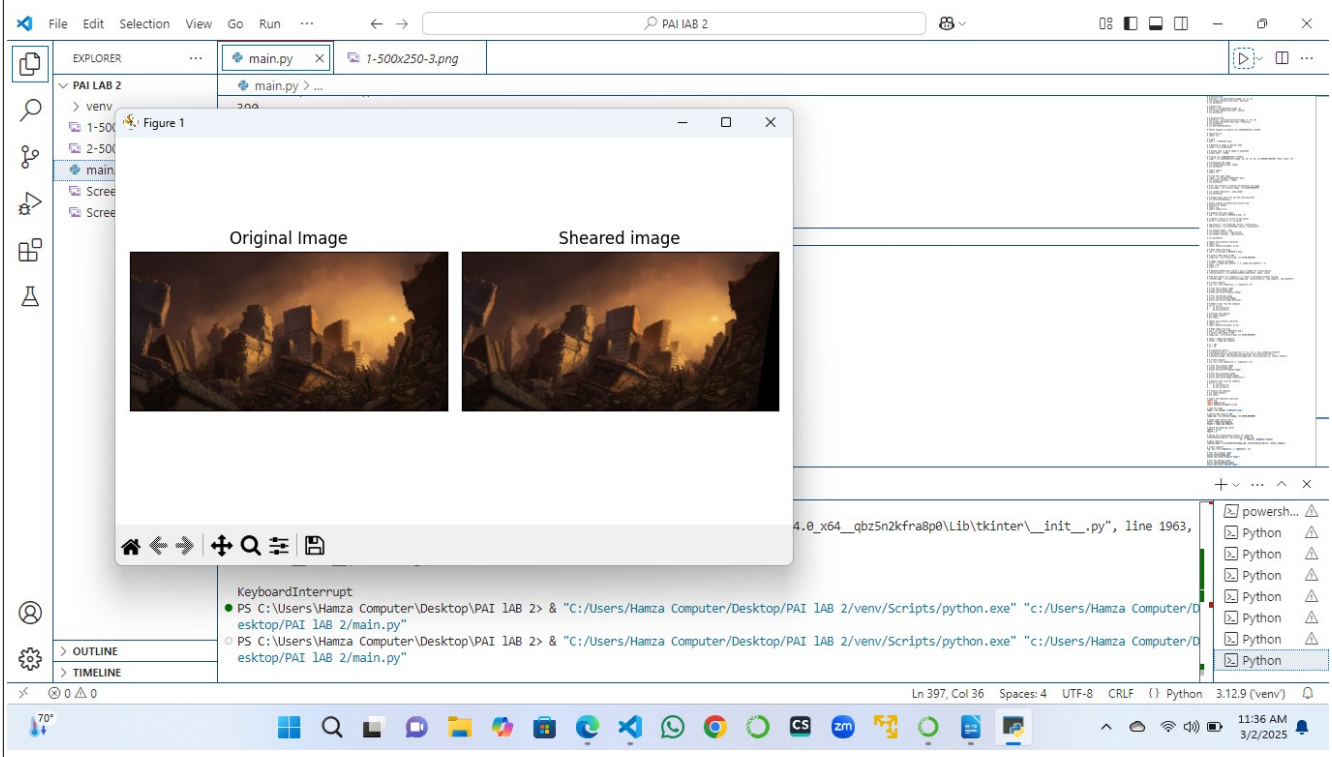
```
import cv2
import matplotlib.pyplot as plt

# Read image from disk.
img = cv2.imread('1-500x250-3.png')
# Convert BGR image to RGB
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

width = image_rgb.shape[1]
height = image_rgb.shape[0]

tx = 100
ty = 70

# Translation matrix
translation_matrix = np.array([[1, 0, tx], [0, 1, ty]], dtype=np.float32)
# warpAffine does appropriate shifting given the Translation matrix.
translated_image = cv2.warpAffine(image_rgb, translation_matrix, (width, height))

# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(7, 4))

# Plot the original image
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')

# Plot the transalted image
axs[1].imshow(translated_image)
axs[1].set_title('Image Translation')

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

# Display the subplots
plt.tight_layout()
plt.show()
```



Shearing an image:

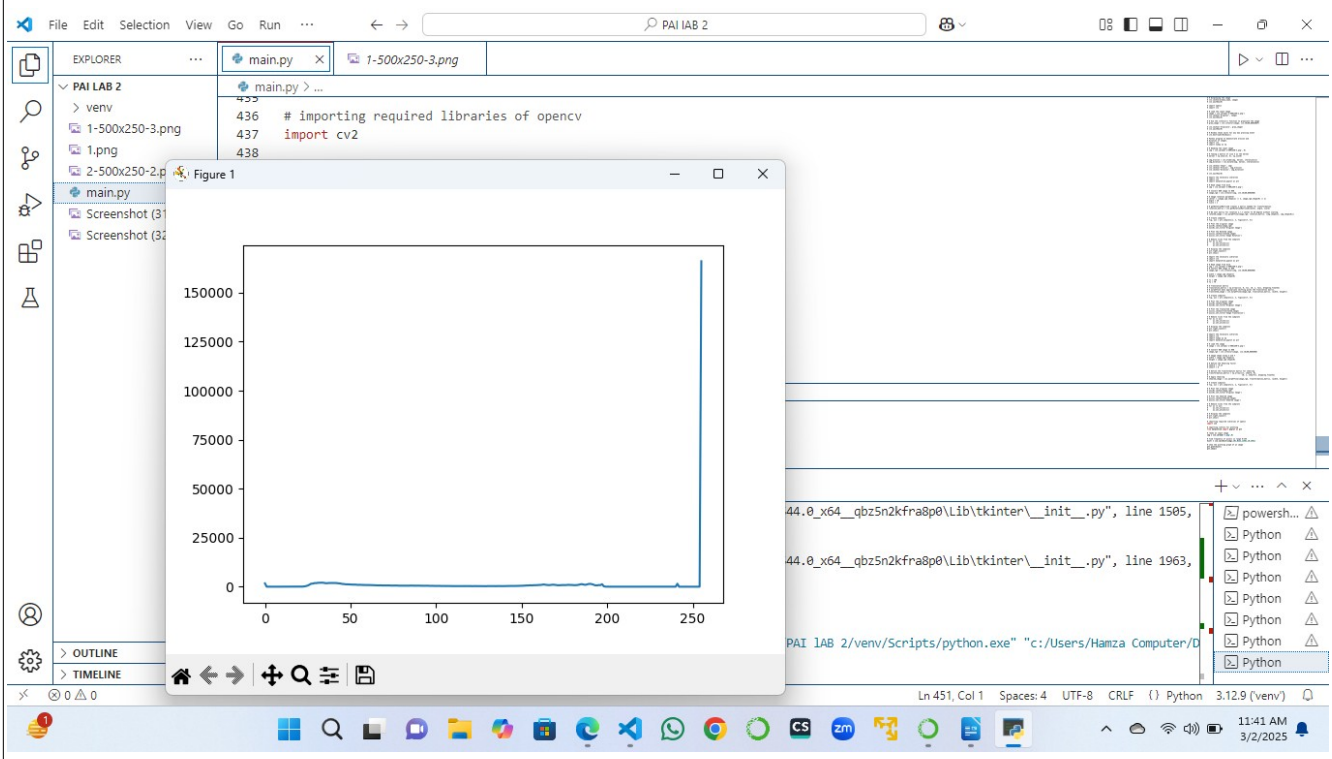
```
import cv2

import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('1-500x250-3.png')

# Convert BGR image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Image shape along X and Y
width = image_rgb.shape[1]
height = image_rgb.shape[0]

# Define the Shearing factor
shearX = -0.15
shearY = 0

# Define the Transformation matrix for shearing
transformation_matrix = np.array([[1, shearX, 0],
                                  [0, 1, shearY]], dtype=np.float32)
```

```
# Apply shearing
sheared_image = cv2.warpAffine(image_rgb, transformation_matrix, (width, height))

# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(7, 4))

# Plot the original image
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')

# Plot the Sheared image
axs[1].imshow(sheared_image)
axs[1].set_title('Sheared image')

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

# Display the subplots
plt.tight_layout()
plt.show()
```



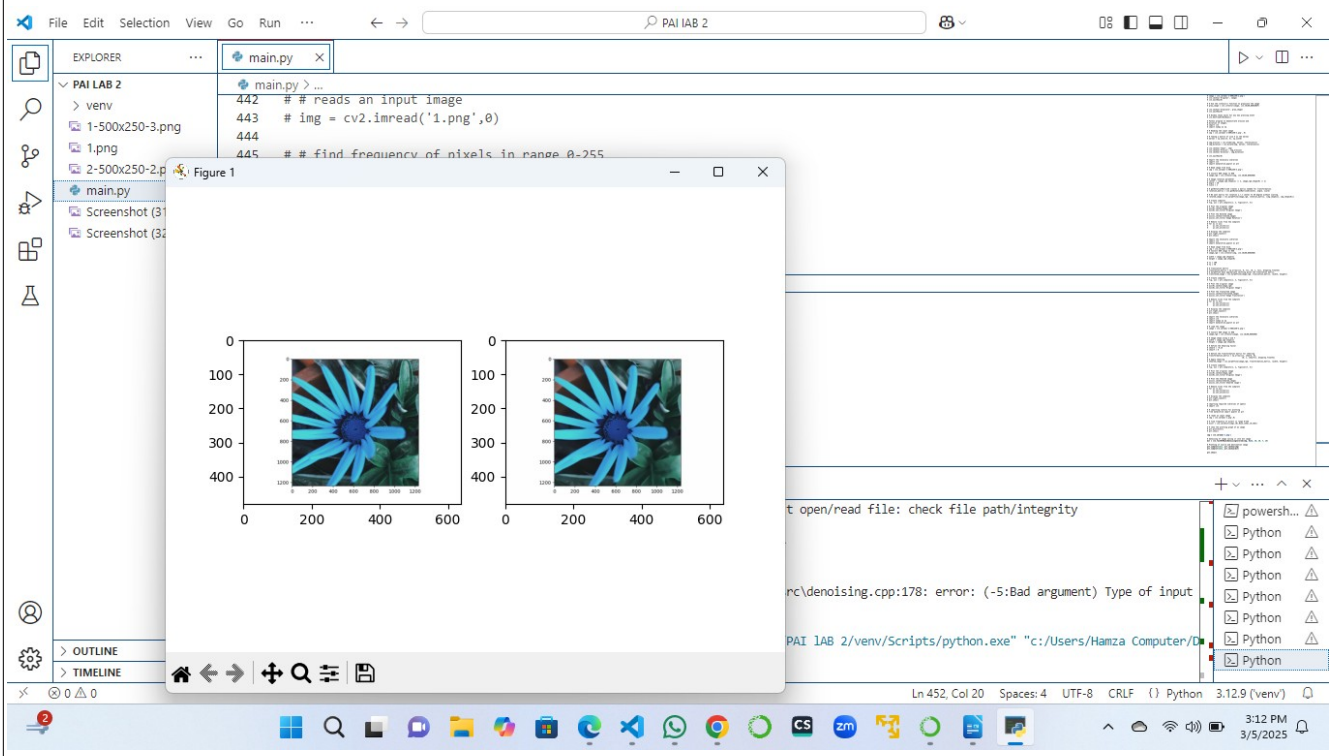
Analyze an image using histogram:

```
import cv2

# importing library for plotting
from matplotlib import pyplot as plt

# reads an input image
img = cv2.imread('1.png',0)

# find frequency of pixels in range 0-255
histr = cv2.calcHist([img],[0],None,[256],[0,256])

# show the plotting graph of an image
plt.plot(histr)
plt.show()
```



Denoising of image:

```
img = cv2.imread('1.png')

# denoising of image saving it into dst image
dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 15)
# Plotting of source and destination image
plt.subplot(121), plt.imshow(img)
plt.subplot(122), plt.imshow(dst)

plt.show()
```



Visualization of an image:

```
img = cv2.imread('1.png', 1)

# We can alternatively convert
# image by using cv2color
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Shows the image
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

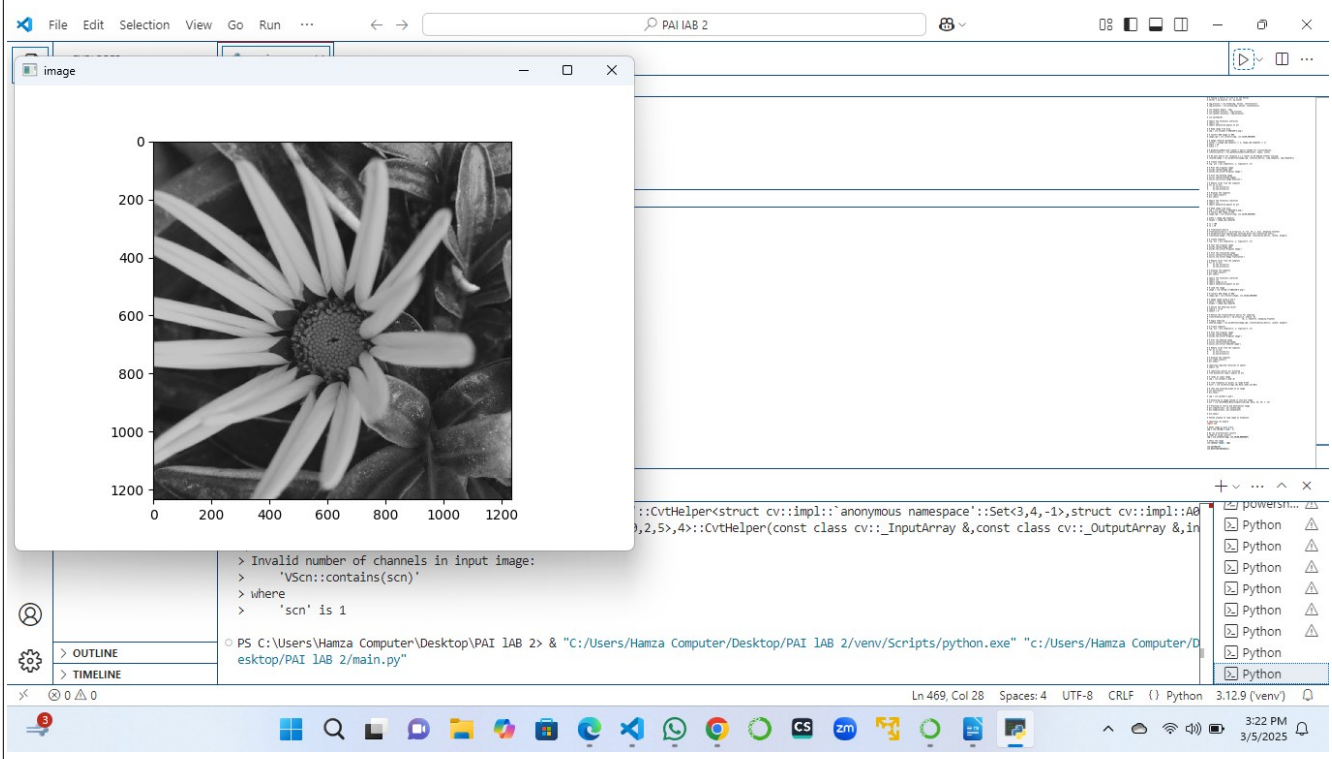
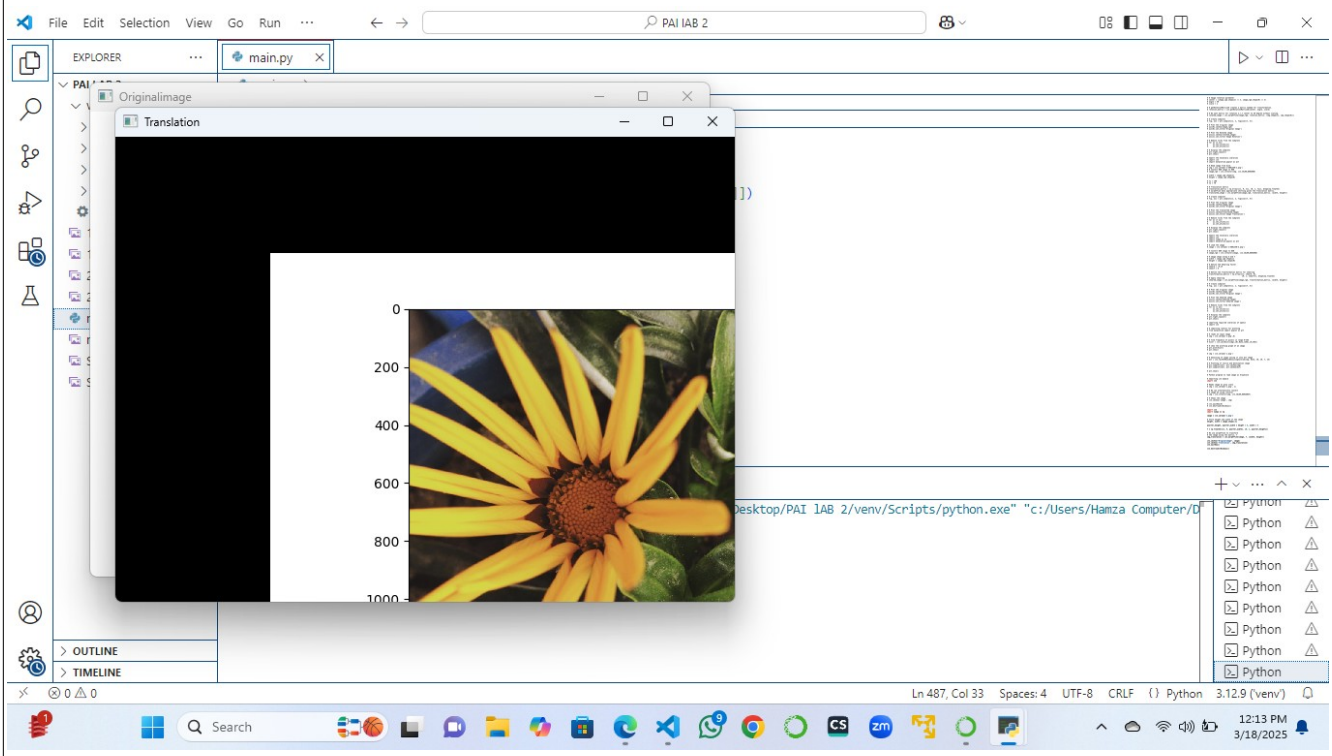


Image Translation:

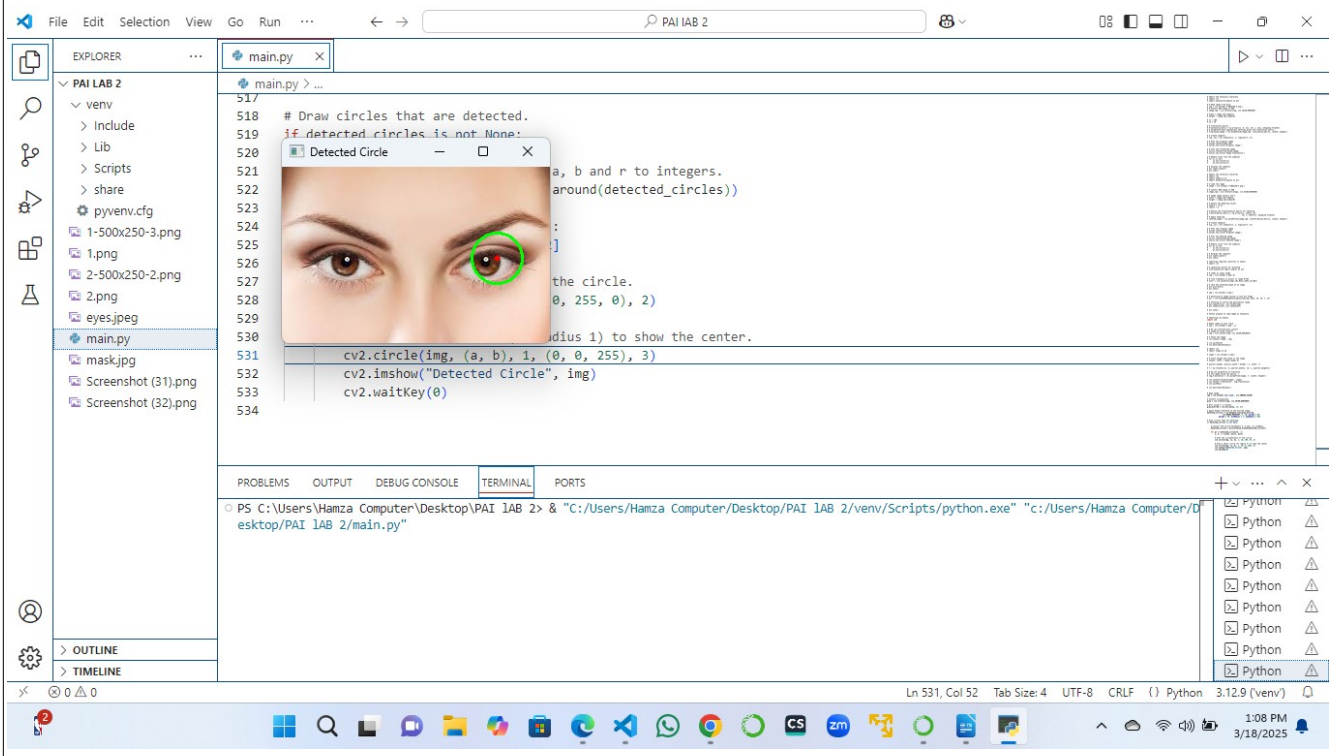
```
import cv2
import numpy as np

image = cv2.imread('1.png')

# Store height and width of the image
height, width = image.shape[:2]

quarter_height, quarter_width = height / 4, width / 4

T = np.float32([[1, 0, quarter_width], [0, 1, quarter_height]])

# We use warpAffine to transform
# the image using the matrix, T
img_translation = cv2.warpAffine(image, T, (width, height))

cv2.imshow("Originalimage", image)
cv2.imshow('Translation', img_translation)
cv2.waitKey()

cv2.destroyAllWindows()
```



Circle Detection:

```
img = cv2.imread('eyes.jpeg', cv2.IMREAD_COLOR)


# Convert to grayscale.
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Blur using 3 * 3 kernel.
gray_blurred = cv2.blur(gray, (3, 3))

# Apply Hough transform on the blurred image.
detected_circles = cv2.HoughCircles(gray_blurred,
                   cv2.HOUGH_GRADIENT, 1, 20, param1 = 50,
               param2 = 30, minRadius = 1, maxRadius = 40)

# Draw circles that are detected.
if detected_circles is not None:

    # Convert the circle parameters a, b and r to integers.
    detected_circles = np.uint16(np.around(detected_circles))

    for pt in detected_circles[0, :]:
```

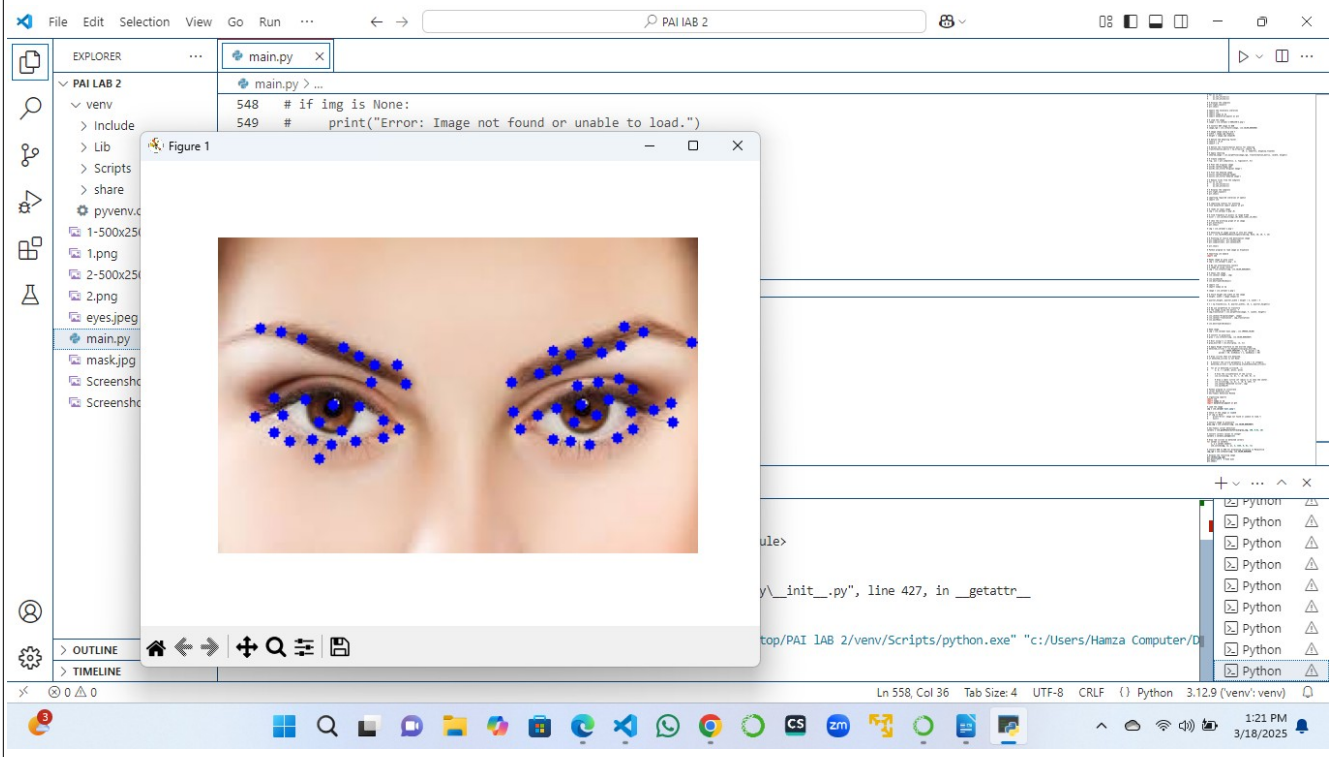
```
a, b, r = pt[0], pt[1], pt[2]

# Draw the circumference of the circle.
cv2.circle(img, (a, b), r, (0, 255, 0), 2)

# Draw a small circle (of radius 1) to show the center.
cv2.circle(img, (a, b), 1, (0, 0, 255), 3)
cv2.imshow("Detected Circle", img)
cv2.waitKey(0)
```



Corner Detection using Shi-Tomasi method:

```
import cv2

import numpy as np
import matplotlib.pyplot as plt

# Load the image
img = cv2.imread('eyes.jpeg')

# Convert image to grayscale
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
# Shi-Tomasi corner detection
corners = cv2.goodFeaturesToTrack(gray_img, 100, 0.01, 10)

# Convert corners values to integer
corners = corners.astype(int)

# Draw red circles on detected corners
for corner in corners:
    x, y = corner.ravel()
    cv2.circle(img, (x, y), 3, (255, 0, 0), -1)

# Convert BGR to RGB for displaying correctly in Matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display the resulting image
plt.imshow(img_rgb)
plt.axis("off")  # Hide axes
plt.show()
```

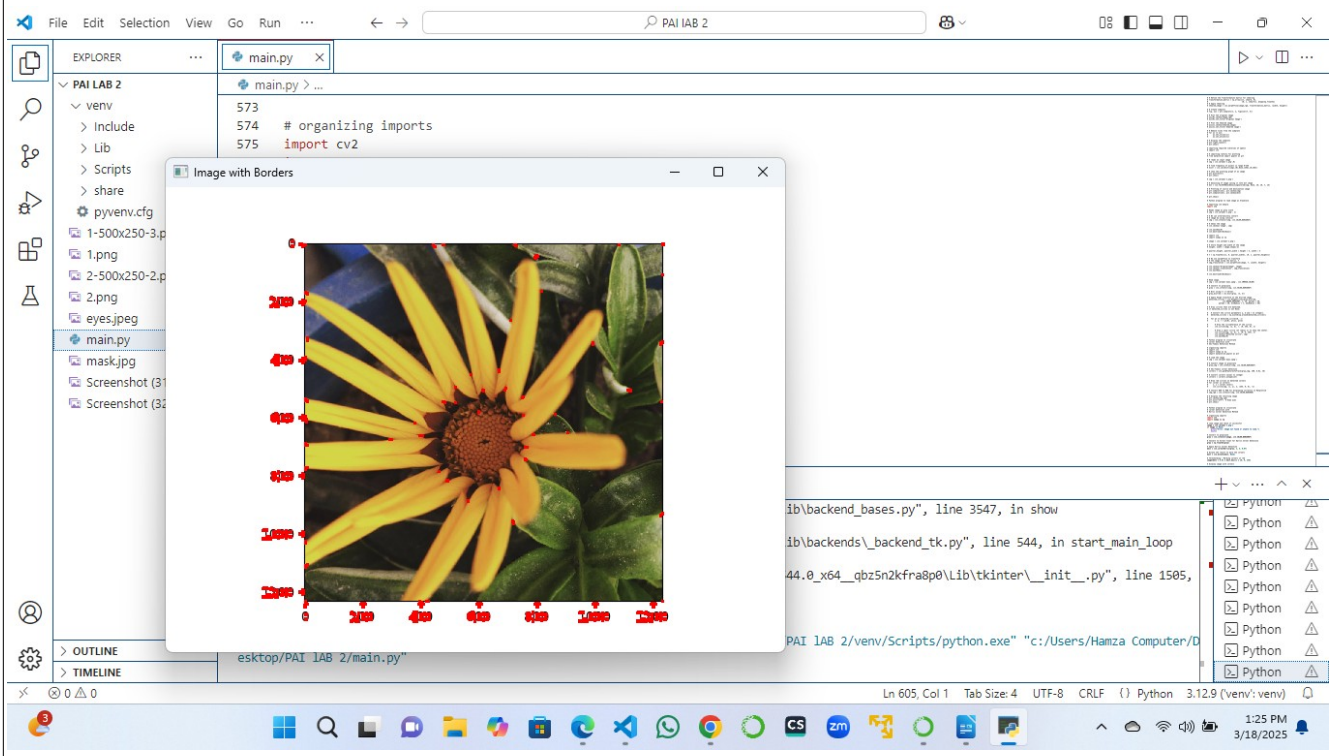


Corner Detection using Harris method:

```
import cv2

import numpy as np
```

Programming for Artificial Intelligence

```
# Load image and check if successful
image = cv2.imread('1.png')
# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Convert to 32-bit float for Harris Corner Detection
gray = np.float32(gray)
# Apply Harris Corner Detection
dest = cv2.cornerHarris(gray, 2, 5, 0.07)
# Dilate the result to mark the corners
dest = cv2.dilate(dest, None)
# Thresholding - Marking corners in red
image[dest > 0.01 * dest.max()] = [0, 0, 255]
# Display image with corners
cv2.imshow('Image with Borders', image)

# Wait for a key press and close the window
cv2.waitKey(0)
cv2.destroyAllWindows()
```

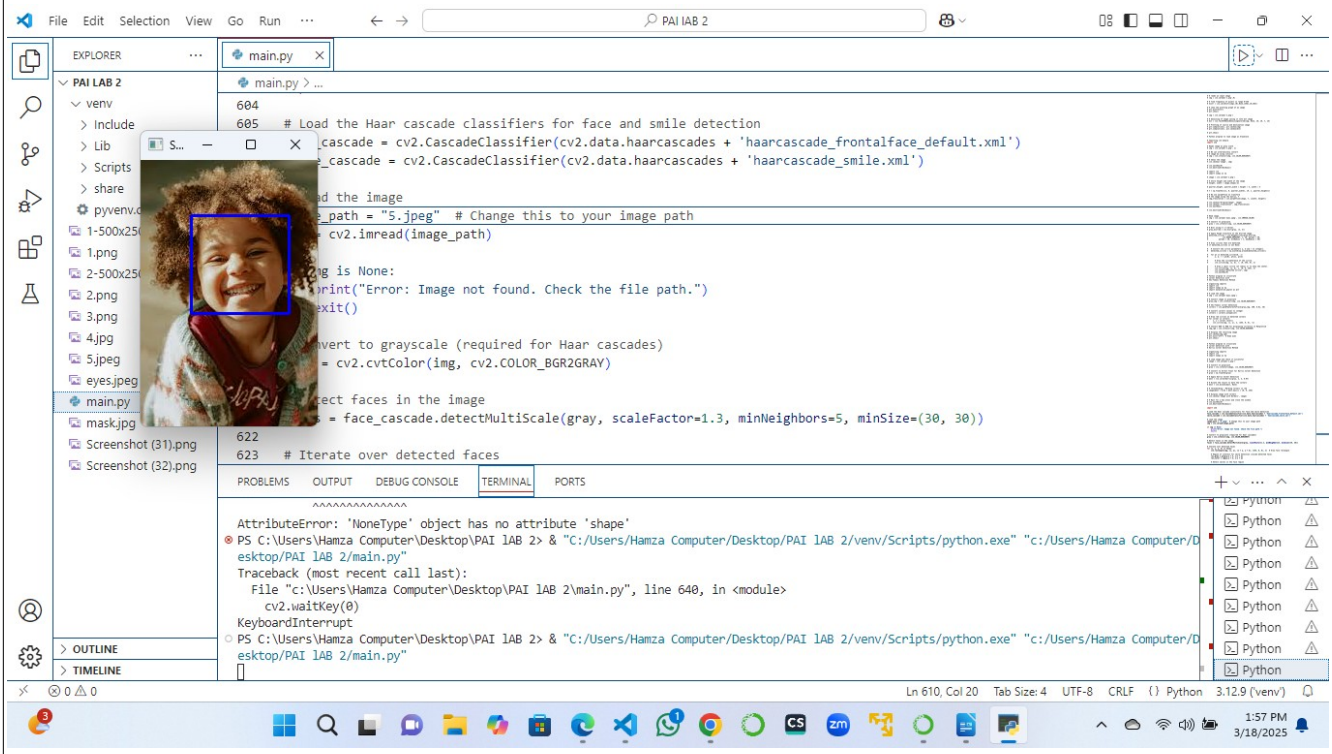


Smile Detection:

```
import cv2
```

Programming for Artificial Intelligence

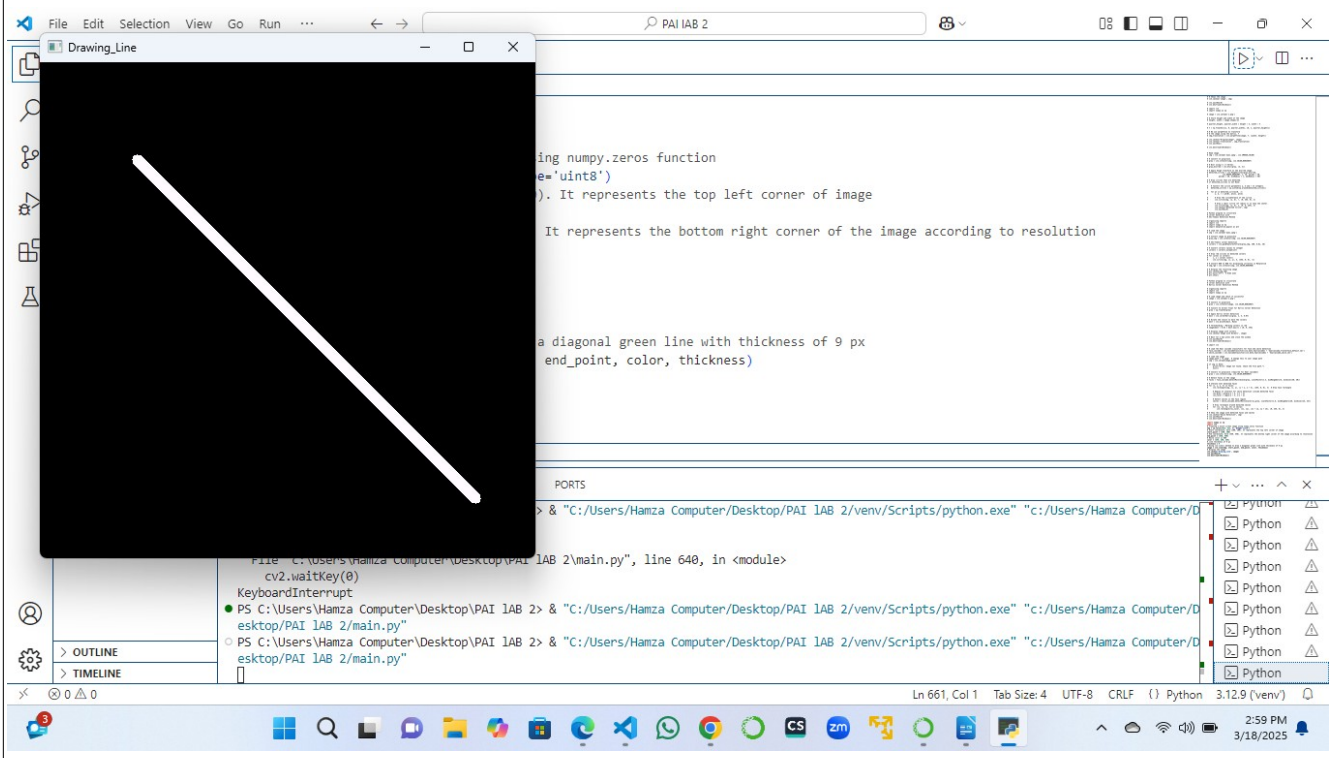
```
# Load the Haar cascade classifiers for face and smile detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
smile_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_smile.xml')

# Load the image
image_path = "5.jpeg"  # Change this to your image path
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not found. Check the file path.")
    exit()

# Convert to grayscale (required for Haar cascades)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5, minSize=(30, 30))

# Iterate over detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)  # Draw face rectangle

    # Region of interest for smile detection (inside detected face)
    roi_gray = gray[y:y + h, x:x + w]
    roi_color = img[y:y + h, x:x + w]

    # Detect smiles in the face region
    smiles = smile_cascade.detectMultiScale(roi_gray, scaleFactor=1.8, minNeighbors=20,
minSize=(25, 25))

    # Draw rectangle around detected smiles
    for (sx, sy, sw, sh) in smiles:
        cv2.rectangle(roi_color, (sx, sy), (sx + sw, sy + sh), (0, 255, 0), 2)

# Show the image with detected faces and smiles
cv2.imshow("Smile Detection", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Drawing a line:

```
import numpy as np

import cv2
# Creating a black screen image using numpy.zeros function
Img = np.zeros((512, 512, 3), dtype='uint8')
# Start coordinate, here (100, 100). It represents the top left corner of image
start_point = (100, 100)
# End coordinate, here (450, 450). It represents the bottom right corner of the image according to resolution
end_point = (450, 450)
# White color in BGR
color = (255, 250, 255)
# Line thickness of 9 px
thickness = 9
# Using cv2.line() method to draw a diagonal green line with thickness of 9 px
image = cv2.line(Img, start_point, end_point, color, thickness)
# Display the image
cv2.imshow('Drawing_Line', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

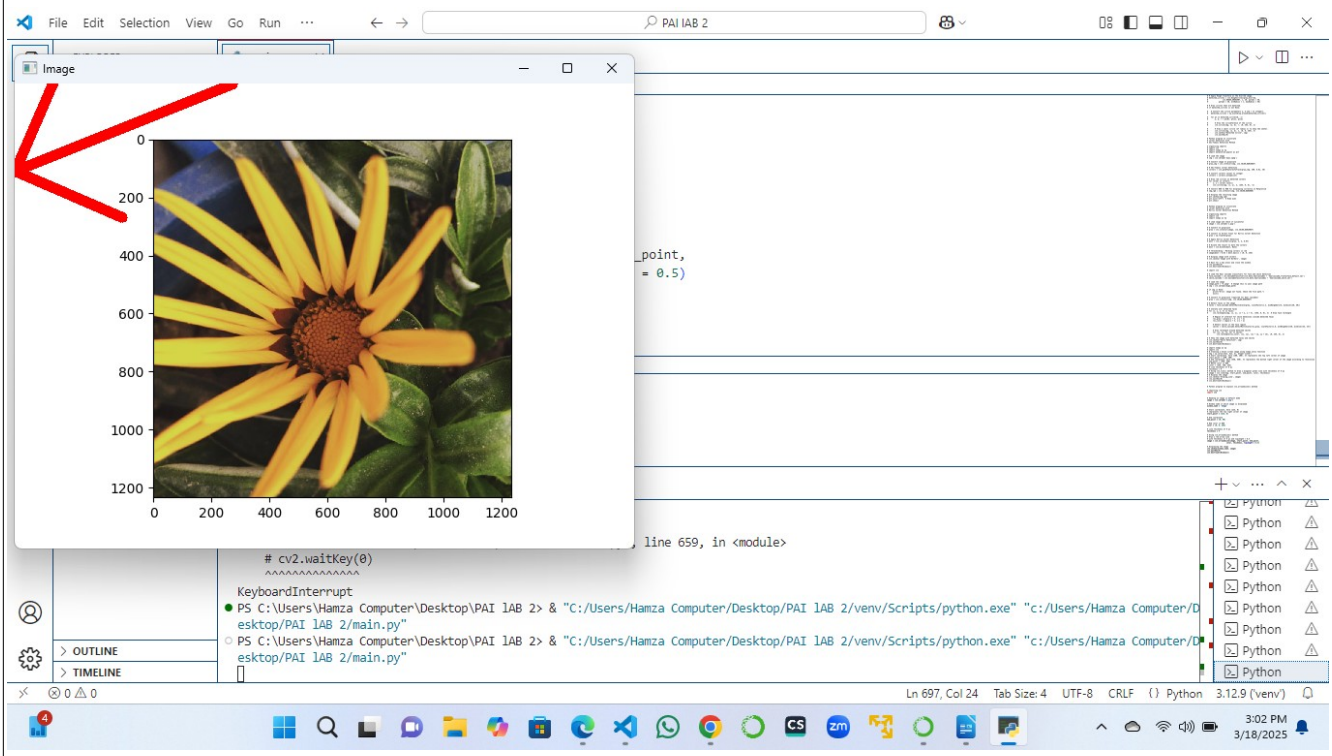
Programming for Artificial Intelligence



Drawing an arrow:

```
import cv2

# Reading an image in default mode
image = cv2.imread('1.png')

# Window name in which image is displayed
window_name = 'Image'

# Start coordinate, here (225, 0)
# represents the top right corner of image
start_point = (225, 0)

# End coordinate
end_point = (0, 90)

# Red color in BGR
color = (0, 0, 255)

# Line thickness of 9 px
thickness = 9
```

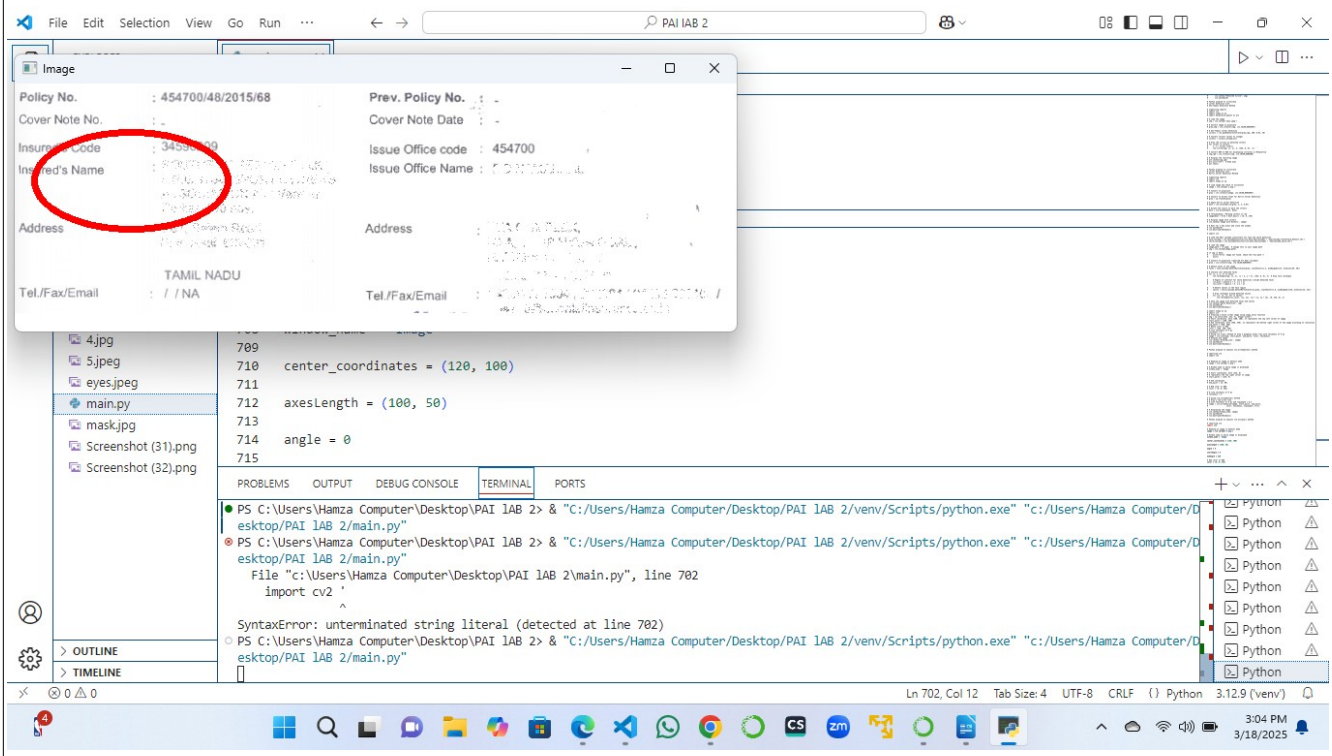
```
# Using cv2.arrowedLine() method
# Draw a red arrow line
# with thickness of 9 px and tipLength = 0.5
image = cv2.arrowedLine(image, start_point, end_point,
                        color, thickness, tipLength = 0.5)

# Displaying the image
cv2.imshow(window_name, image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Drawing an elipse:

```
import cv2

# Reading an image in default mode
image = cv2.imread('3.png')

# Window name in which image is displayed
window_name = 'Image'

center_coordinates = (120, 100)

axesLength = (100, 50)

angle = 0
```

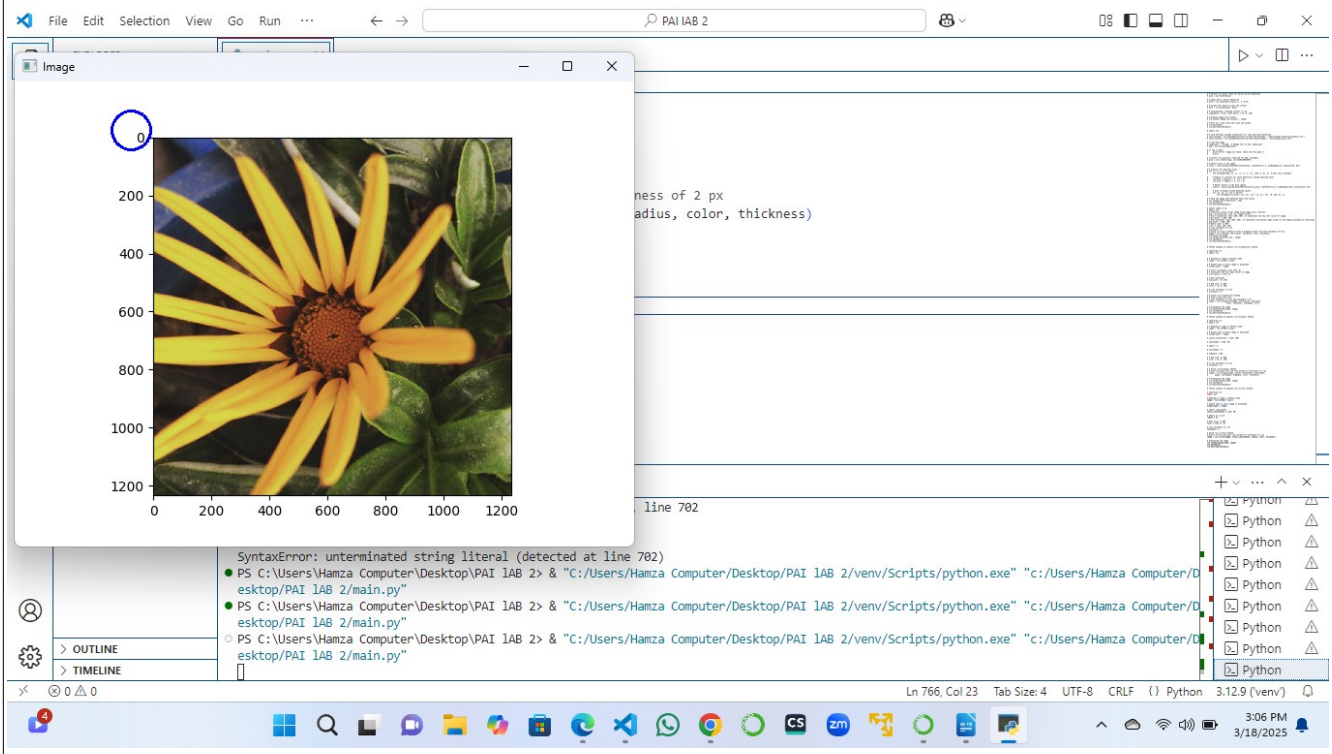
```
startAngle = 0

endAngle = 360

# Red color in BGR
color = (0, 0, 255)

# Line thickness of 5 px
thickness = 5

# Using cv2.ellipse() method
# Draw a ellipse with red line borders of thickness of 5 px
image = cv2.ellipse(image, center_coordinates, axesLength,
        angle, startAngle, endAngle, color, thickness)

# Displaying the image
cv2.imshow(window_name, image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Drawing a circle:

```
import cv2
```

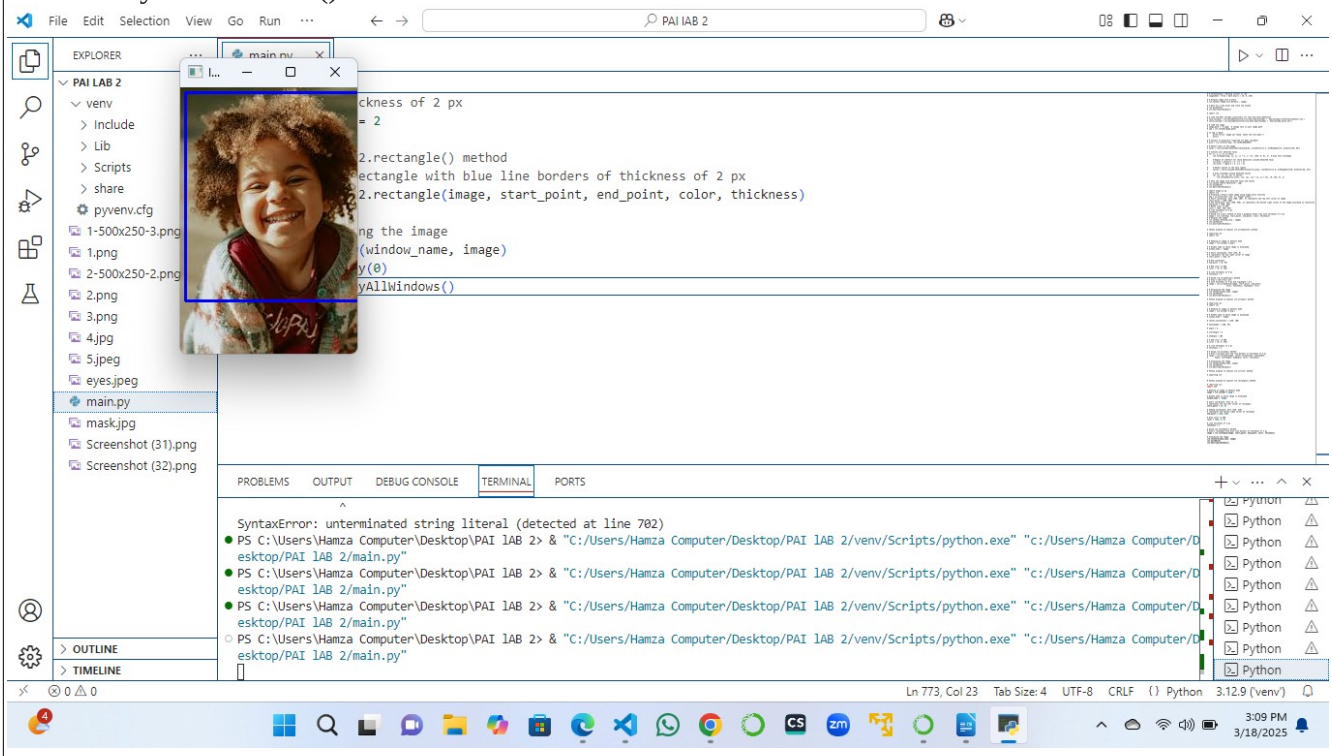
```
# Reading an image in default mode
image = cv2.imread('1.png')
# Window name in which image is displayed
window_name = 'Image'
# Center coordinates
center_coordinates = (120, 50)
# Radius of circle
radius = 20
# Blue color in BGR
color = (255, 0, 0)
# Line thickness of 2 px
thickness = 2
# Using cv2.circle() method
# Draw a circle with blue line borders of thickness of 2 px
image = cv2.circle(image, center_coordinates, radius, color, thickness)
# Displaying the image
cv2.imshow(window_name, image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Drawing a rectangle:

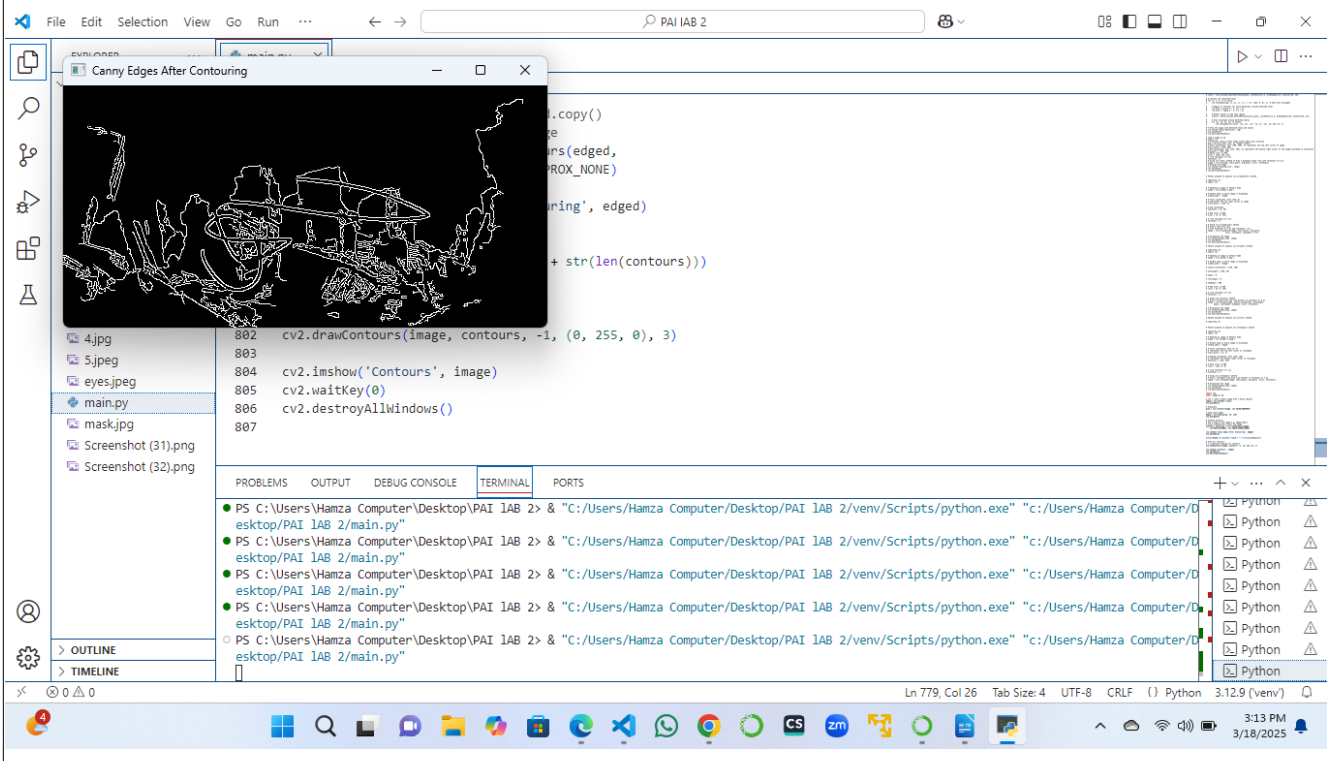
Programming for Artificial Intelligence

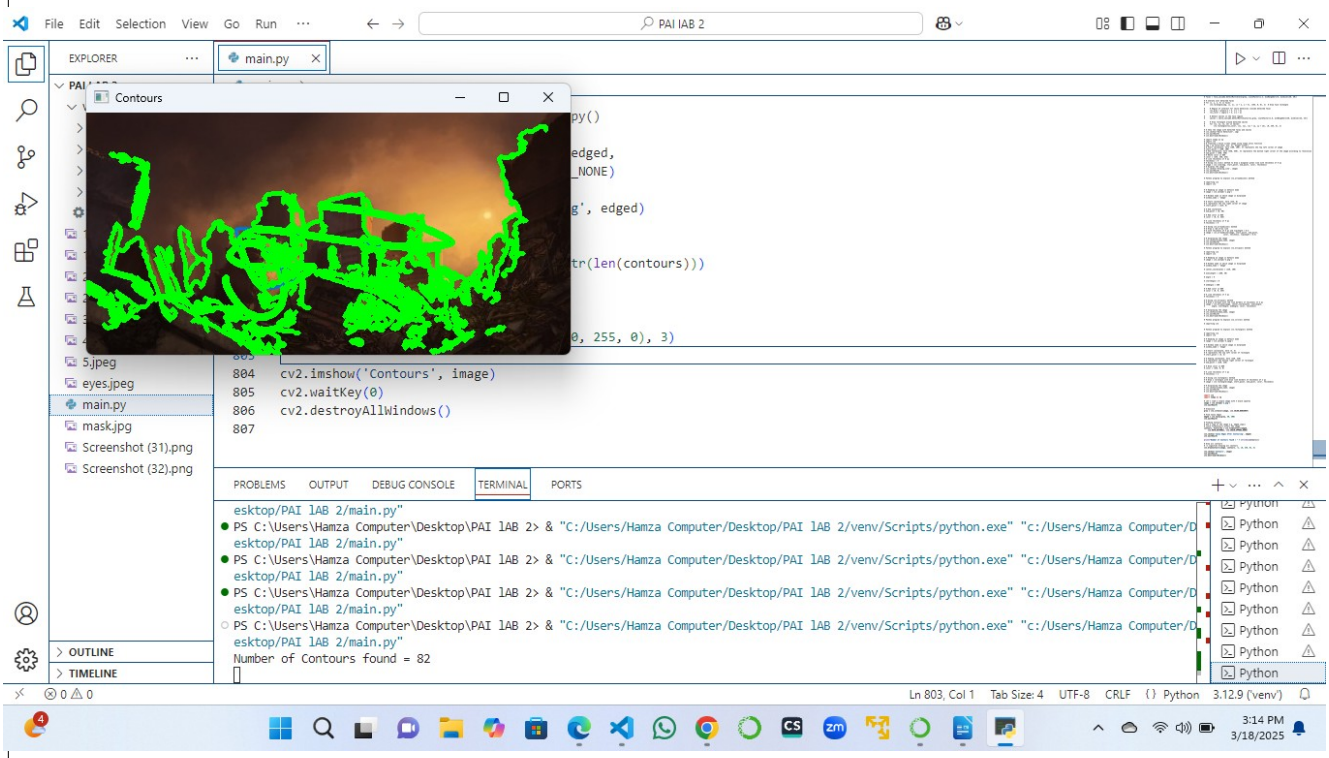
```
import cv2

# Reading an image in default mode
image = cv2.imread('5.jpeg')
# Window name in which image is displayed
window_name = 'Image'
# Start coordinate, here (5, 5)
# represents the top left corner of rectangle
start_point = (5, 5)
# Ending coordinate, here (220, 220)
# represents the bottom right corner of rectangle
end_point = (220, 220)
# Blue color in BGR
color = (255, 0, 0)
# Line thickness of 2 px
thickness = 2
# Using cv2.rectangle() method
# Draw a rectangle with blue line borders of thickness of 2 px
image = cv2.rectangle(image, start_point, end_point, color, thickness)
#Displaying the image
cv2.imshow(window_name, image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Find and draw circles and contours:

```
import cv2
import numpy as np
# Let's load a simple image with 3 black squares
image = cv2.imread('2.png')
cv2.waitKey(0)
# Grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Find Canny edges
edged = cv2.Canny(gray, 30, 200)
cv2.waitKey(0)
# Finding Contours
# Use a copy of the image e.g. edged.copy()
# since findContours alters the image
contours, hierarchy = cv2.findContours(edged,
    cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
cv2.imshow('Canny Edges After Contouring', edged)
cv2.waitKey(0)
print("Number of Contours found = " + str(len(contours)))
# Draw all contours
# -1 signifies drawing all contours
cv2.drawContours(image, contours, -1, (0, 255, 0), 3)
cv2.imshow('Contours', image)
```

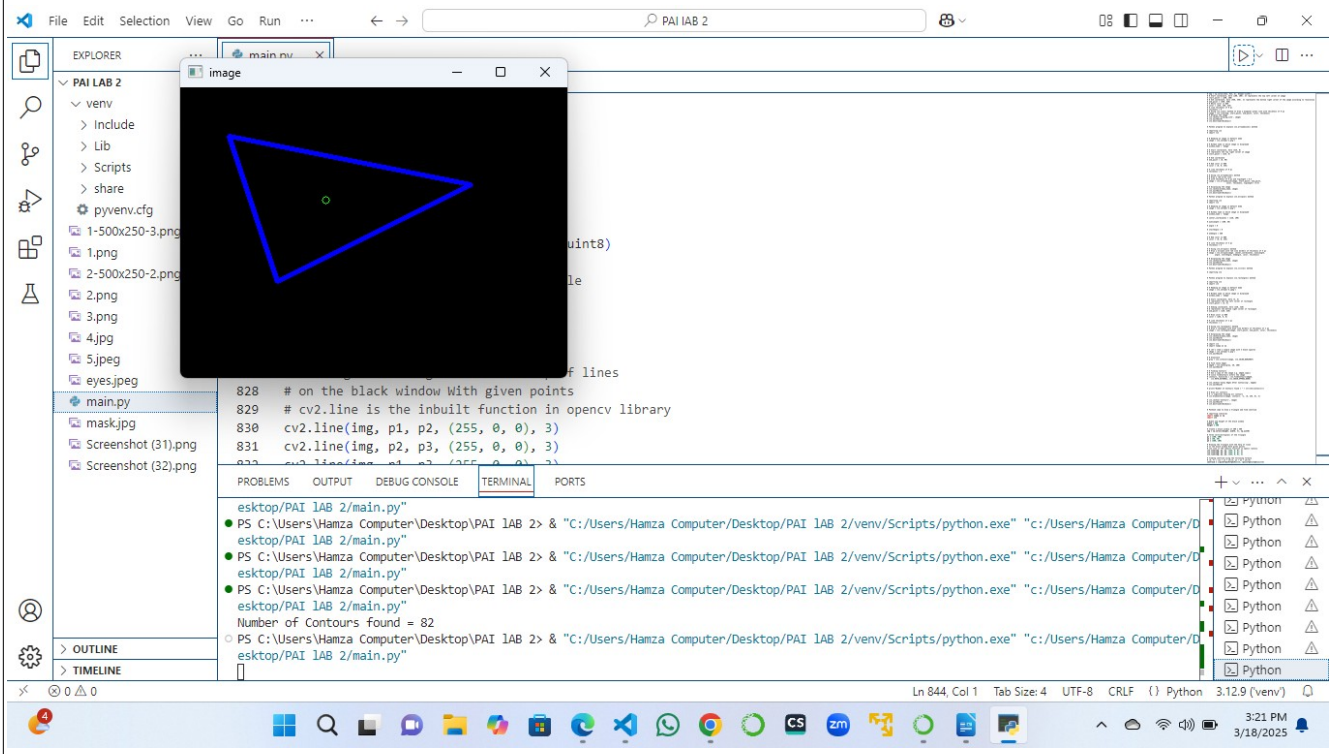




Draw a triangle with centroid:

```
import numpy as np

import cv2
# Width and height of the black window
width = 400
height = 300
# Create a black window of 400 x 300
img = np.zeros((height, width, 3), np.uint8)
# Three vertices(tuples) of the triangle
p1 = (100, 200)
p2 = (50, 50)
p3 = (300, 100)
# Drawing the triangle with the help of lines
# on the black window With given points
# cv2.line is the inbuilt function in opencv library
cv2.line(img, p1, p2, (255, 0, 0), 3)
cv2.line(img, p2, p3, (255, 0, 0), 3)
cv2.line(img, p1, p3, (255, 0, 0), 3)
# finding centroid using the following formula
# (X, Y) = (x1 + x2 + x3//3, y1 + y2 + y3//3)
centroid = ((p1[0]+p2[0]+p3[0])//3, (p1[1]+p2[1]+p3[1])//3)
```

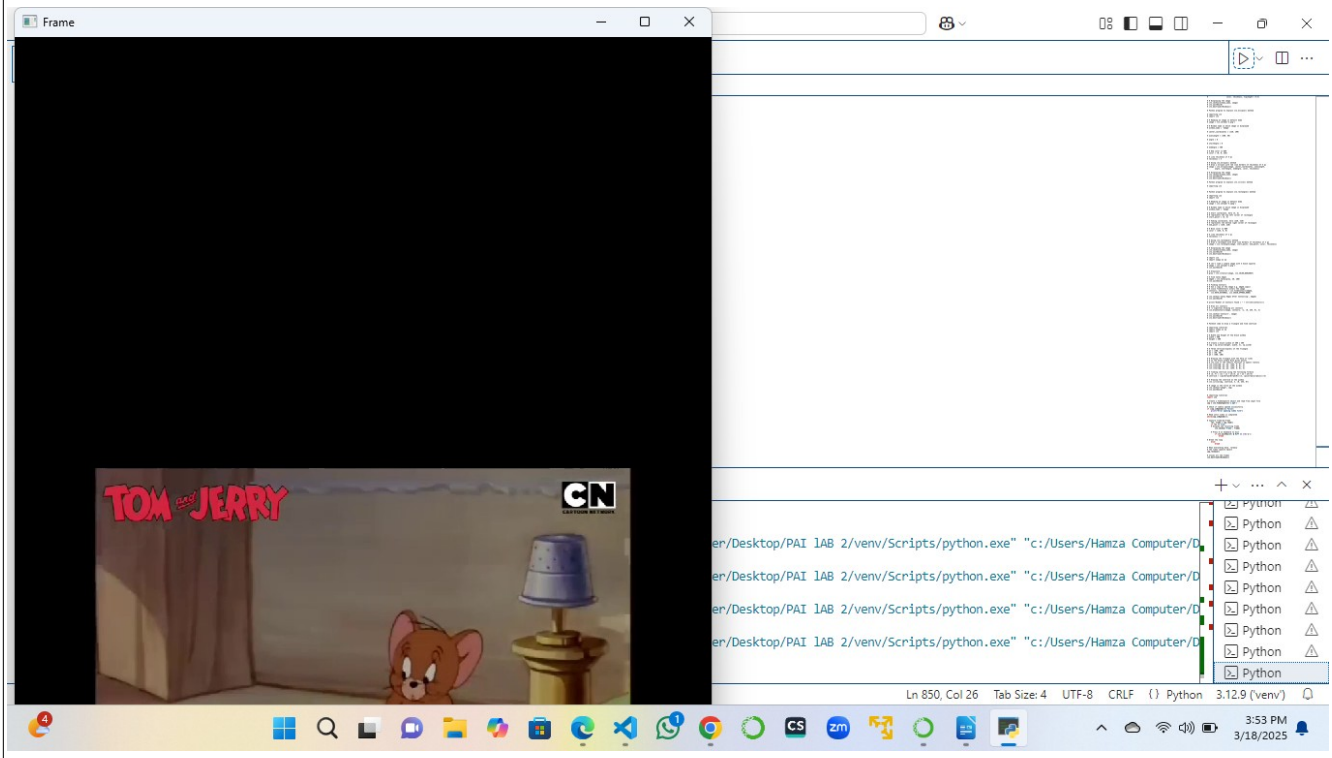
```
# Drawing the centroid on the window
cv2.circle(img, centroid, 4, (0, 255, 0))
# image is the title of the window
cv2.imshow("image", img)
cv2.waitKey(0)
```



Playing a video:

```
import cv2

# Create a VideoCapture object and read from input file
cap = cv2.VideoCapture('1.mp4')

# Check if camera opened successfully
if (cap.isOpened()== False):
    print("Error opening video file")

# Read until video is completed
while(cap.isOpened()):

# Capture frame-by-frame
    ret, frame = cap.read()
    if ret == True:
    # Display the resulting frame
        cv2.imshow('Frame', frame)
```

```
    # Press Q on keyboard to exit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

# Break the loop
    else:
        break

# When everything done, release
# the video capture object
cap.release()

# Closes all the frames
cv2.destroyAllWindows()
```



Extracting images from video:

```
import cv2
import os

# Read the video from specified path
cam = cv2.VideoCapture("1.mp4")
```

```
try:

    # creating a folder named data
    if not os.path.exists('data'):
        os.makedirs('data')

# if not created then raise error
except OSError:
    print ('Error: Creating directory of data')

# frame
currentframe = 0

while(True):

    # reading from frame
    ret,frame = cam.read()

    if ret:
        # if video is still left continue creating images
        name = './data/frame' + str(currentframe) + '.jpg'
        print ('Creating...' + name)

        # writing the extracted images
        cv2.imwrite(name, frame)

        # increasing counter so that it will
        # show how many frames are created
        currentframe += 1
    else:
        break

# Release all space and windows once done
cam.release()
cv2.destroyAllWindows()
```

Programming for Artificial Intelligence