Pixels
EVERY PIXEL IS KNOWLEDGE

فيما أفتناه

Computer Vision

Represented by : Basma Ahmed .

Pixels
EVERY PIXEL IS KNOWLEDGE

The content for today:

1. Introduction to Computer Vision.
2. OpenCV Basics.
3. Reading and Writing Images.
4. Drawing Geometric Shapes on Images.
5. Creating Graphics with OpenCV.
6. Setting Camera Parameters.
7. Bitwise Operations on Images.

What is Computer Vision?

1. What is Computer Vision?

❑ Computer Vision is a field of artificial intelligence (AI) that enables computers to interpret and process visual information, such as images or videos, like the human brain.

❑ It focuses on tasks like image recognition, object detection, segmentation, and scene understanding.

1.2 Applications of Computer Vision:

- **Color Detection :identify colors in an image or video** by analyzing **RGB or HSV values.**

- **Edge Detection in Images .**

- **Face Detection in Images or Video** : detect **faces in images or videos using** Haar Cascade Classifier.

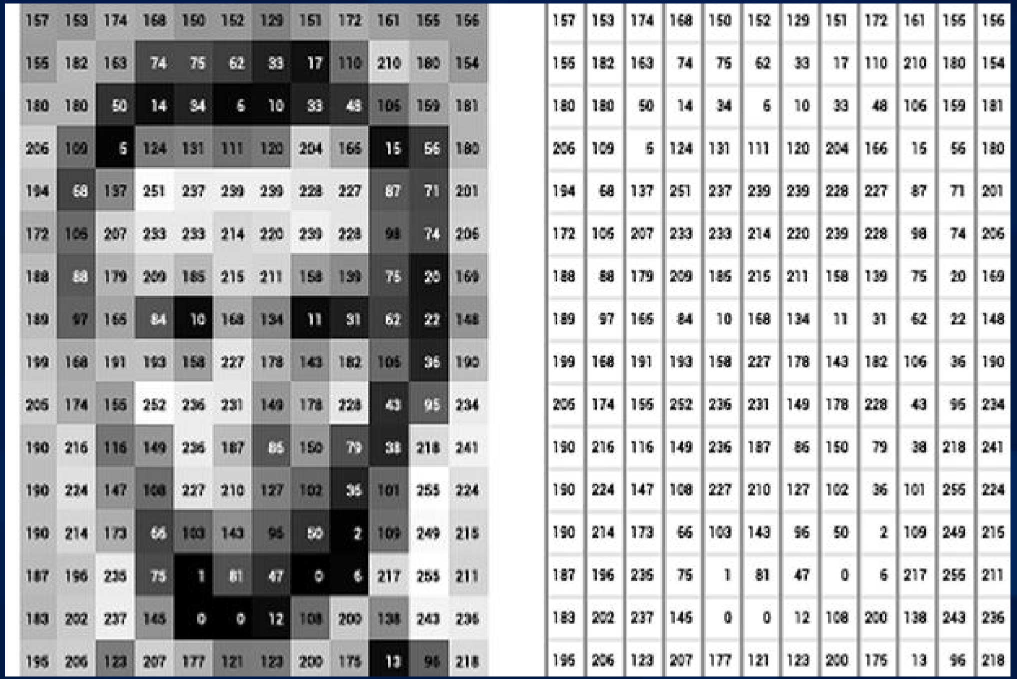
- **Object Tracking : identify and follow a specific object across video frames.**

1.3 What is an image ?

An image is represented by its dimensions (height and width) based on the number of pixels.

Example :

- if the dimensions of an image are 500 x 400
- (width x height), the total number of pixels in the image is 200000.



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

1.4 Images & Pixels :

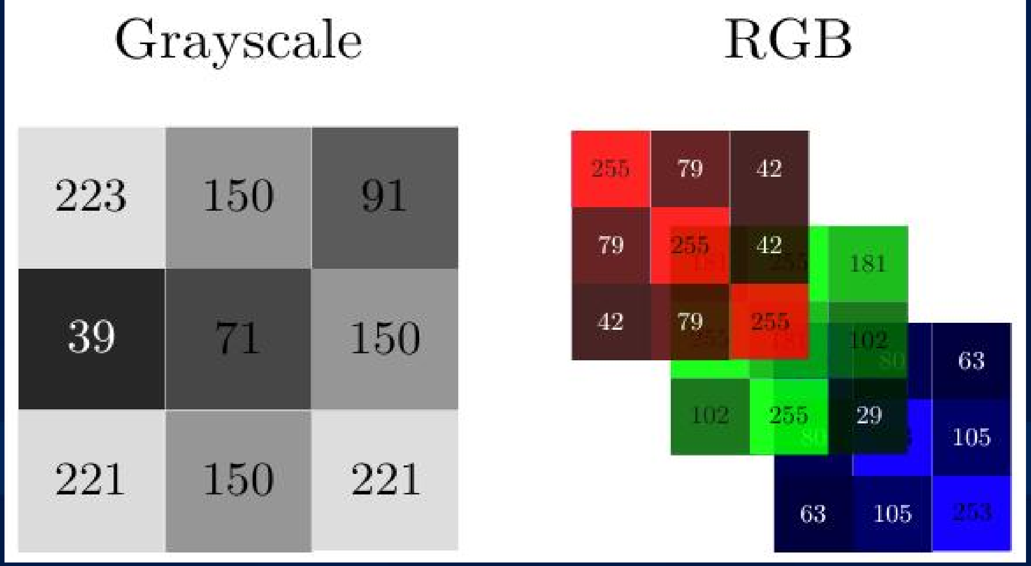
*Grayscale images contain only shades of gray, represented by a single intensity value per pixel, while RGB images consist of **three-color channels** (Red, Green, Blue) per pixel, allowing for a wider range of colors.*

Grayscale :

each pixel can take
a value from (0 -> 255)
0 -> very dark , 255 -> very bright.

RGB :

each pixel can take
three values for each color from
(Red , Green , Blue).



Grayscale

223	150	91
39	71	150
221	150	221

RGB

2. What is OpenCV?

- OpenCV (Open-Source Computer Vision Library) is an open-source library designed for real-time computer vision tasks.

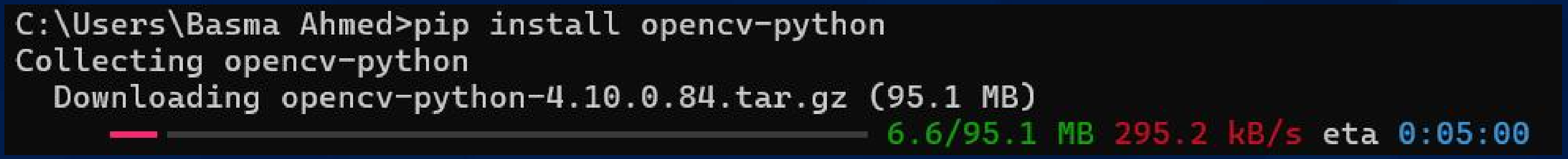
- Written in C++ but also provides Python bindings.

Features of OpenCV:

- ➢ Image processing (reading, writing, editing images).
- ➢ Real-time video processing.
- ➢ Machine learning tools for computer vision.

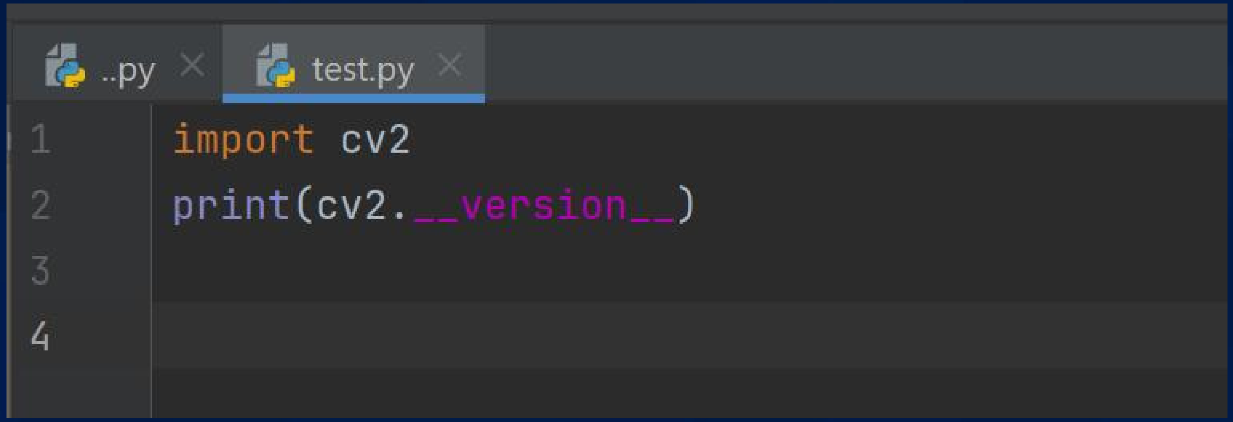
2.1 Setting Up OpenCV:

- Install OpenCV via Python's **"pip install OpenCV-python"** command.



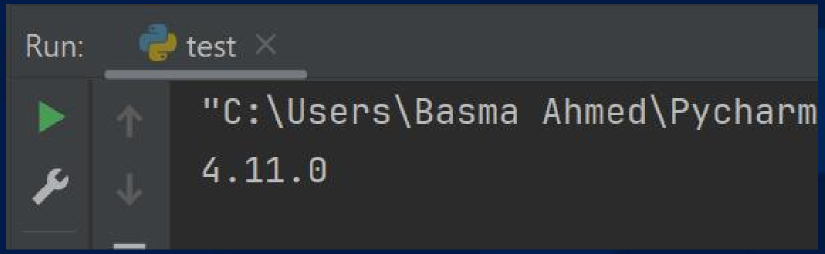
```
C:\Users\Basma Ahmed>pip install opencv-python
Collecting opencv-python
  Downloading opencv-python-4.10.0.84.tar.gz (95.1 MB)
                                                 6.6/95.1 MB 295.2 kB/s eta 0:05:00
```

- Import it in your Python code as **import cv2.**



```
import cv2
print(cv2.__version__)


```

The Output :



```
"C:\Users\Basma Ahmed\Pycharm
4.11.0
```

3. Reading and Writing Images :

Key Functions:

1. cv2.imread(*image_name , *flags): Reads an image from a file.
   flag : 1 => colored img , 0 => grayscale img.

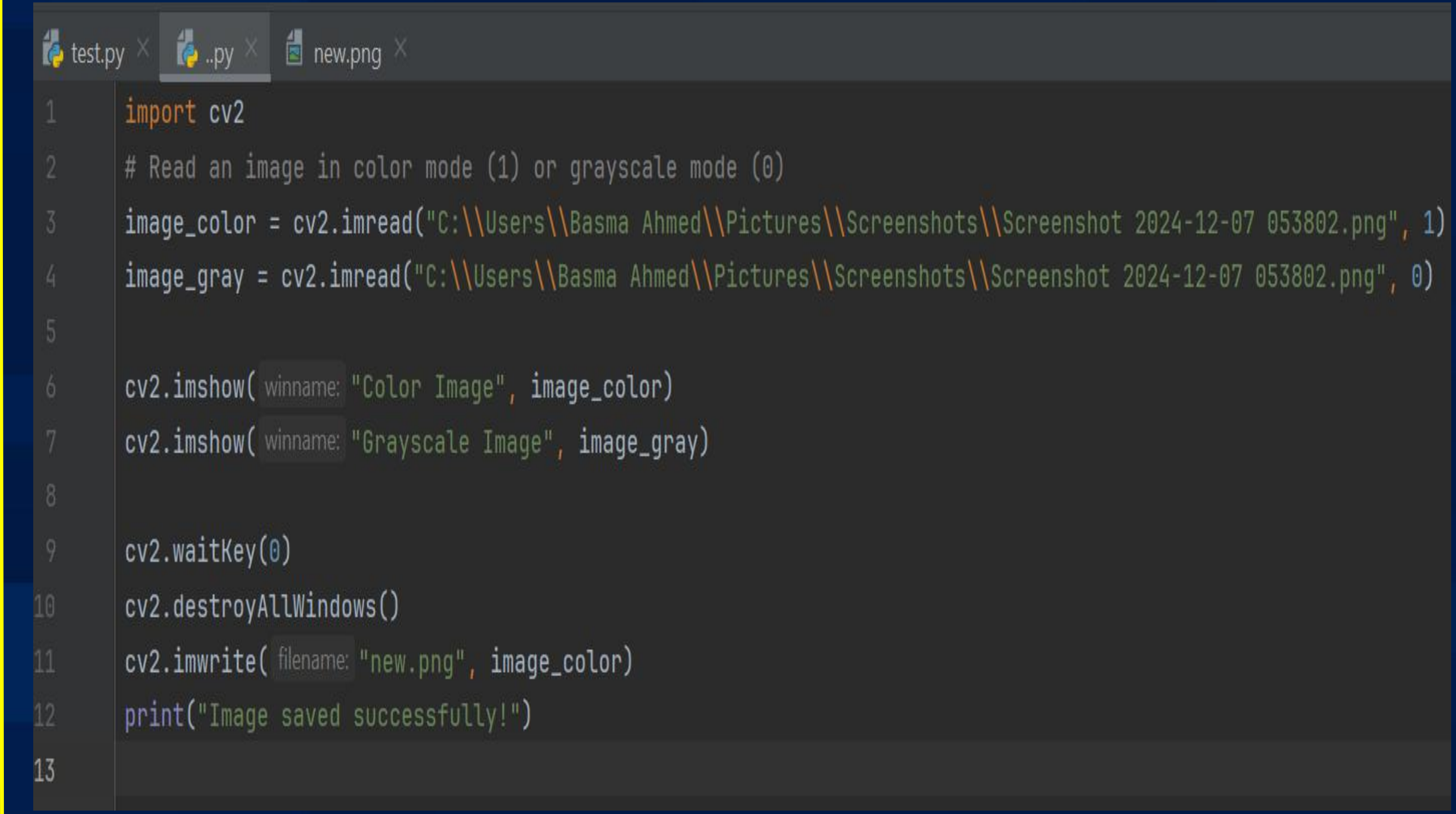
2. cv2.imwrite (filename, image): Saves an image to a file.

3. cv2.imshow (*window_title , *image).

4. cv2.waitKey (*milliseconds).

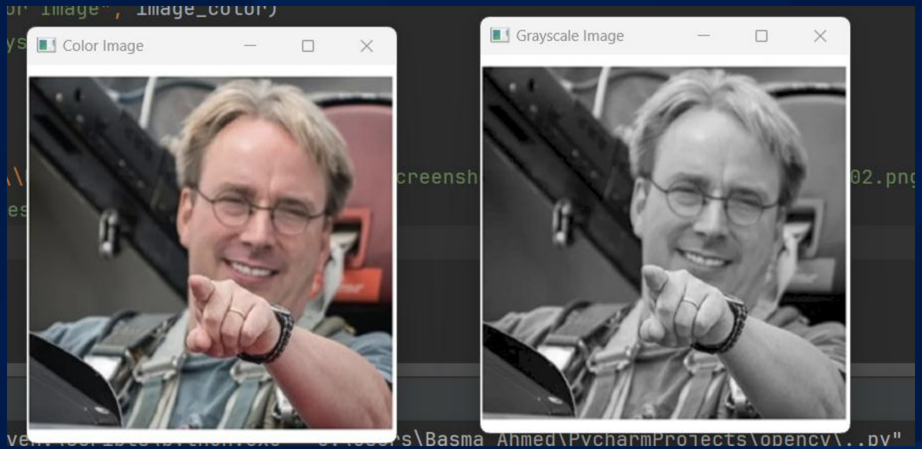
5. cv2.destroyAllWindows().

Example :



```
import cv2
# Read an image in color mode (1) or grayscale mode (0)
image_color = cv2.imread("C:\\Users\\Basma Ahmed\\Pictures\\Screenshots\\Screenshot 2024-12-07 053802.png", 1)
image_gray = cv2.imread("C:\\Users\\Basma Ahmed\\Pictures\\Screenshots\\Screenshot 2024-12-07 053802.png", 0)

cv2.imshow( winname: "Color Image", image_color)
cv2.imshow( winname: "Grayscale Image", image_gray)

cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite( filename: "new.png", image_color)
print("Image saved successfully!")

```

The Output :



Task 1:

- **Read an image** from a file using cv2.imread().

- **Convert the image to grayscale** and display both original & grayscale versions.

- **Save the grayscale image** using cv2.imwrite().

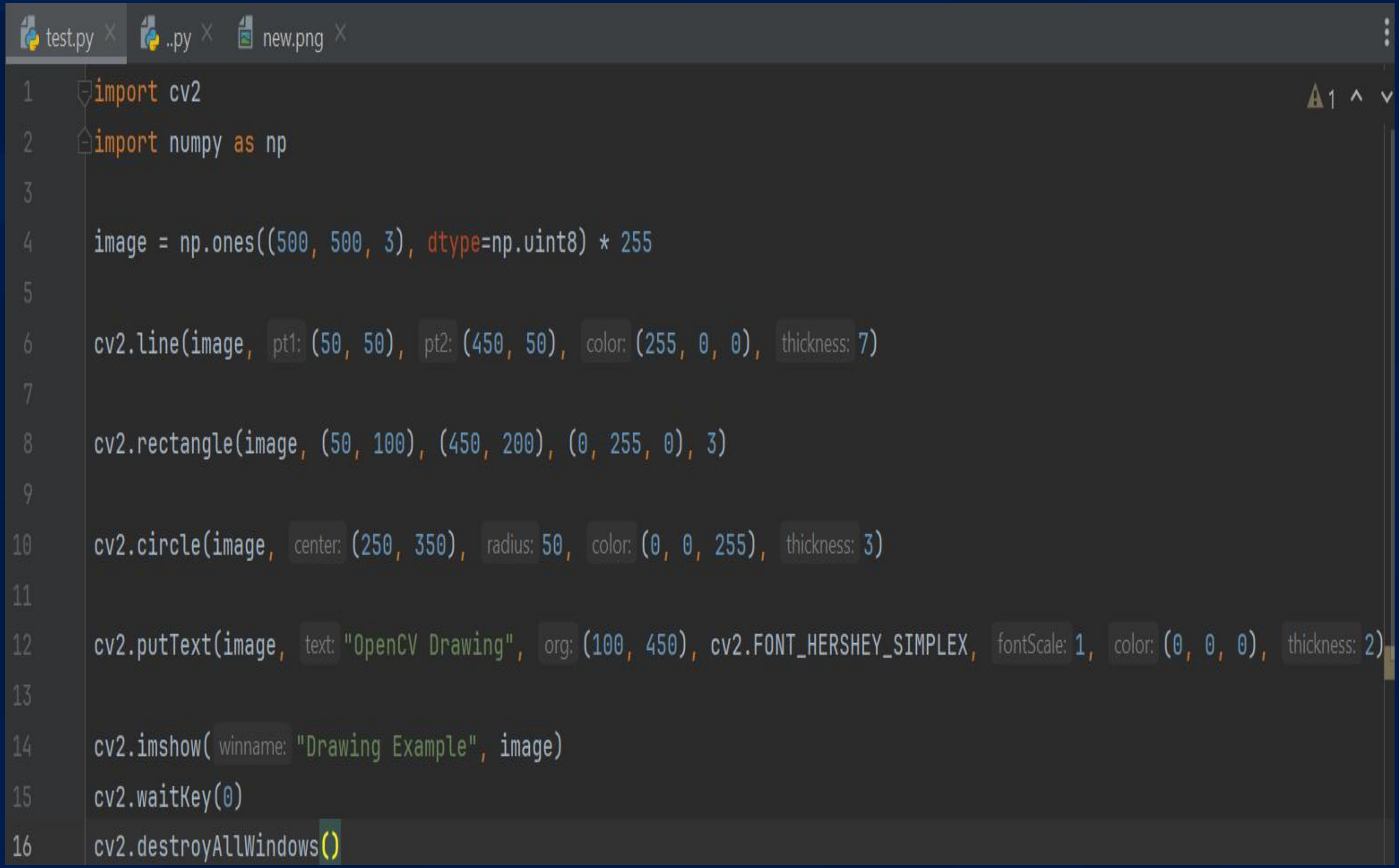
- **Wait for a key press** before closing the windows.

4. Drawing Geometric Shapes on Images

Key Functions:

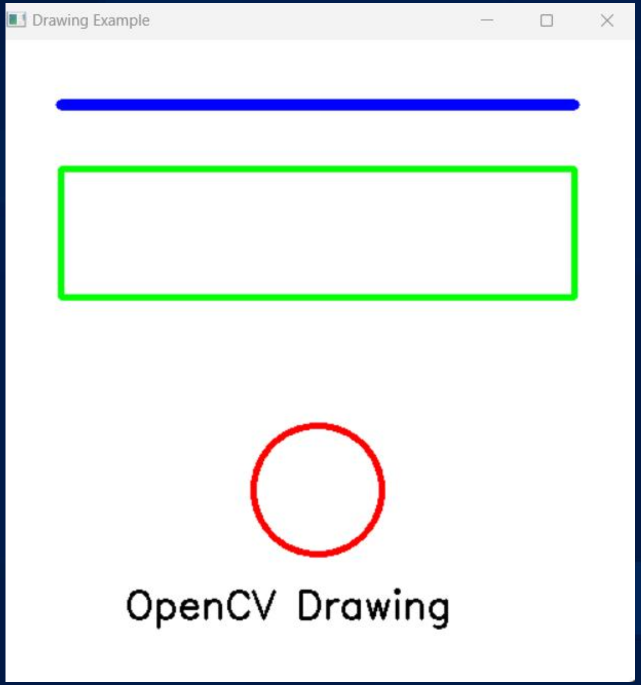
- cv2.line(img, pt1, pt2, color, thickness): Draws a line .
- cv2.rectangle(img, pt1, pt2, color, thickness): Draws a rectangle .
- cv2.circle(img, center, radius, color, thickness): Draws a circle .
- cv2.putText(img, text, position, font, size, color, thickness): Adds text .

4.1 Example on Drawing Geometric :



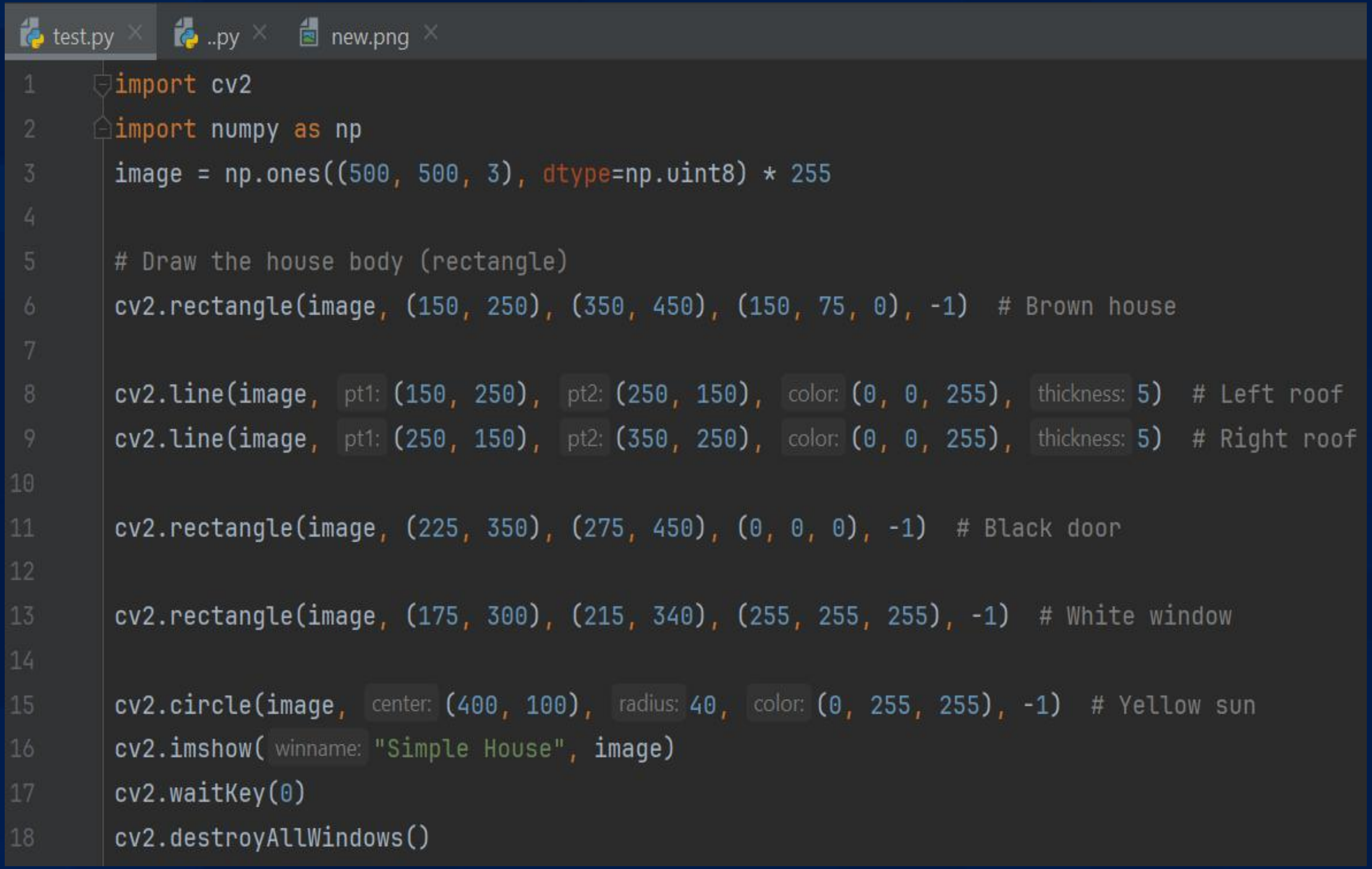
```
import cv2
import numpy as np

image = np.ones((500, 500, 3), dtype=np.uint8) * 255

cv2.line(image, pt1: (50, 50), pt2: (450, 50), color: (255, 0, 0), thickness: 7)

cv2.rectangle(image, (50, 100), (450, 200), (0, 255, 0), 3)

cv2.circle(image, center: (250, 350), radius: 50, color: (0, 0, 255), thickness: 3)

cv2.putText(image, text: "OpenCV Drawing", org: (100, 450), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 0, 0), thickness: 2)

cv2.imshow( winname: "Drawing Example", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



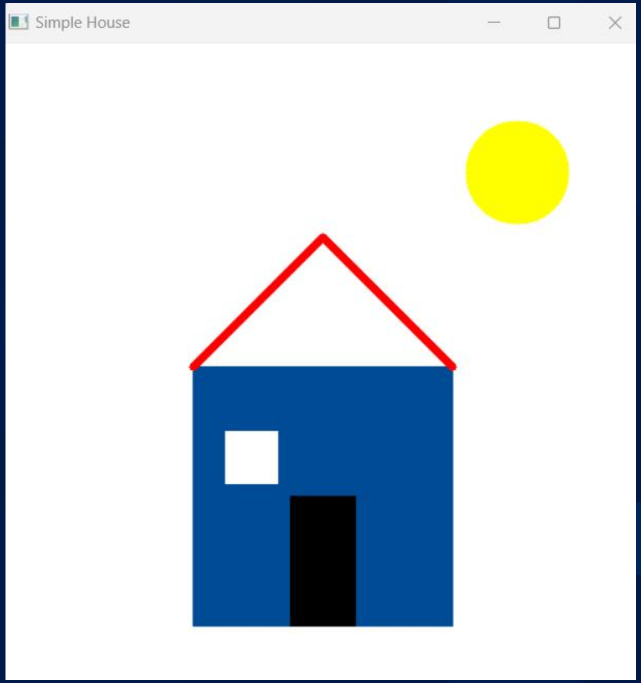
OpenCV Drawing

4.1 Example on Drawing Geometric :



```
import cv2
import numpy as np
image = np.ones((500, 500, 3), dtype=np.uint8) * 255

# Draw the house body (rectangle)
cv2.rectangle(image, (150, 250), (350, 450), (150, 75, 0), -1)  # Brown house

cv2.line(image, pt1: (150, 250), pt2: (250, 150), color: (0, 0, 255), thickness: 5)  # Left roof
cv2.line(image, pt1: (250, 150), pt2: (350, 250), color: (0, 0, 255), thickness: 5)  # Right roof

cv2.rectangle(image, (225, 350), (275, 450), (0, 0, 0), -1)  # Black door

cv2.rectangle(image, (175, 300), (215, 340), (255, 255, 255), -1)  # White window

cv2.circle(image, center: (400, 100), radius: 40, color: (0, 255, 255), -1)  # Yellow sun
cv2.imshow( winname: "Simple House", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

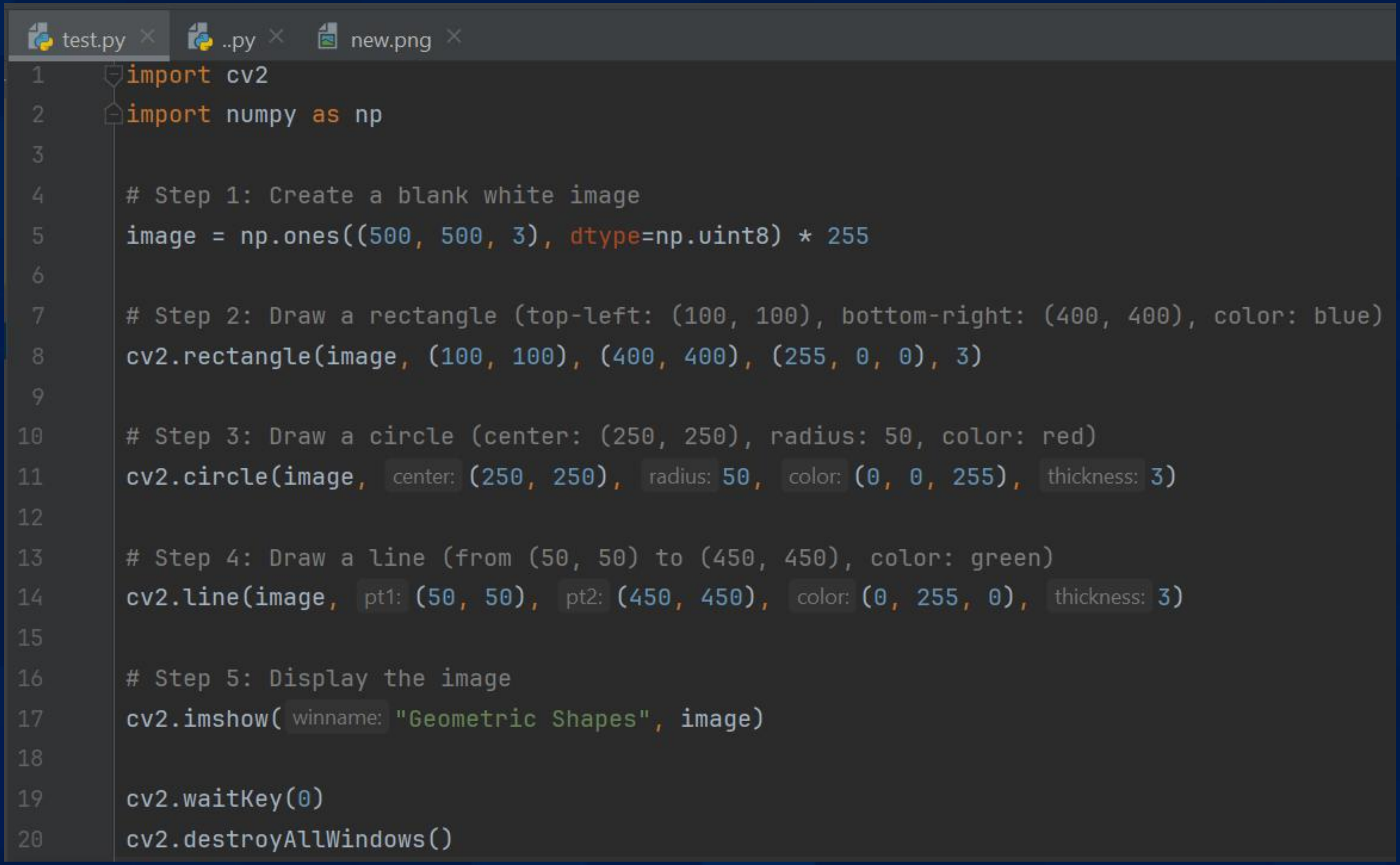
The Output :



Task 2:

Draw different geometric shapes
(a rectangle, a circle, and a line) on an image .

Solution

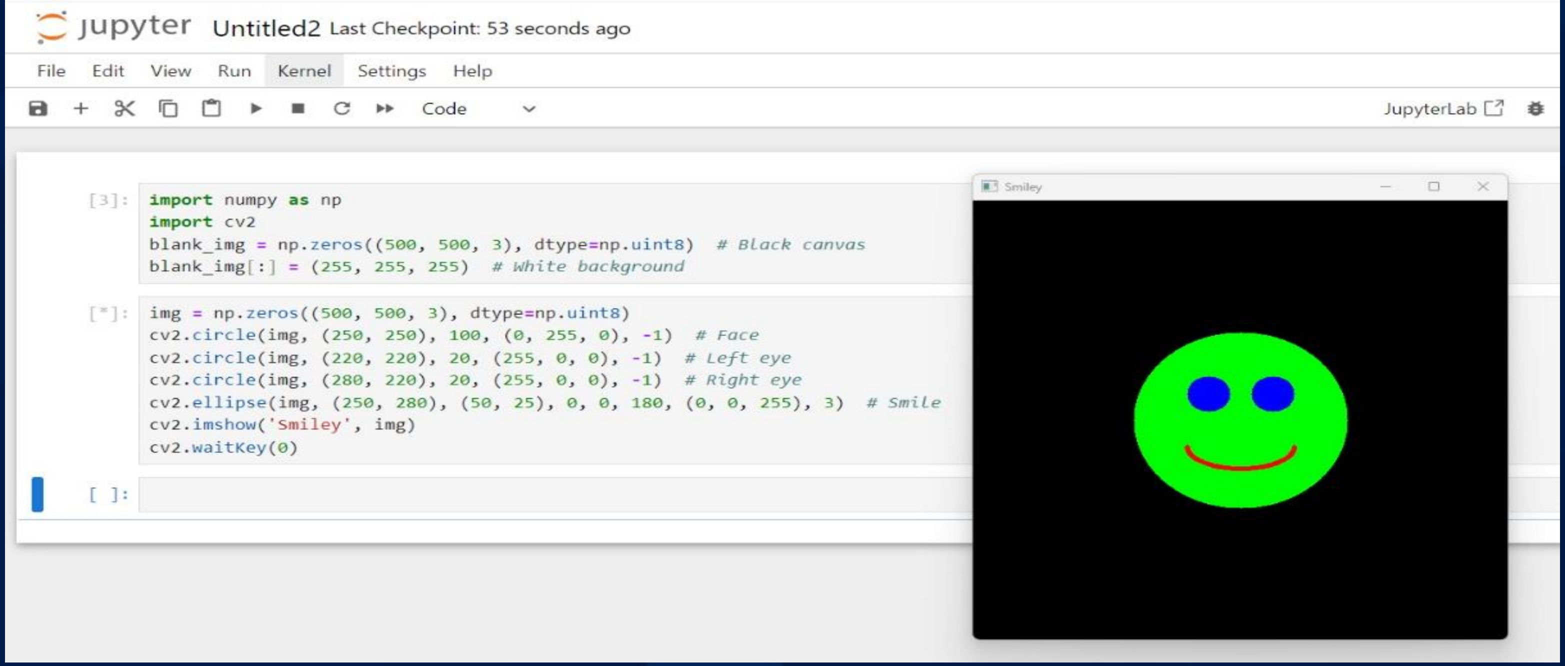


```
import cv2
import numpy as np

# Step 1: Create a blank white image
image = np.ones((500, 500, 3), dtype=np.uint8) * 255

# Step 2: Draw a rectangle (top-left: (100, 100), bottom-right: (400, 400), color: blue)
cv2.rectangle(image, (100, 100), (400, 400), (255, 0, 0), 3)

# Step 3: Draw a circle (center: (250, 250), radius: 50, color: red)
cv2.circle(image, center: (250, 250), radius: 50, color: (0, 0, 255), thickness: 3)

# Step 4: Draw a line (from (50, 50) to (450, 450), color: green)
cv2.line(image, pt1: (50, 50), pt2: (450, 450), color: (0, 255, 0), thickness: 3)

# Step 5: Display the image
cv2.imshow( winname: "Geometric Shapes", image)

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

5. Creating Graphics with OpenCV :

Creating a Blank Image:

- ➢ Use np.zeros() for a black image or fill it with specific values for different colors .
- ➢ Use cv2.imshow() from Matplotlib for visualization .


Jupyter Untitled2 Last Checkpoint: 53 seconds ago
File Edit View Run Kernel Settings Help
Code
JupyterLab
Smiley

```
[3]: import numpy as np
     import cv2
     blank_img = np.zeros((500, 500, 3), dtype=np.uint8)  # Black canvas
     blank_img[:] = (255, 255, 255)  # White background

[*]: img = np.zeros((500, 500, 3), dtype=np.uint8)
     cv2.circle(img, (250, 250), 100, (0, 255, 0), -1)  # Face
     cv2.circle(img, (220, 220), 20, (255, 0, 0), -1)  # Left eye
     cv2.circle(img, (280, 220), 20, (255, 0, 0), -1)  # Right eye
     cv2.ellipse(img, (250, 280), (50, 25), 0, 0, 180, (0, 0, 255), 3)  # Smile
     cv2.imshow('Smiley', img)
     cv2.waitKey(0)

[ ]:
```

❑ ***cv2.ellipse(image, center, axes, angle, start Angle, end Angle, color, thickness)***

5. Applications of Graphics with OpenCV :

1- Custom GUI Overlays for Videos:

Add annotations, bounding boxes, or labels to live video Feeds (e.g., during object detection).

2- Data Visualization:

Plot data directly onto images (e.g., charts, points, or heatmaps).

3- Interactive Dashboards:

Create real-time interactive visualizations or feedback systems (e.g., touch-based apps).

5. Following Applications of Graphics :

4- Image Annotation:

Draw shapes and text to annotate or highlight specific parts of an image.

5- Game Development:

Create 2D game-like graphics, such as sprites or interactive environments.

6- Augmented Reality (AR):

Draw virtual objects or patterns on real-world images to create AR effects.

6. Setting Camera Parameters :

1- cv2.VideoCapture(arg).

arg : 0 => open camera of computer OR ‘vid_name’.

2- cv2.VideoWriter_fourcc(*’mp4v’).

isOpened().

3- get(cv2.CAP_PROP_FRAME_WIDTH) OR get(3).

get(cv2.CAP_PROP_FRAME_HEIGHT) OR get(4).

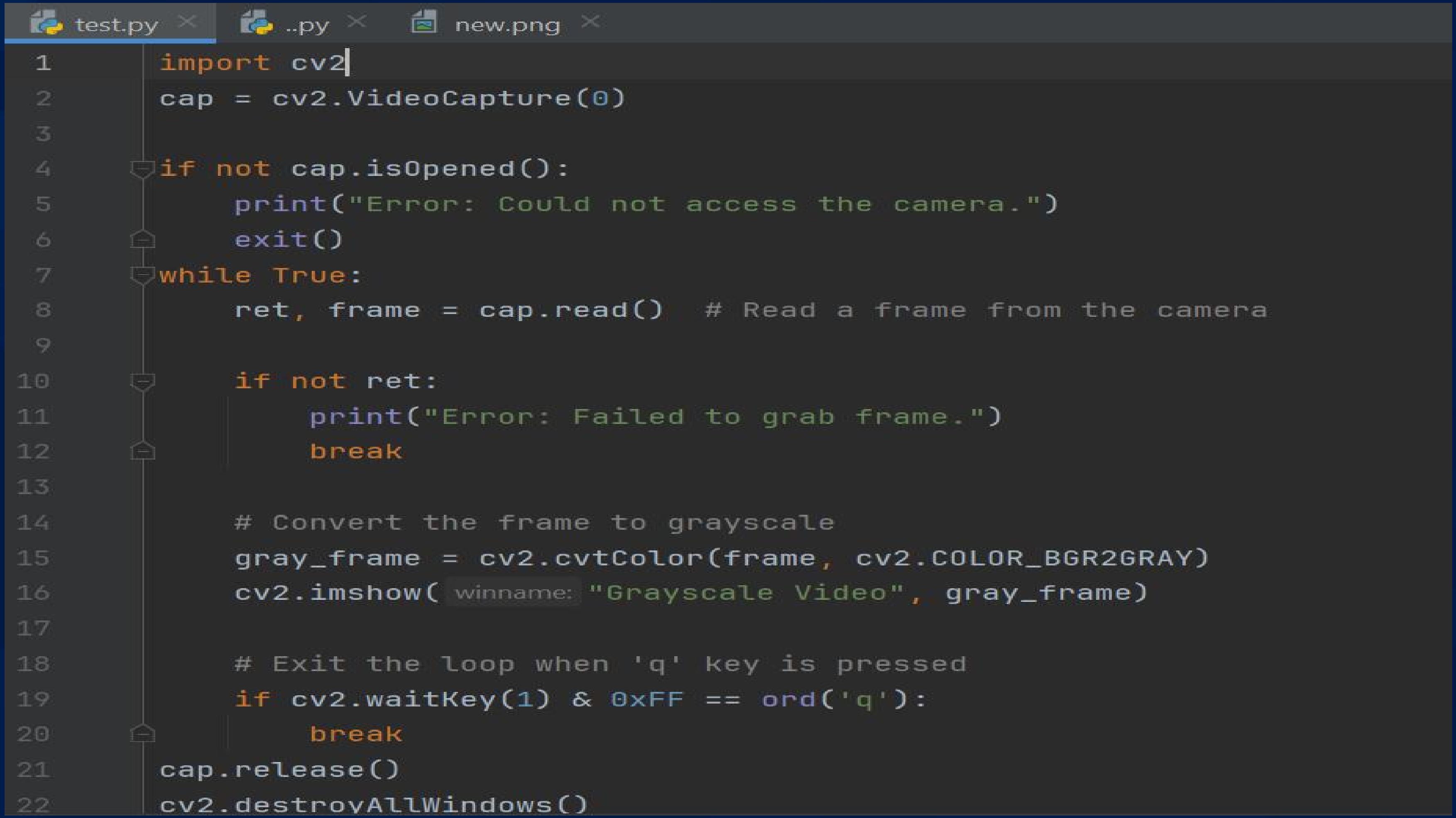
4- Cap.set(3,new_width).

Cap.set(4,new_height).

5- read().

release().

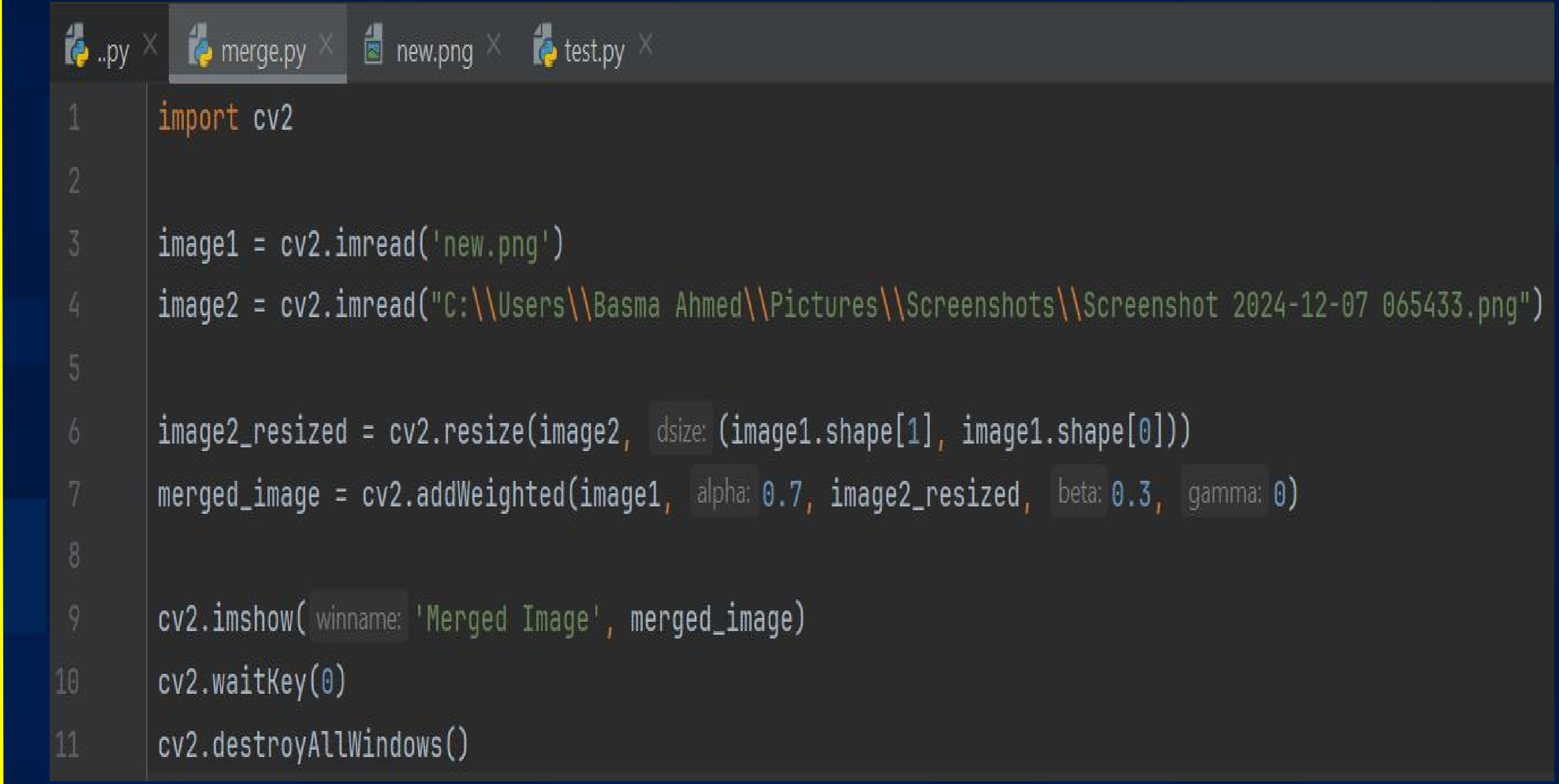
Example on Opening the camera:



```python
import cv2
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Error: Could not access the camera.")
    exit()
while True:
    ret, frame = cap.read()  # Read a frame from the camera

    if not ret:
        print("Error: Failed to grab frame.")
        break

    # Convert the frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cv2.imshow( winname: "Grayscale Video", gray_frame)

    # Exit the loop when 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

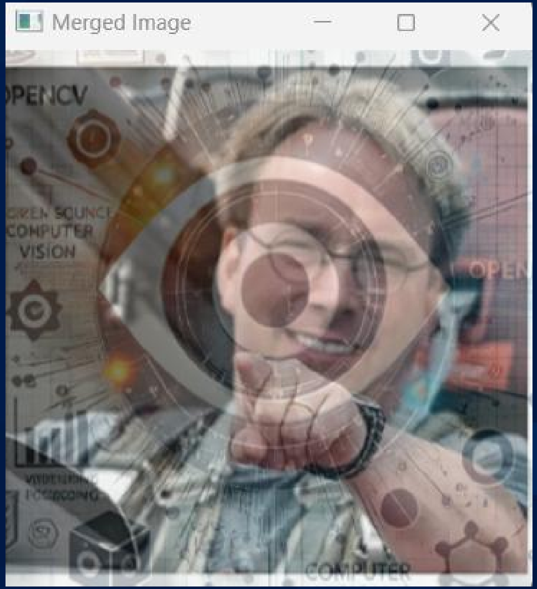
Processes on images

Merging two photos :



```
import cv2

image1 = cv2.imread('new.png')
image2 = cv2.imread("C:\\Users\\Basma Ahmed\\Pictures\\Screenshots\\Screenshot 2024-12-07 065433.png")

image2_resized = cv2.resize(image2, dsize: (image1.shape[1], image1.shape[0]))
merged_image = cv2.addWeighted(image1, alpha: 0.7, image2_resized, beta: 0.3, gamma: 0)

cv2.imshow( winname: 'Merged Image', merged_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



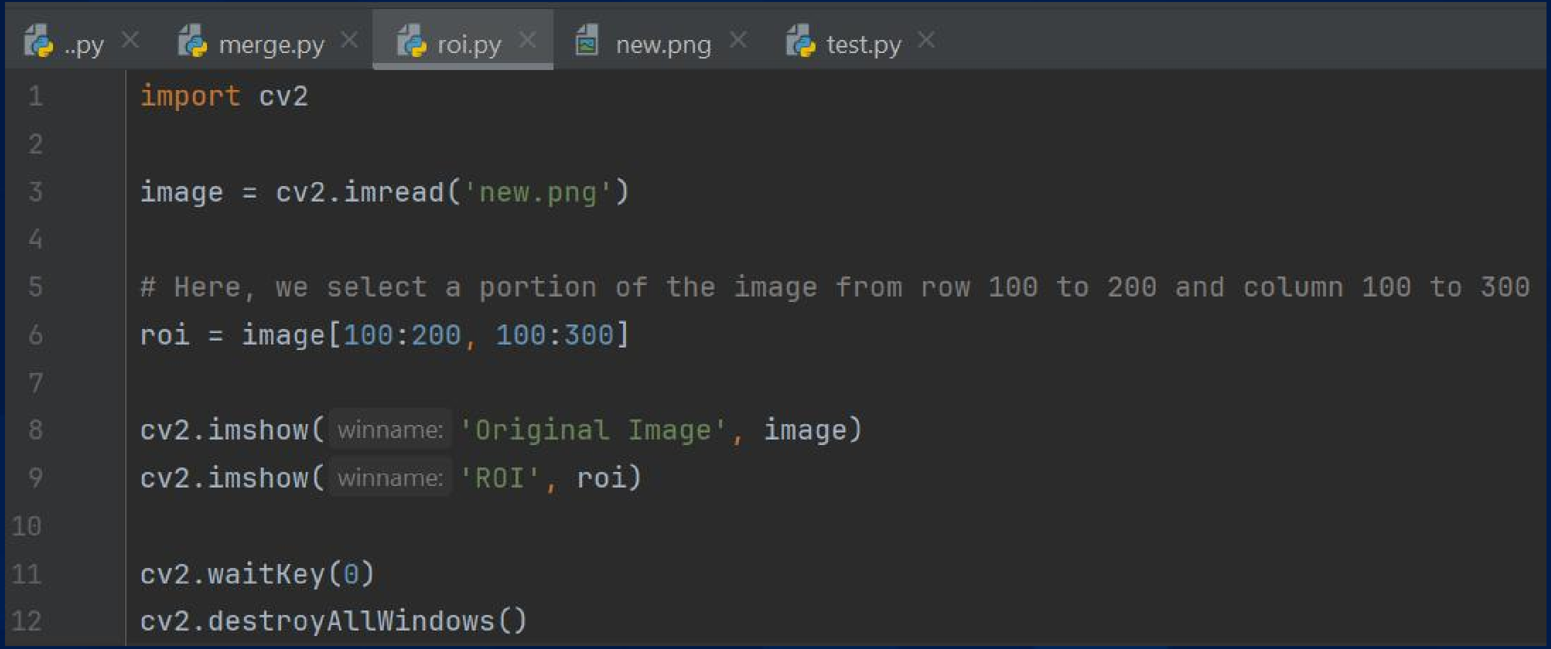
Processes on images

Roi (Region of Interest) :

It allows you to focus on a part of the image (for example, a face or an object) instead of the entire image.

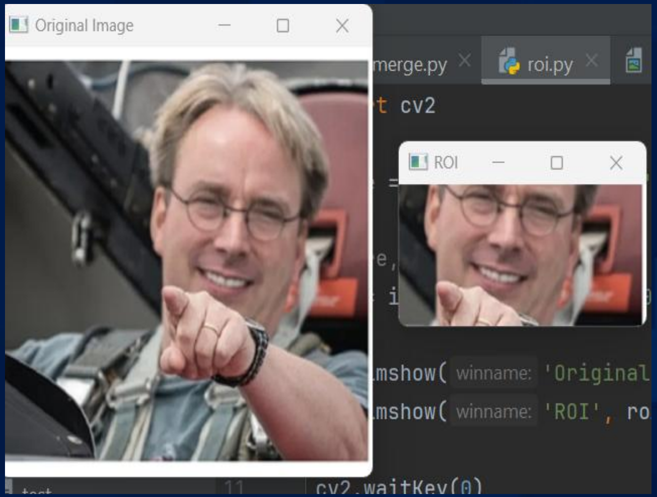
It is usually defined as

[start_row:end_row, start_col:end_col] for an image.



```
import cv2

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

# Here, we select a portion of the image from row 100 to 200 and column 100 to 300
roi = image[100:200, 100:300]

cv2.imshow( winname: 'Original Image', image)
cv2.imshow( winname: 'ROI', roi)

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

The Output :



Important definitions

Key Parameters for Face Detection

scaleFactor:

Purpose: Controls how the image is resized during detection to handle faces of different sizes.

Example Values:

- **1.1:** Resizes the image by 10% at each scale. Higher accuracy, but slower.
- **1.5:** Resizes by 50% at each scale. Faster but less precise.

Effect: Lower values detect smaller faces but take more time.

minNeighbors:

Purpose: Specifies how many overlapping rectangles are required to confirm a face.

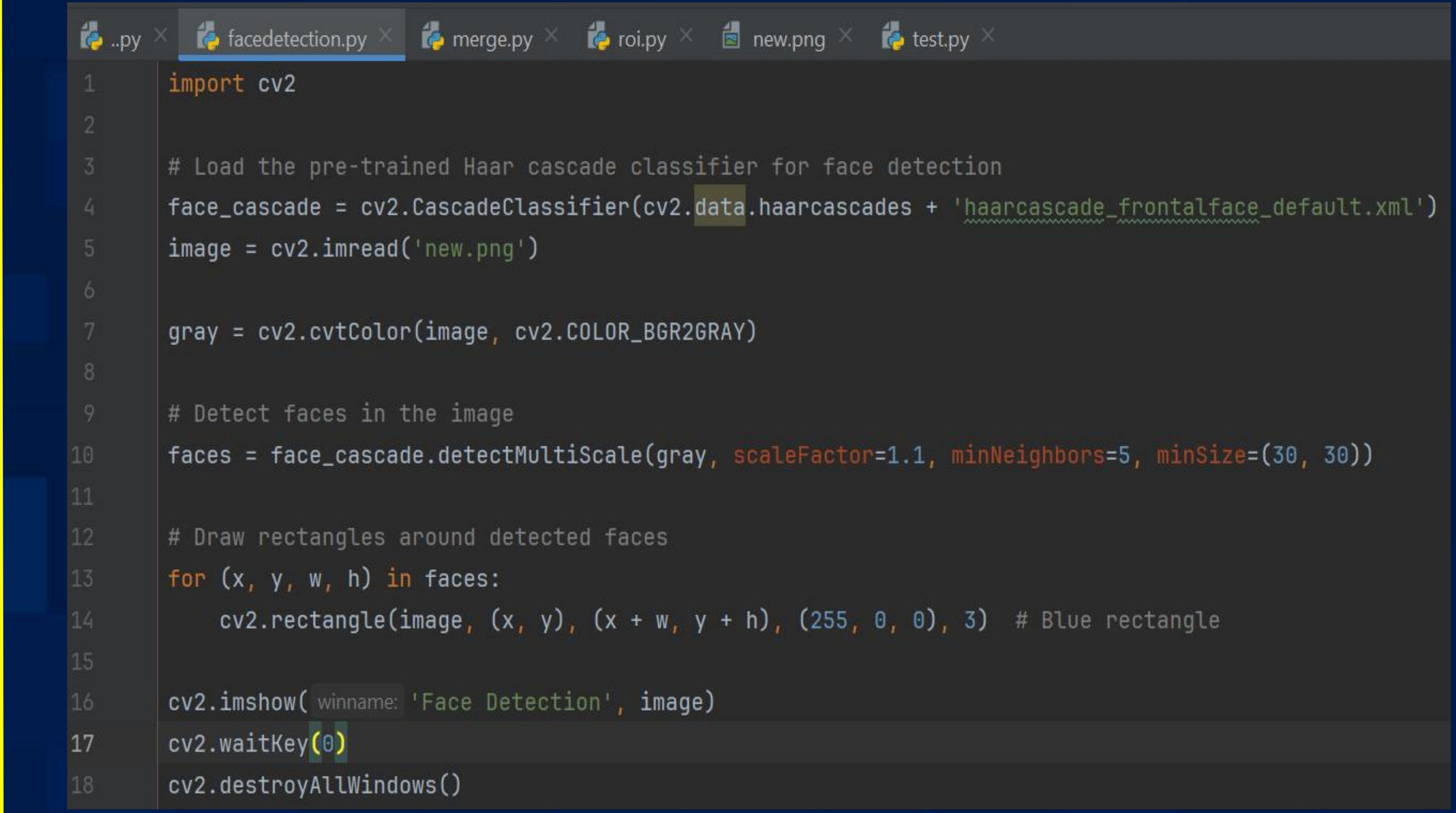
Example Values:

- **3:** Less strict, detects more faces but may include false positives.
- **5:** Stricter, fewer false positives but may miss some faces.

Effect: Higher values reduce false positives but might miss some faces.

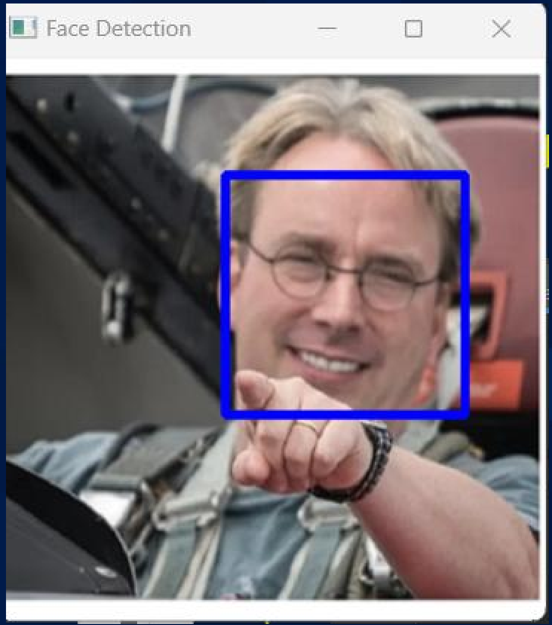
Face detection

Using Haar Cascade :

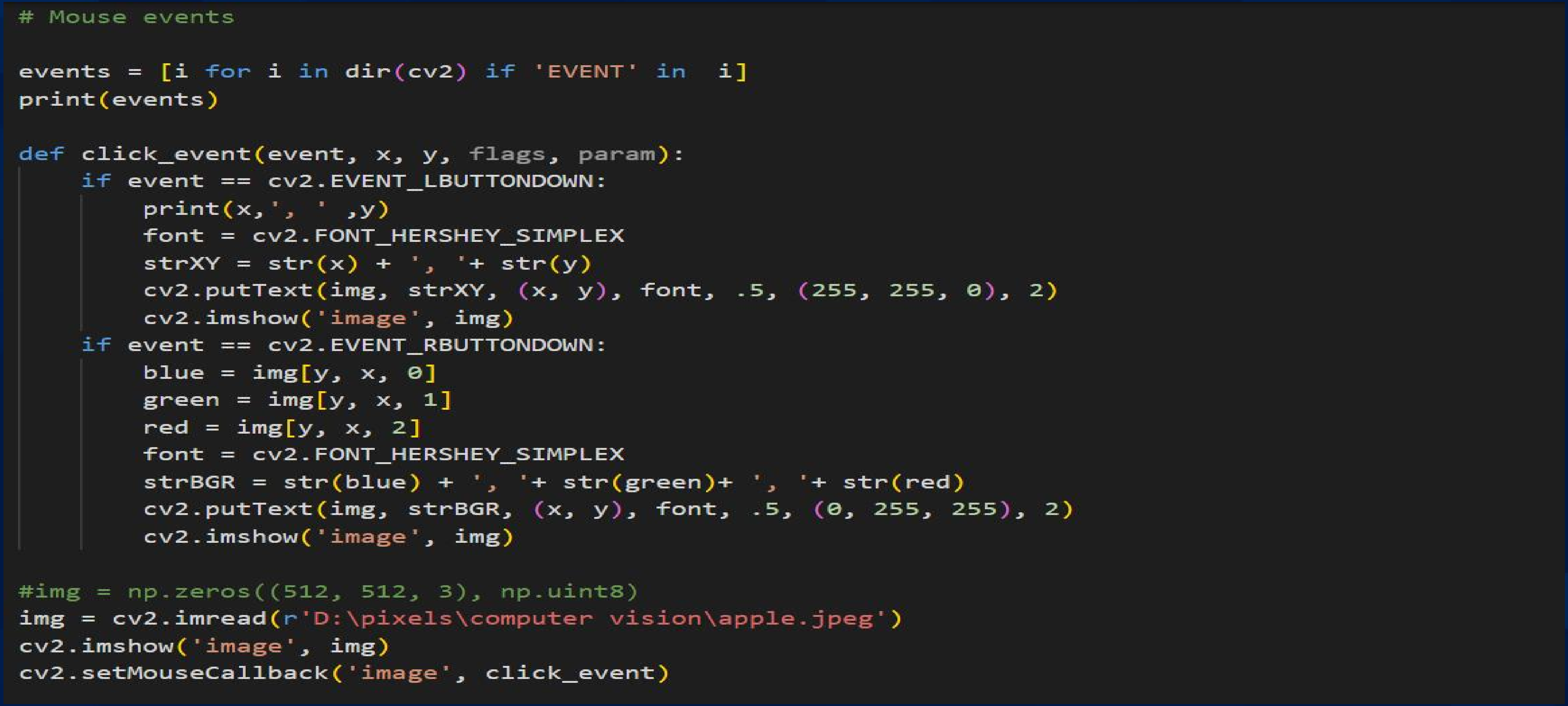


```python
import cv2

# Load the pre-trained Haar cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
image = cv2.imread('new.png')

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Draw rectangles around detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 3)  # Blue rectangle

cv2.imshow( winname: 'Face Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The Output :



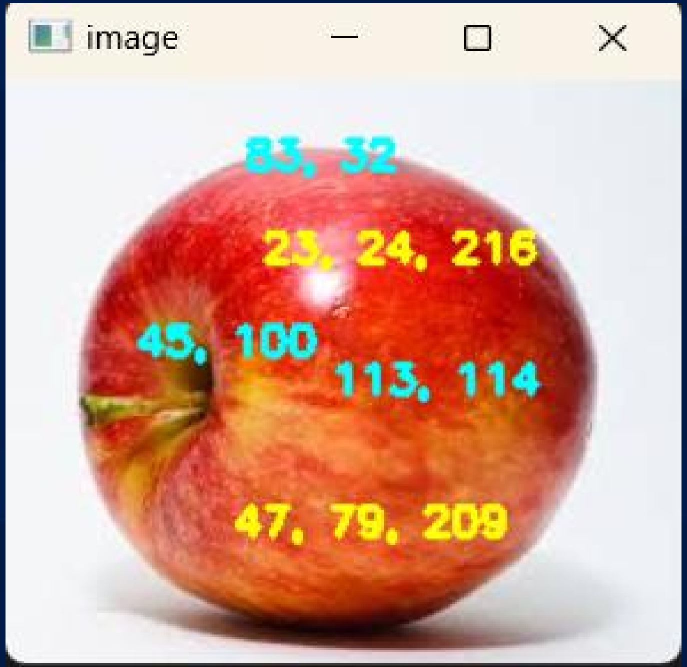
Mouse events



```
# Mouse events

events = [i for i in dir(cv2) if 'EVENT' in  i]
print(events)

def click_event(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        print(x,', ' ,y)
        font = cv2.FONT_HERSHEY_SIMPLEX
        strXY = str(x) + ', '+ str(y)
        cv2.putText(img, strXY, (x, y), font, .5, (255, 255, 0), 2)
        cv2.imshow('image', img)
    if event == cv2.EVENT_RBUTTONDOWN:
        blue = img[y, x, 0]
        green = img[y, x, 1]
        red = img[y, x, 2]
        font = cv2.FONT_HERSHEY_SIMPLEX
        strBGR = str(blue) + ', '+ str(green)+ ', '+ str(red)
        cv2.putText(img, strBGR, (x, y), font, .5, (0, 255, 255), 2)
        cv2.imshow('image', img)

#img = np.zeros((512, 512, 3), np.uint8)
img = cv2.imread(r'D:\pixels\computer vision\apple.jpeg')
cv2.imshow('image', img)
cv2.setMouseCallback('image', click_event)
```

Mouse events

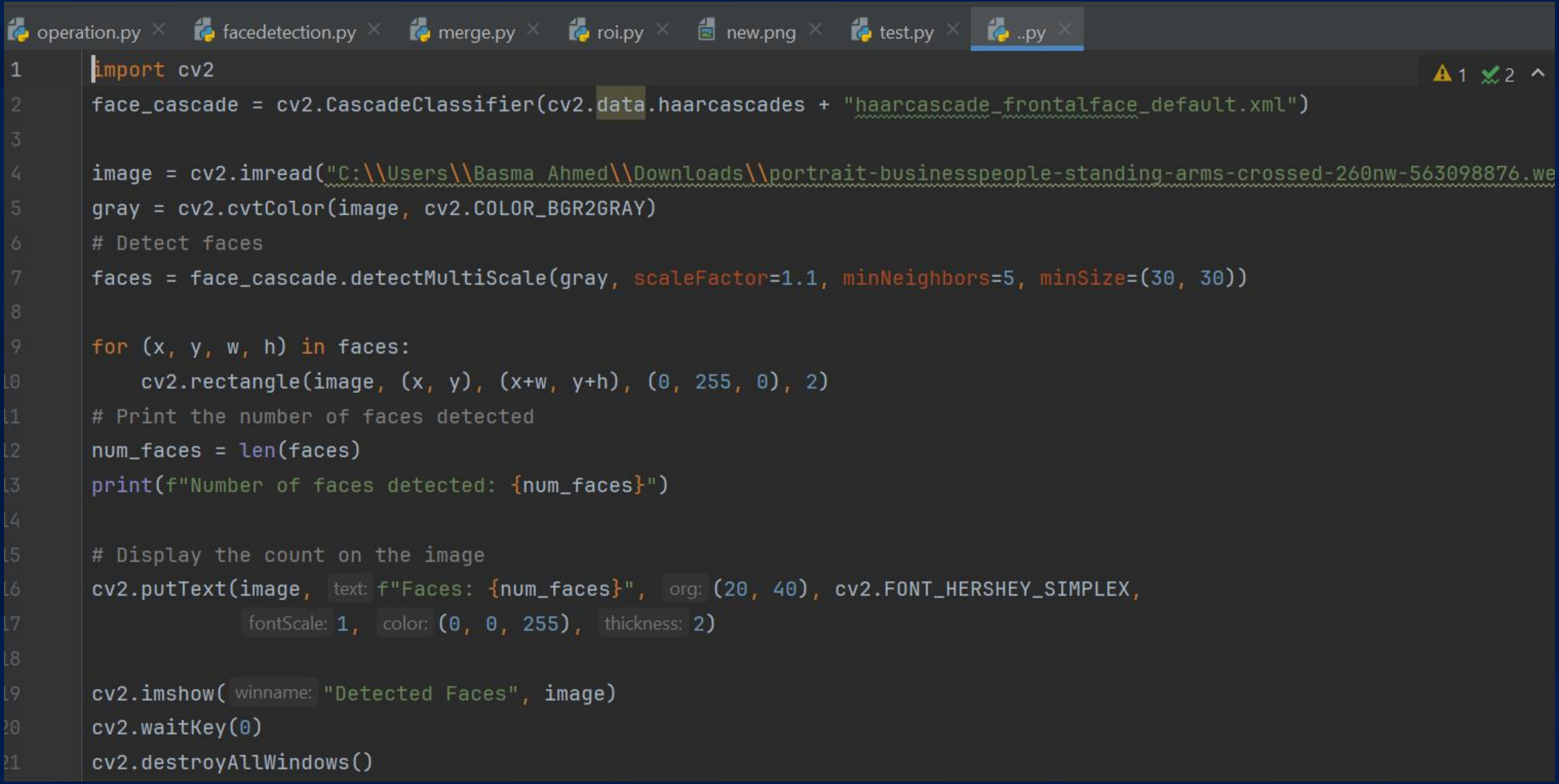
The output



Task 3

detect faces in an image,
draw rectangles around them,
and display the total number of faces
detected on the image.

Solution



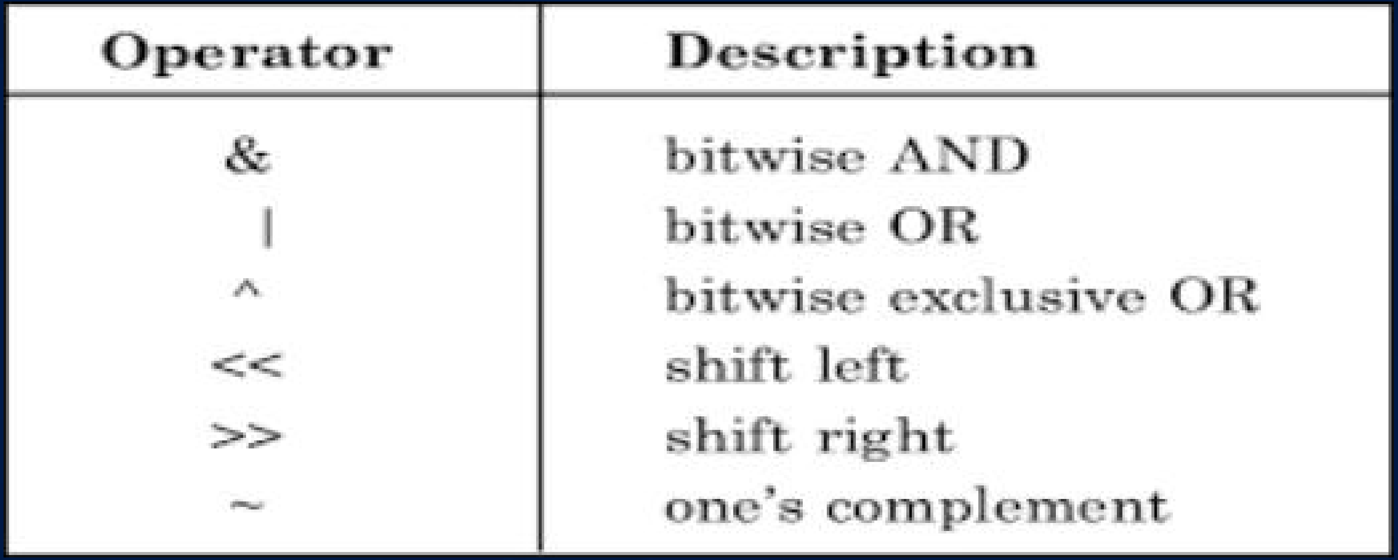
```
import cv2
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

image = cv2.imread("C:\\Users\\Basma Ahmed\\Downloads\\portrait-businesspeople-standing-arms-crossed-260nw-563098876.we
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Detect faces
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
# Print the number of faces detected
num_faces = len(faces)
print(f"Number of faces detected: {num_faces}")

# Display the count on the image
cv2.putText(image,  text: f"Faces: {num_faces}",  org: (20, 40), cv2.FONT_HERSHEY_SIMPLEX,
             fontScale: 1,  color: (0, 0, 255),  thickness: 2)

cv2.imshow( winname: "Detected Faces", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

What are Bitwise Operations?

7.0 Intro for Bitwise Operations :

- Bitwise operations are operations that directly manipulate individual bits of binary numbers.

- These operations work at the bit level, meaning they operate on the binary digits (0s and 1s) that make up numbers.



Operator	Description
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	one's complement

Pixels
EVERY PIXEL IS KNOWLEDGE

How to convert integers to binary?

Converting integers to binary numbers is a fundamental concept in computer science.

Here's how it's typically done:

Manual Conversion (Division Method):

1. Divide the integer by 2.
2. Record the remainder (0 or 1) .
3. Divide the quotient by 2 and repeat until the quotient is 0.
4. Write the remainders in reverse order to get the binary representation.

Example on converting to binary :

Convert 13 to binary ?

Solution:

1) 13 ÷ 2 = 6 remainder 1 .

2) 6 ÷ 2 = 3 remainder 0 .

3) 3 ÷ 2 = 1 remainder 1 .

4) 1 ÷ 2 = 0 remainder 1 .

★ **Final result:**

Binary of 13 : 1101.

☆ **As a reversed order start from bottom to the top to get your solution !.**

Task 4 (for discussion) :

- **Mission 1** : What happens when you apply the NOT operation to a number?
- **Mission 2** : How does the Left Shift operation affect the number in terms of binary representation?
- **Mission 3** : Why does the AND operation return fewer 1s in the binary result than OR?

Key Bitwise Operators:

a. AND (&):

Compares corresponding bits of two numbers and returns 1 if both bits are 1, otherwise returns 0.

Example: 5 & 3 (Binary: 0101 & 0011 = 0001) → Result : 1.

b. OR (|):

Compares corresponding bits of two numbers and returns 1 if at least one of the bits is 1, otherwise returns 0.

Example: 5 | 3 (Binary: 0101 | 0011 = 0111) → Result : 7.

C. XOR (^):

Compares corresponding bits of two numbers and returns 1 if the bits are different, otherwise returns 0.

Example: 5 ^ 3 (Binary: 0101 ^ 0011 = 0110) → Result: 6.

D. NOT (~):

Inverts all the bits of a number, changing 1 to 0 and 0 to 1. This is also known as the "bitwise complement".

Example: ~5 (Binary: ~0101 = 1010) → Result: -6 (in two's complement representation).

Small Note:
Two's complement is a binary encoding scheme used to represent negative and positive numbers.

e. Shift Left (<<):

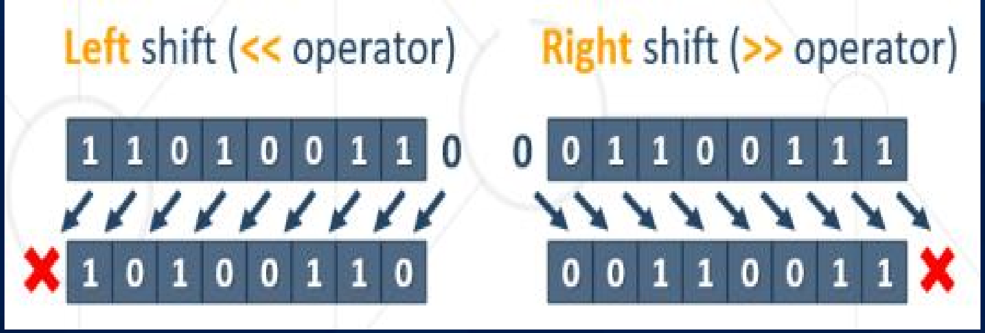
Shifts all bits of a number to the left by a specified number of positions, adding zeros to the right .

Example : 5 << 1 (Binary: 0101 << 1 = 1010) → Result: 10.

f. Shift Right (>>):

Shifts all bits of a number to the right by a specified number of positions, discarding bits on the right .

Example: 5 >> 1 (Binary: 0101 >> 1 = 0010) → Result : 2.



Example :

If we have two numbers 5 and 3, we can apply bitwise operations :

Solution:

1. **AND** : 5 & 3 = 1 (Only positions where both have 1) .

2. **OR** : 5 | 3 = 7 (At least one 1 in each position) .

3. **XOR** : 5 ^ 3 = 6 (Positions where bits differ) .

4. **NOT: ~5 = -6** (Flips all the bits of the number, changes 1s to 0s and 0s to 1s) .

5. **Left Shift :** 5 << 1 = 10 (Shifts all bits to the left, multiplying the number by 2) .

6. **Right Shift :** 5 >> 1 = 2 (Shifts all bits to the right, dividing the number by 2) .

Pixels
EVERY PIXEL IS KNOWLEDGE

Task 5:

Perform and understand the results of bitwise operations
(AND, OR ., XOR, NOT, left shift, right shift)
on two integers.

فيما أفناه

Pixels
EVERY PIXEL IS KNOWLEDGE

Applications of Bitwise Operations:

- **Performance Optimization** : Often used in situations where speed is critical, such as embedded systems or gaming .

- **Masking** : Used to isolate specific bits in a number (e.g., turning off specific bits or flags in settings) .

- **Cryptography** : Bitwise operations are fundamental to encryption algorithms .

Following The Applications :

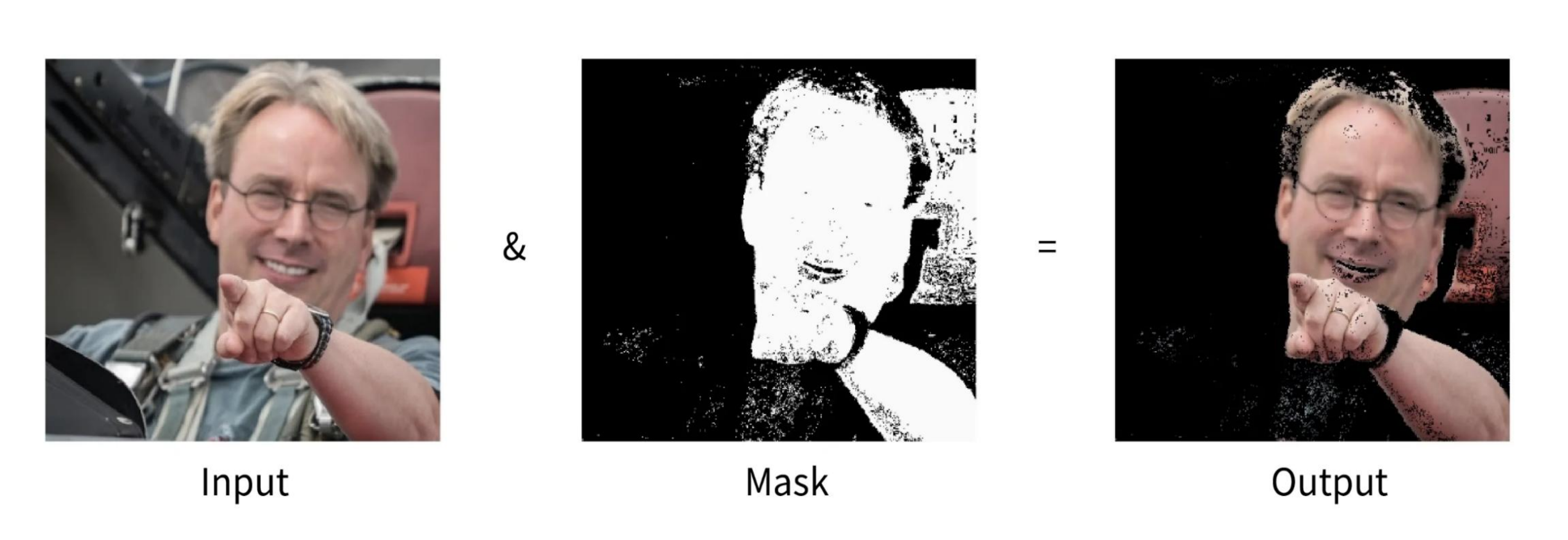
- **Networking** : IP addresses and subnetting often use bitwise operations to calculate networks and subnets .

- **Manipulating Image Pixels** : Can directly manipulate pixel values for effects like image inversion, filtering, etc .

- **Combining Images** : Helps in overlaying or blending images by using bitwise AND, OR, and XOR to combine pixel information .

7. Bitwise Operations on Images :

Bitwise operations manipulate image pixels at the binary level .
Useful for creating masks and combining images .



7.1. Bitwise Operations on Images :

Common Operations :

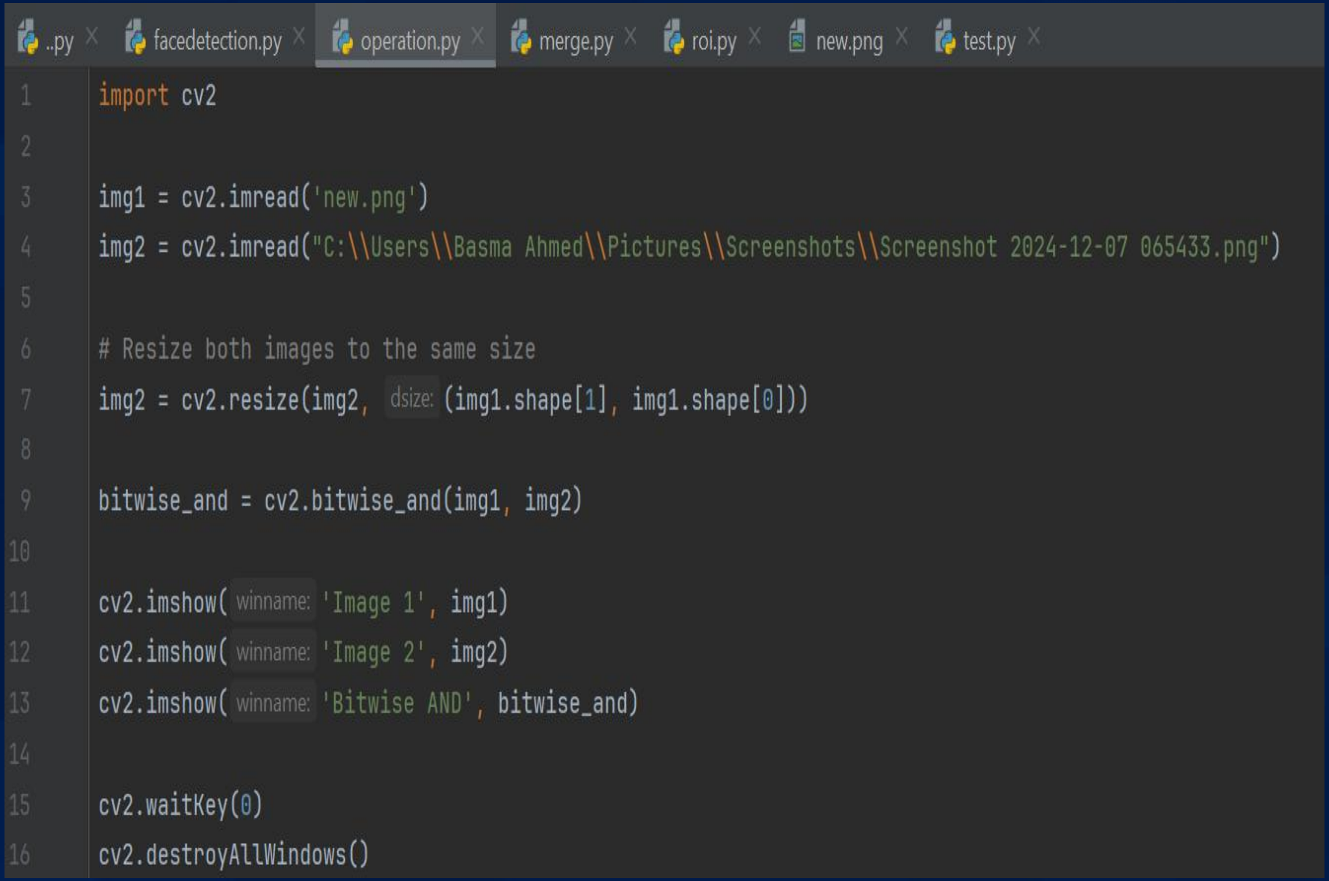
I. cv2.bitwise_and(img1, img2) : Performs AND operation .

I. cv2.bitwise_or(img1, img2) : Performs OR operation .

II. cv2.bitwise_xor(img1, img2) : Performs XOR operation .

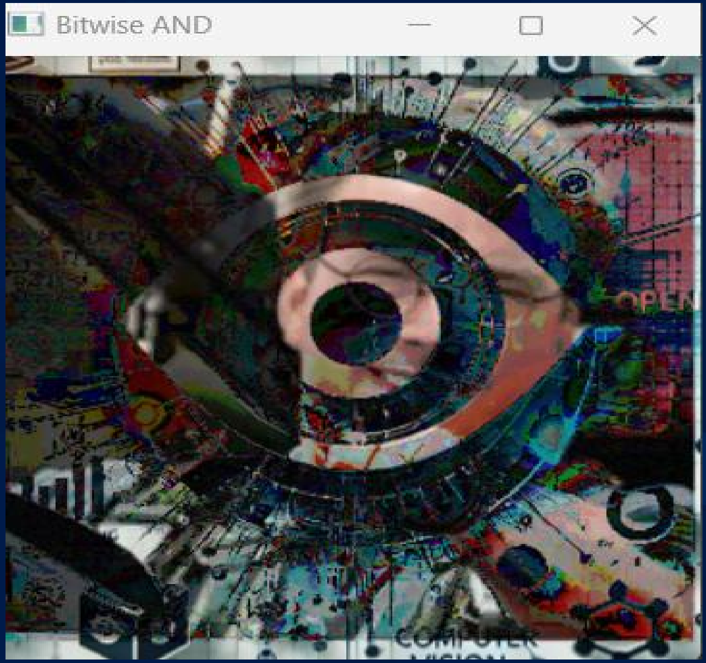
III. cv2.bitwise_not(img) : Performs NOT operation .

Example on Common Operations :



```
import cv2

img1 = cv2.imread('new.png')
img2 = cv2.imread("C:\\Users\\Basma Ahmed\\Pictures\\Screenshots\\Screenshot 2024-12-07 065433.png")

# Resize both images to the same size
img2 = cv2.resize(img2, dsize: (img1.shape[1], img1.shape[0]))

bitwise_and = cv2.bitwise_and(img1, img2)

cv2.imshow( winname: 'Image 1', img1)
cv2.imshow( winname: 'Image 2', img2)
cv2.imshow( winname: 'Bitwise AND', bitwise_and)

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

The Output :



Task 6:

Combine two images using cv2.bitwise_and, cv2.bitwise_or, cv2.bitwise_xor, and cv2.bitwise_not, and visualize the differences.

Pixels
EVERY PIXEL IS KNOWLEDGE

THANK YOU .

فيما أفناه