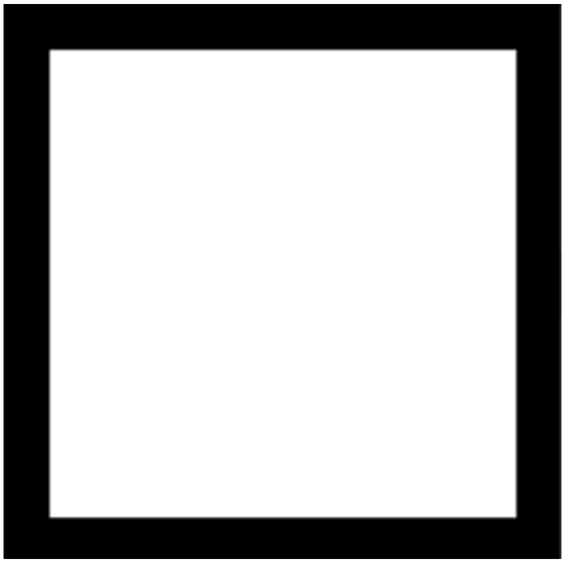
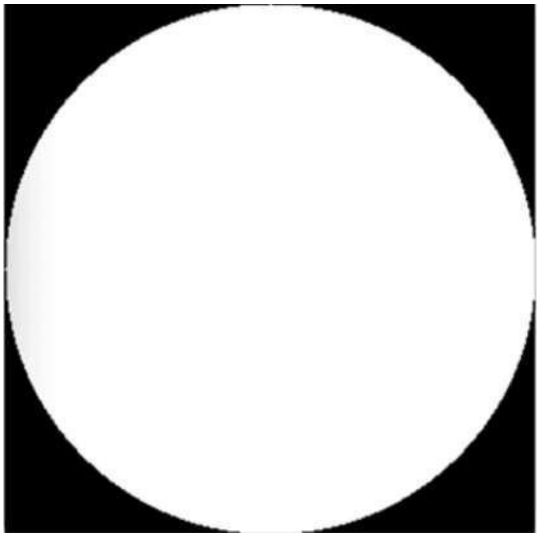
Practice Tasks : **Evaluation in next Lecture**

Task#1: Execute lab1.py and describe your observations.

It shows images in six color form.It is mainly based on parameters(weights) on which we perform arithmetic operations and that changes the color of images and then it plots them in the graph.

Task#2: Use the given below images as your input images and perform all arithmetic and logical operations. Show your code and also state your observations for each.

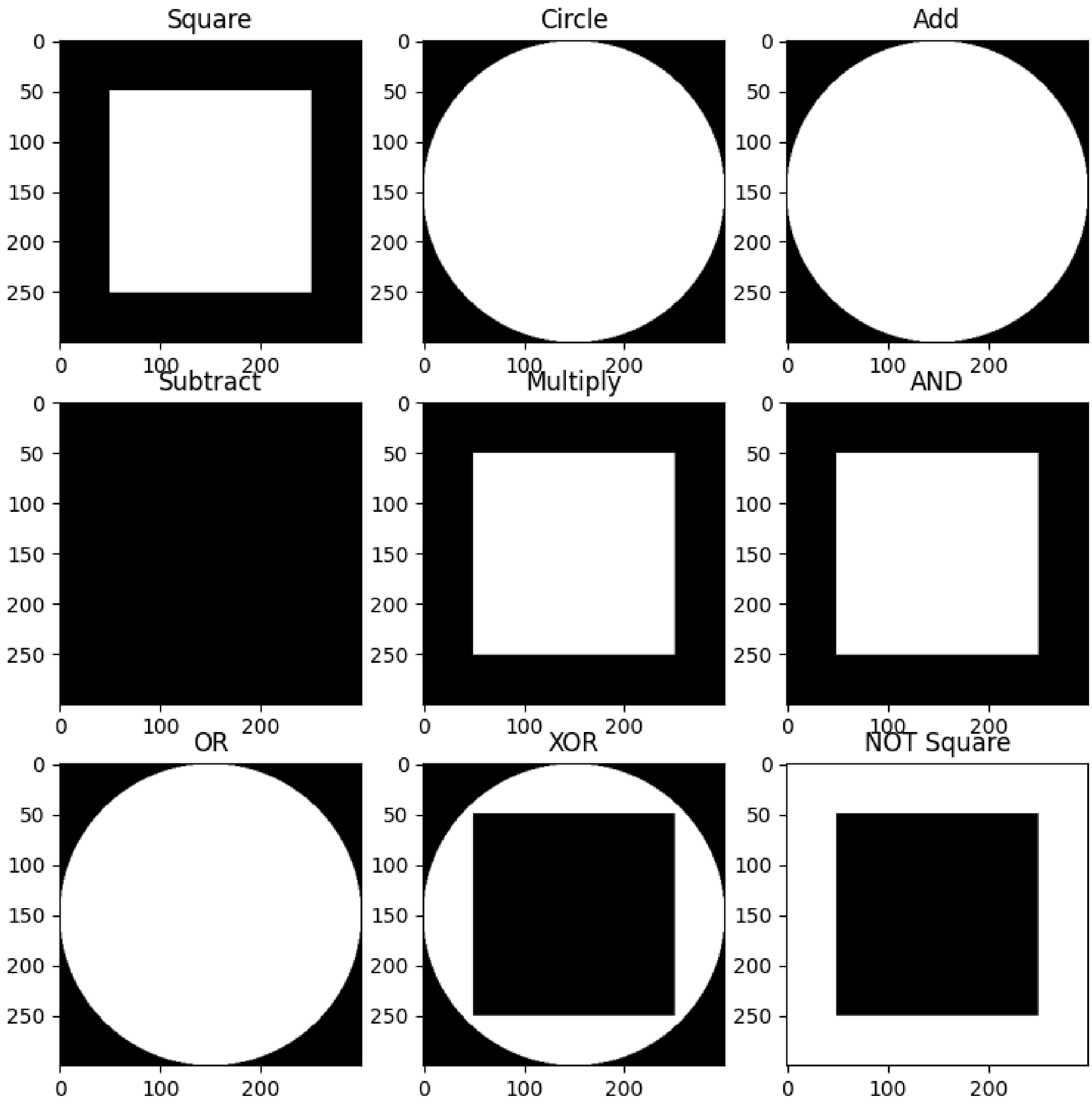




Code:

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

# Create a black square image
square = np.zeros((300, 300), dtype="uint8")
cv2.rectangle(square, (50, 50), (250, 250), 255, -1)  # Draw a white square

# Create a black circle image
circle = np.zeros((300, 300), dtype="uint8")
cv2.circle(circle, (150, 150), 150, 255, -1)  # Draw a white circle

# Perform arithmetic operations
add_image = cv2.add(square, circle)
subtract_image = cv2.subtract(square, circle)
multiply_image = cv2.multiply(square, circle)

# Perform logical operations
bitwise_and = cv2.bitwise_and(square, circle)
bitwise_or = cv2.bitwise_or(square, circle)
bitwise_xor = cv2.bitwise_xor(square, circle)
bitwise_not_square = cv2.bitwise_not(square)

# Plot results
fig = plt.figure(figsize=(9, 9))

# Original images
fig.add_subplot(3, 3, 1)
plt.title('Square')
plt.imshow(square, cmap='gray')

fig.add_subplot(3, 3, 2)
plt.title('Circle')
plt.imshow(circle, cmap='gray')

# Arithmetic operations
fig.add_subplot(3, 3, 3)
plt.title('Add')
plt.imshow(add_image, cmap='gray')

fig.add_subplot(3, 3, 4)
plt.title('Subtract')
plt.imshow(subtract_image, cmap='gray')
```

```
fig.add_subplot(3, 3, 5)
plt.title('Multiply')
plt.imshow(multiply_image, cmap='gray')

# Logical operations
fig.add_subplot(3, 3, 6)
plt.title('AND')
plt.imshow(bitwise_and, cmap='gray')

fig.add_subplot(3, 3, 7)
plt.title('OR')
plt.imshow(bitwise_or, cmap='gray')

fig.add_subplot(3, 3, 8)
plt.title('XOR')
plt.imshow(bitwise_xor, cmap='gray')

fig.add_subplot(3, 3, 9)
plt.title('NOT Square')
plt.imshow(bitwise_not_square, cmap='gray')

# Save the image
plt.savefig('image_operations_results.png')
plt.show()
```



Task#3: Capture your image through webcam and save it. You may use code given in CaptureVideoImage.py.

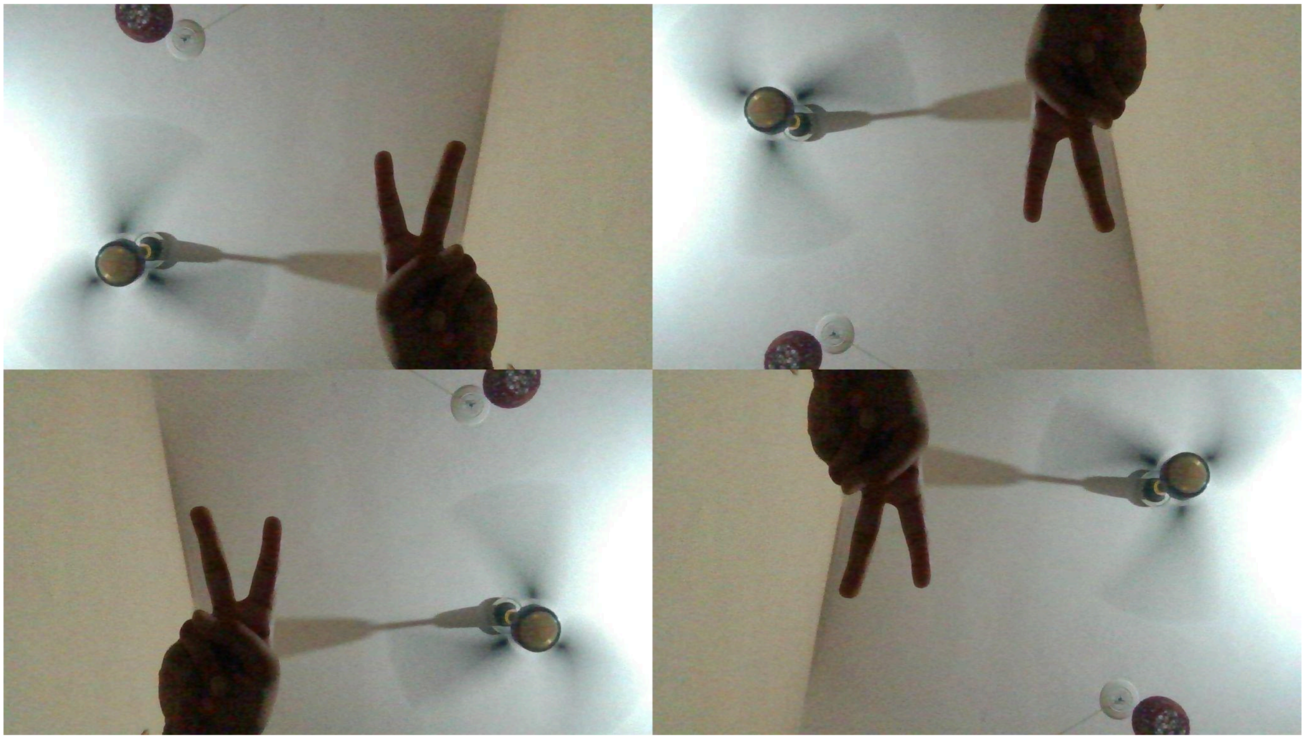
Uploaded pic as demanded. It release a camera and start capturing me.

Task#4: Perform all the types of flipping on the captured image. Show the flipped images.

Code:

```
import cv2
import numpy as np

# Load the image
input_image = cv2.imread("Pic_for_Lab_2.jpg")

# Apply different flips
flipped_vertically = cv2.flip(input_image, 0)  # Flip on the vertical axis
(x-axis)
flipped_horizontally = cv2.flip(input_image, 1)  # Flip on the horizontal
axis (y-axis)
flipped_both_axes = cv2.flip(input_image, -1)  # Flip on both axes

# Arrange flipped images into a grid
upper_half = np.concatenate((input_image, flipped_vertically), axis=1)  #
Original with vertical flip
lower_half = np.concatenate((flipped_horizontally, flipped_both_axes),
axis=1)  # Horizontal and both-axis flip

# Combine upper and lower rows to make a complete grid
final_image = np.concatenate((upper_half, lower_half), axis=0)

# Display and save the final output
```

```
cv2.imshow('Flipped Images Grid', final_image)
cv2.imwrite("combined_flipped_output.jpg", final_image)

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

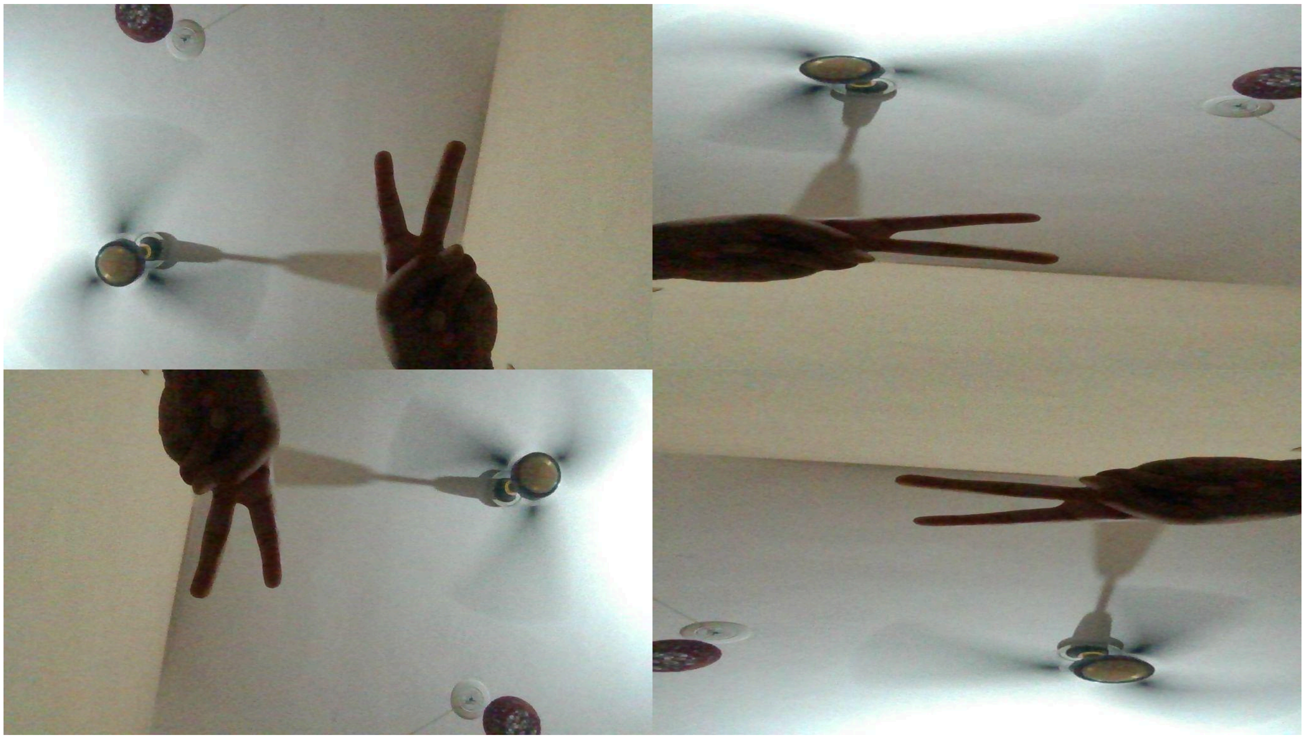


Task#5: Perform all rotations on the captured image. Show the rotated images.

Code:

```
import cv2
import numpy as np

# Load the image
input_image = cv2.imread("Pic_for_Lab_2.jpg")

# Retrieve dimensions of the original image
height, width = input_image.shape[:2]

# Perform rotations on the image
rotated_90_cw = cv2.rotate(input_image,
cv2.ROTATE_90_CLOCKWISE)  # Rotate 90 degrees clockwise
```

```
rotated_180 = cv2.rotate(input_image, cv2.ROTATE_180)  # Rotate
180 degrees
rotated_270_cw = cv2.rotate(input_image,
cv2.ROTATE_90_COUNTERCLOCKWISE)  # Rotate 270 degrees
clockwise

# Resize rotations to original image dimensions for consistency
rotated_90_cw_resized = cv2.resize(rotated_90_cw, (width, height))
rotated_270_cw_resized = cv2.resize(rotated_270_cw, (width, height))

# Arrange rotated images into a grid layout
upper_row = np.concatenate((input_image, rotated_90_cw_resized),
axis=1)  # Original next to 90° rotated
lower_row = np.concatenate((rotated_180, rotated_270_cw_resized),
axis=1)  # 180° next to 270° rotated

# Stack the rows vertically to create a single display grid
final_output = np.concatenate((upper_row, lower_row), axis=0)

# Display and save the final arrangement of rotations
cv2.imshow('Rotated Image Grid', final_output)
cv2.imwrite("final_rotated_grid.jpg", final_output)

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



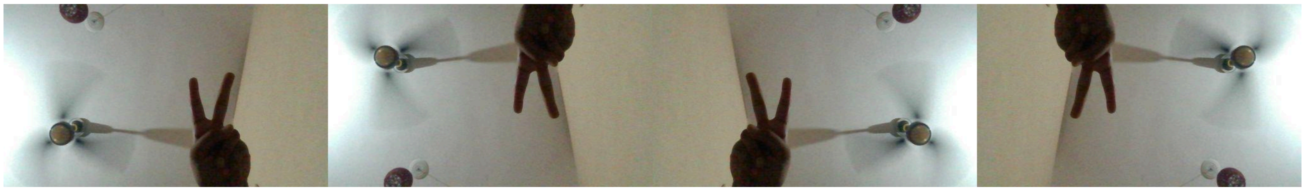
Task#6:

Merge Code:

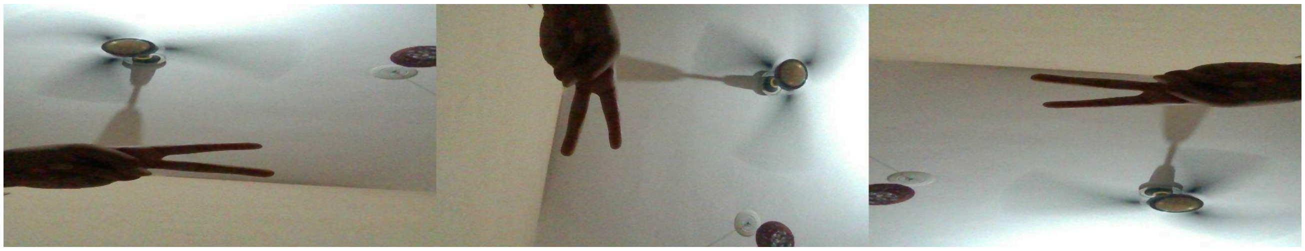
```
import cv2
import numpy as np

# Load the image file
source_image = cv2.imread("Pic_for_Lab_2.jpg")

# Capture original dimensions
height, width = source_image.shape[:2]

# Create flipped versions of the image
flipped_vertically = cv2.flip(source_image, 0)  # Flip along x-axis
flipped_horizontally = cv2.flip(source_image, 1)  # Flip along y-axis
flipped_both_axes = cv2.flip(source_image, -1)  # Flip along both axes

# Ensure flipped images match the original dimensions
flipped_vertically_resized = cv2.resize(flipped_vertically, (width, height))
flipped_horizontally_resized = cv2.resize(flipped_horizontally, (width, height))
flipped_both_axes_resized = cv2.resize(flipped_both_axes, (width, height))
```

```
# Combine original and flipped images horizontally
flipped_combination = cv2.hconcat([source_image,
flipped_vertically_resized, flipped_horizontally_resized,
flipped_both_axes_resized])

# Rotate the image at different angles
rotated_90 = cv2.rotate(source_image,
cv2.ROTATE_90_CLOCKWISE)  # 90° clockwise
rotated_180 = cv2.rotate(source_image, cv2.ROTATE_180)  # 180°
rotation
rotated_270 = cv2.rotate(source_image,
cv2.ROTATE_90_COUNTERCLOCKWISE)  # 270° clockwise or 90°
counterclockwise

# Resize rotated images to maintain consistent dimensions
rotated_90_resized = cv2.resize(rotated_90, (width, height))
rotated_180_resized = cv2.resize(rotated_180, (width, height))
rotated_270_resized = cv2.resize(rotated_270, (width, height))

# Combine rotated images horizontally
rotated_combination = cv2.hconcat([rotated_90_resized,
rotated_180_resized, rotated_270_resized])

# Display combined images for flipped and rotated results
cv2.imshow('Combined Flipped Images', flipped_combination)
cv2.imshow('Combined Rotated Images', rotated_combination)

# Save the resulting combined images
cv2.imwrite("output_flipped_combined.jpg", flipped_combination)
cv2.imwrite("output_rotated_combined.jpg", rotated_combination)

# Wait for a key press and close all displayed windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

i) Concatenate the original and flipped images into 1 image.



ii) Similarly concatenate all the rotated images into 1 image.



Task#7: Execute read.py and describe your observations.

This code begins by displaying an image of cats in a window, which remains open until a key is pressed. Next, it plays a video of a dog frame by frame, showing each one in a window. The video will keep playing until either the “d” key is pressed or it reaches the end. Once the video loop finishes, both the image and video windows close.

OpenCV help functions and their description is given on the next page.

Flip:

We can flip an image around either the x-axis, y-axis, or

even both. Basic Syntax is :

flipped = cv2.flip(image, value)

Value is 1 for horizontal flipping. Value is 0 for vertical flipping. Value is -1 for both axis.

Rotate:

cv2.rotate() method is used to rotate a 2D array in multiples of 90 degrees.
The function cv::rotate rotates the array in three different ways.

1. Rotate by 90 degrees clockwise:
   cv2.rotate(image to be rotated, cv2.ROTATE_90_CLOCKWISE)
2. Rotate by 180 degrees clockwise: cv2.ROTATE_180
3. Rotate by 270 degrees clockwise :
   cv2.ROTATE_90_COUNTERCLOCKWISE

Concatenation of images:

To concatenate images vertically and horizontally with Python, cv2 library comes with two functions as:

1. **hconcat():** It is used as cv2.hconcat() to concatenate images horizontally. Here h means horizontal. cv2.hconcat() is used to combine images of same height horizontally.
2. **vconcat():** It is used as cv2.vconcat() to concatenate images vertically. Here v means vertical. cv2.vconcat() is used to combine images of same width vertically.

Arithmetic and Logical Operators- Bitwise AND, OR, NOR, XOR

Arithmetic Operations like Addition, Subtraction, and Bitwise Operations(AND, OR, NOT, XOR) can be applied to the input images

1. Addition
2. Subtraction

Bitwise operations are used in image manipulation and used for extracting essential parts in the image. In this article, Bitwise operations used are:

1. Bitwise AND
2. Bitwise OR
3. Bitwise XOR
4. Bitwise NOT

1. Addition

- Syntax: cv2.add(img1, img2)
  But adding the pixels is not an ideal situation. So, we use cv2.addweighted(). Remember, both images should be of equal size and depth.

- Syntax: cv2.addWeighted(img1, wt1, img2, wt2, gammaValue)

Parameters:

- img1: First Input Image array(Single-channel, 8-bit or floating-point)
- wt1: Weight of the first input image elements to be applied to the final image
- img2: Second Input Image array(Single-channel, 8-bit or floating-point)
- wt2: Weight of the second input image elements to be applied to the final image
- gammaValue: Measurement of light

2. Subtraction of Image:

Just like addition, we can subtract the pixel values in two images and merge them with the help of cv2.subtract(). The images should be of equal size and depth.

Syntax: cv2.subtract(image1, image2)

3. **AND:** A bitwise AND is true *if and only if* both pixels are greater than zero.

4. **OR:** A bitwise OR is true *if either* of the two pixels is greater than zero.

5. **XOR:** A bitwise XOR is true *if and only if* one of the two pixels is greater than zero, *but not both.*

6. **NOT:** A bitwise NOT inverts the <on= and <off= pixels in an image.

Syntax:

```
bitwiseAnd =

cv2.bitwise_and(rectangle, circle)

bitwiseOr =

cv2.bitwise_or(rectangle, circle)

bitwiseXor =

cv2.bitwise_xor(rectangle, circle)

bitwiseNot = cv2.bitwise_not(circle)
```