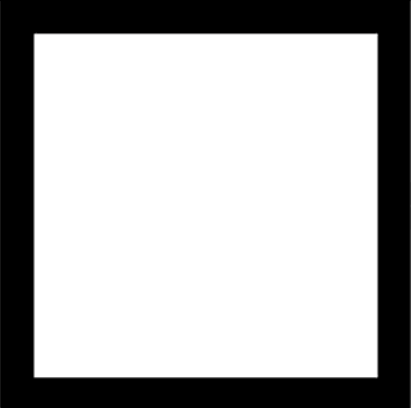
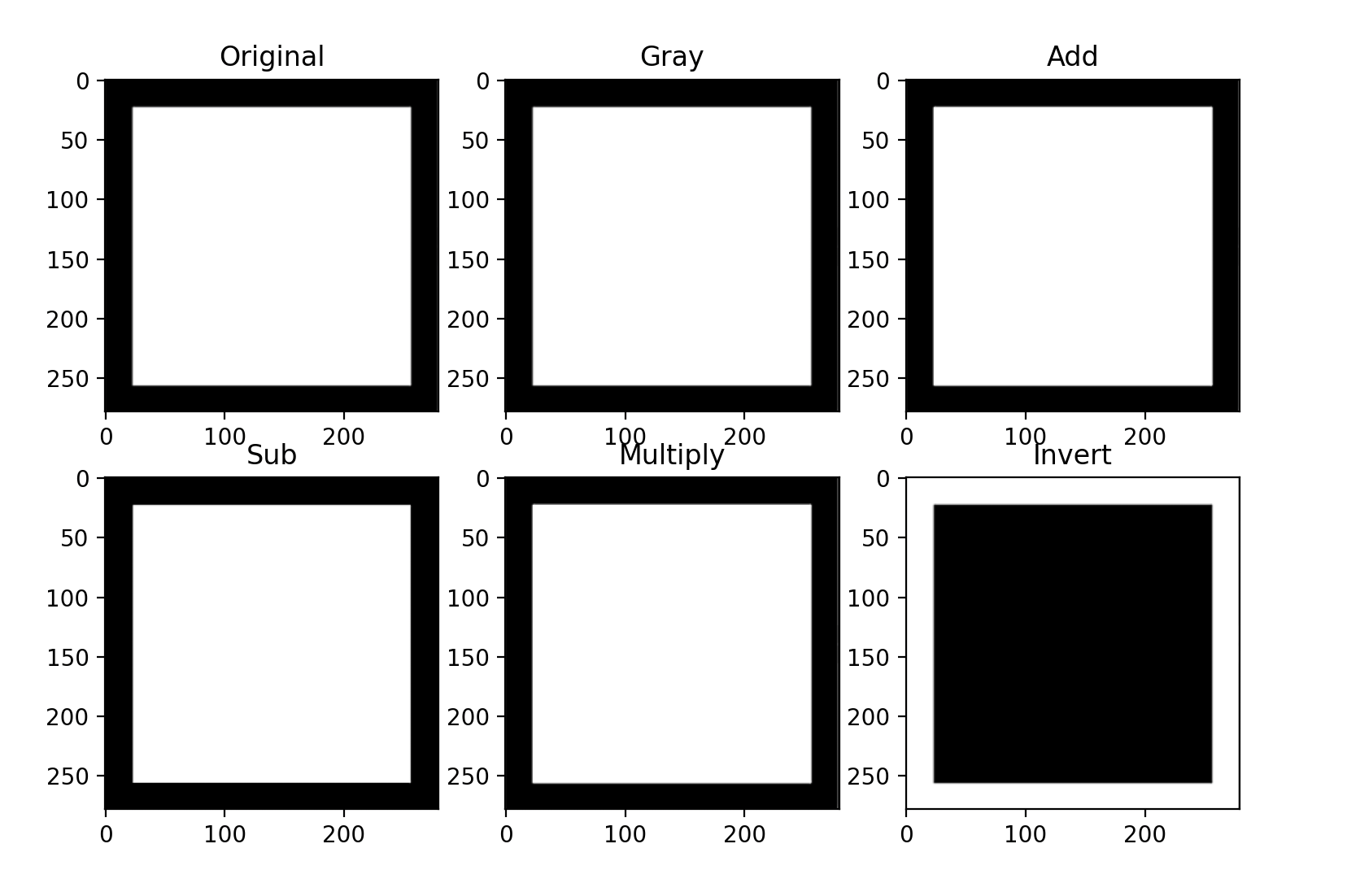
**Practice Tasks : Evaluation in next Lecture**

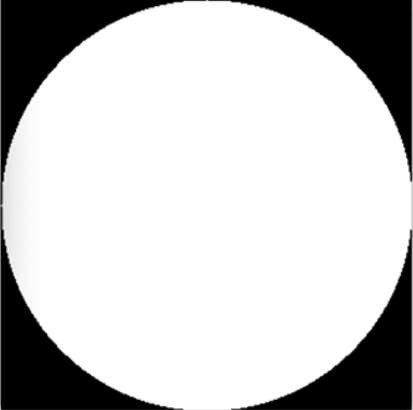
**Task#1: Execute lab1.py and describe your observations.**

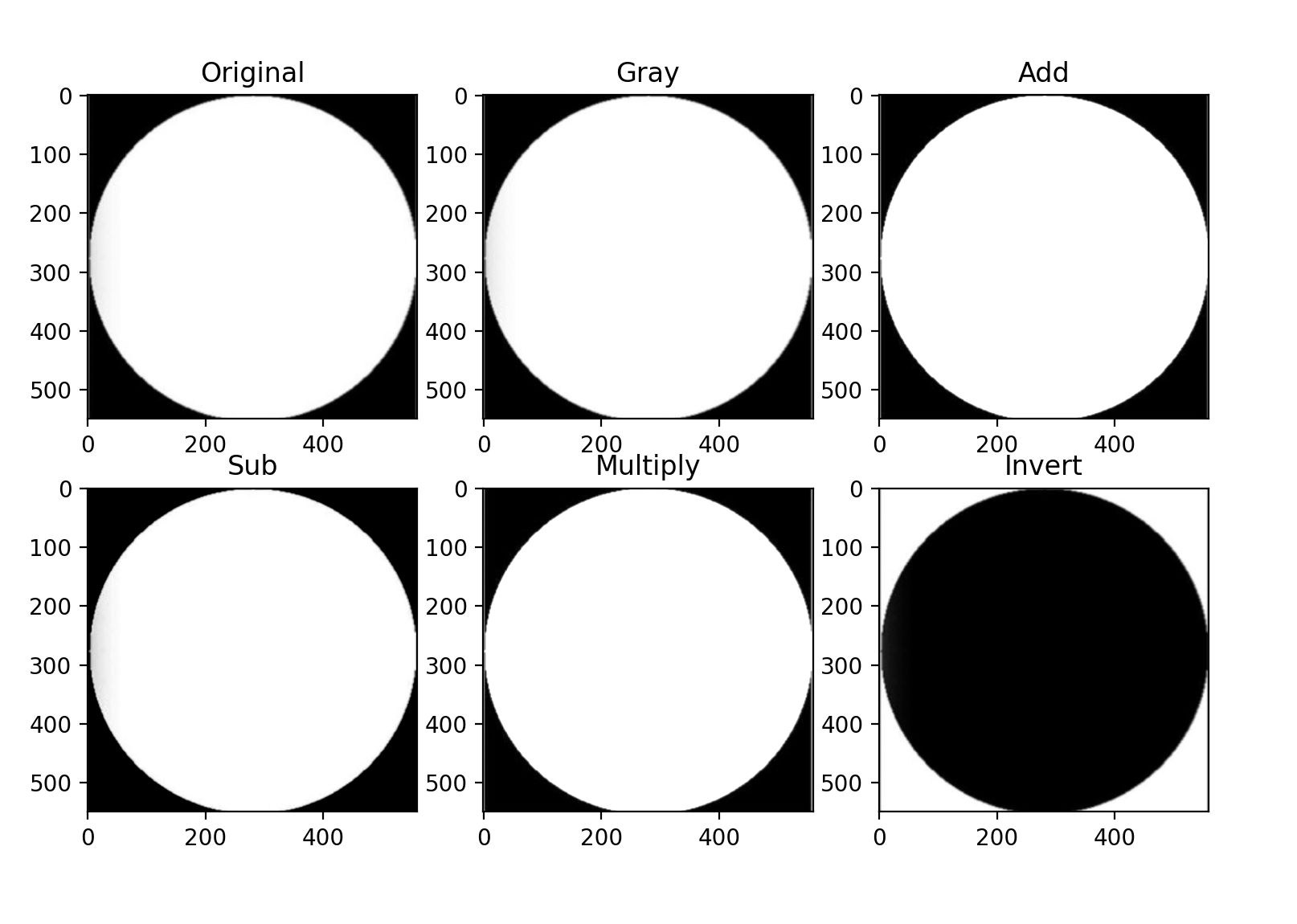
There were 6 pics, showing original, grey scale, add names (which actually some weighted sum), subtract (which is actually some weighted subtract), multiply and an inverted image.

**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.





As you can see that only the inverted image can be seen…

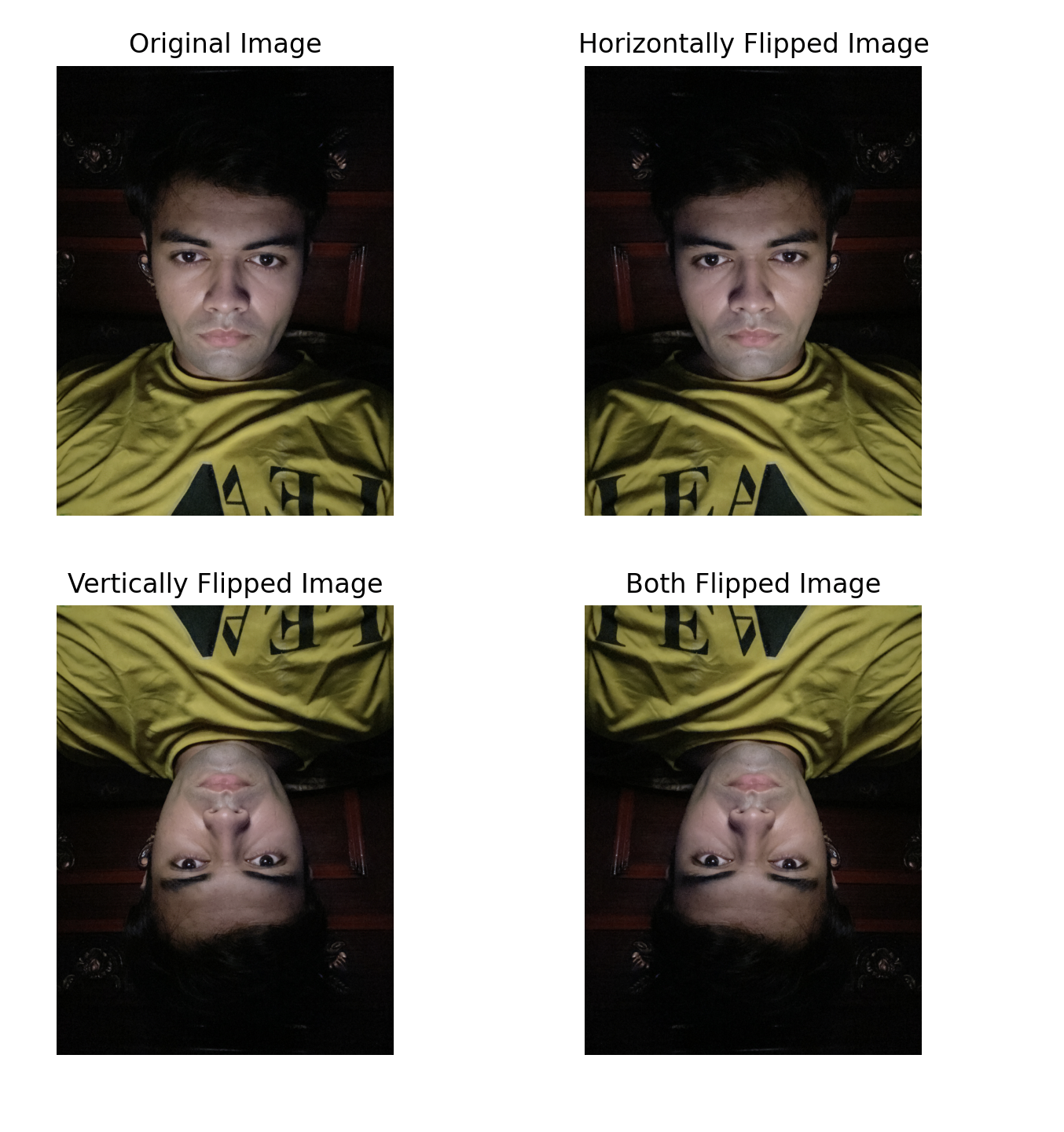
****

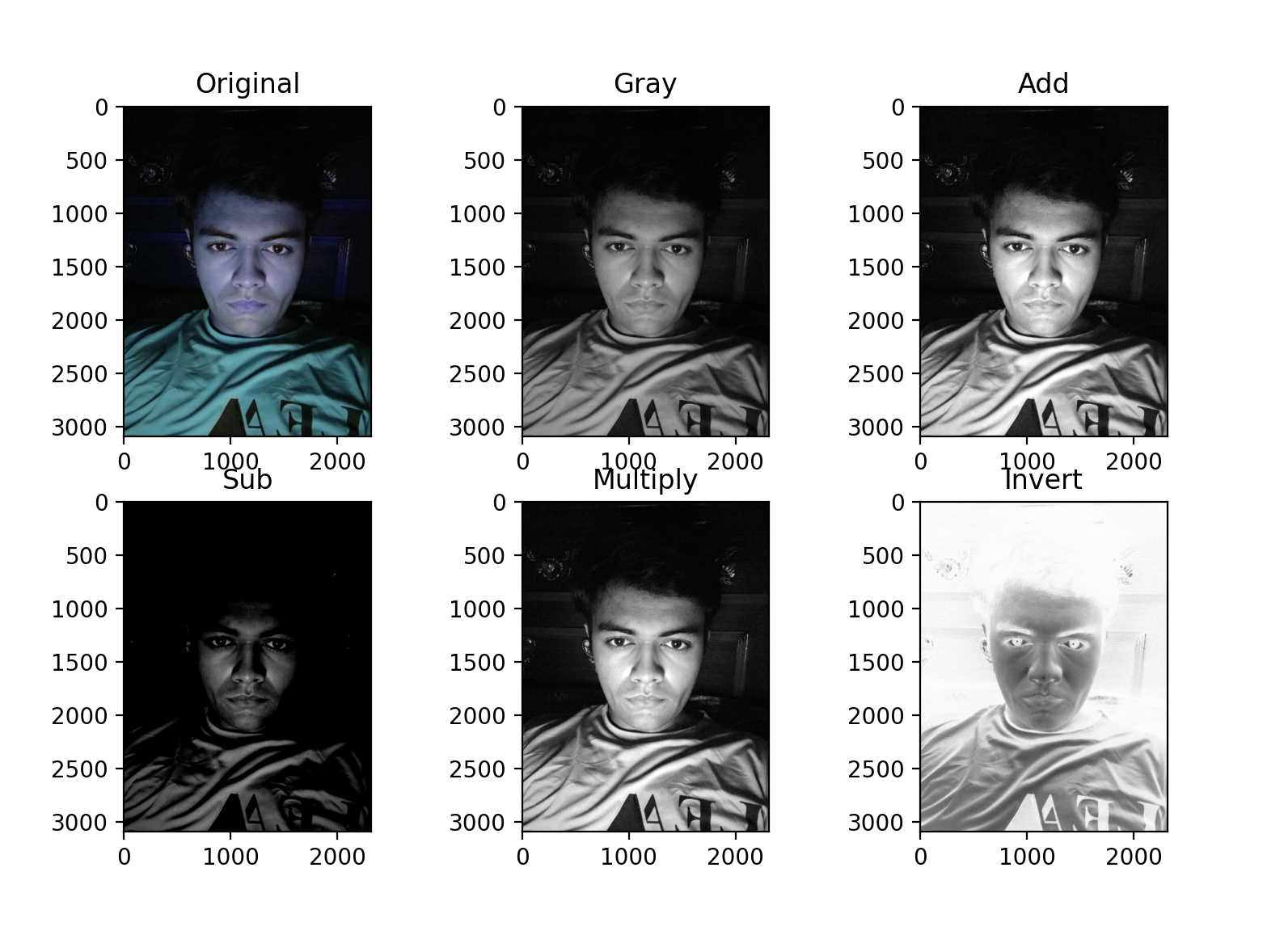
Here again only the inverted image is changed.

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

I am using mac and hence I was not able to capture the image using the program.

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

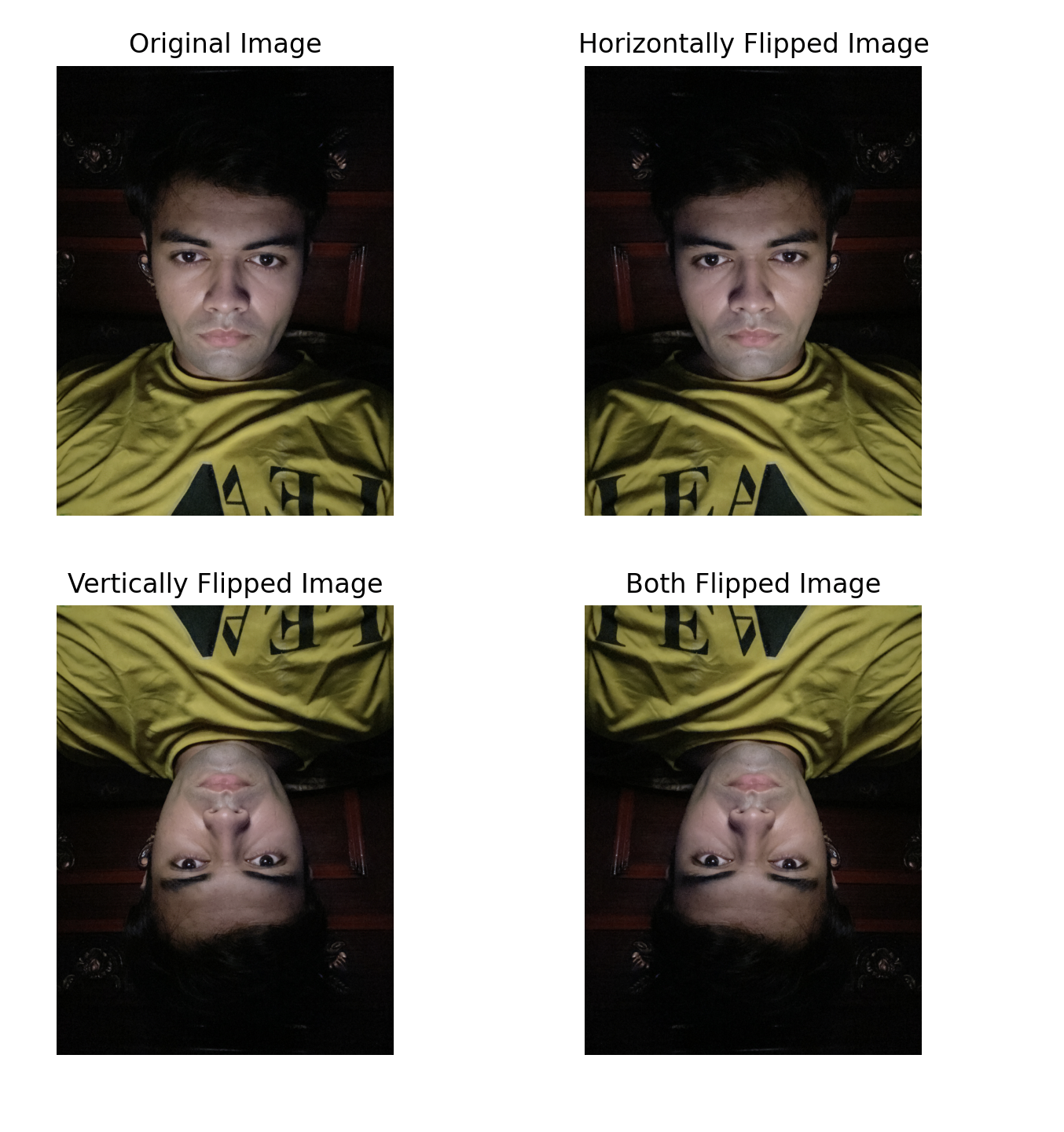


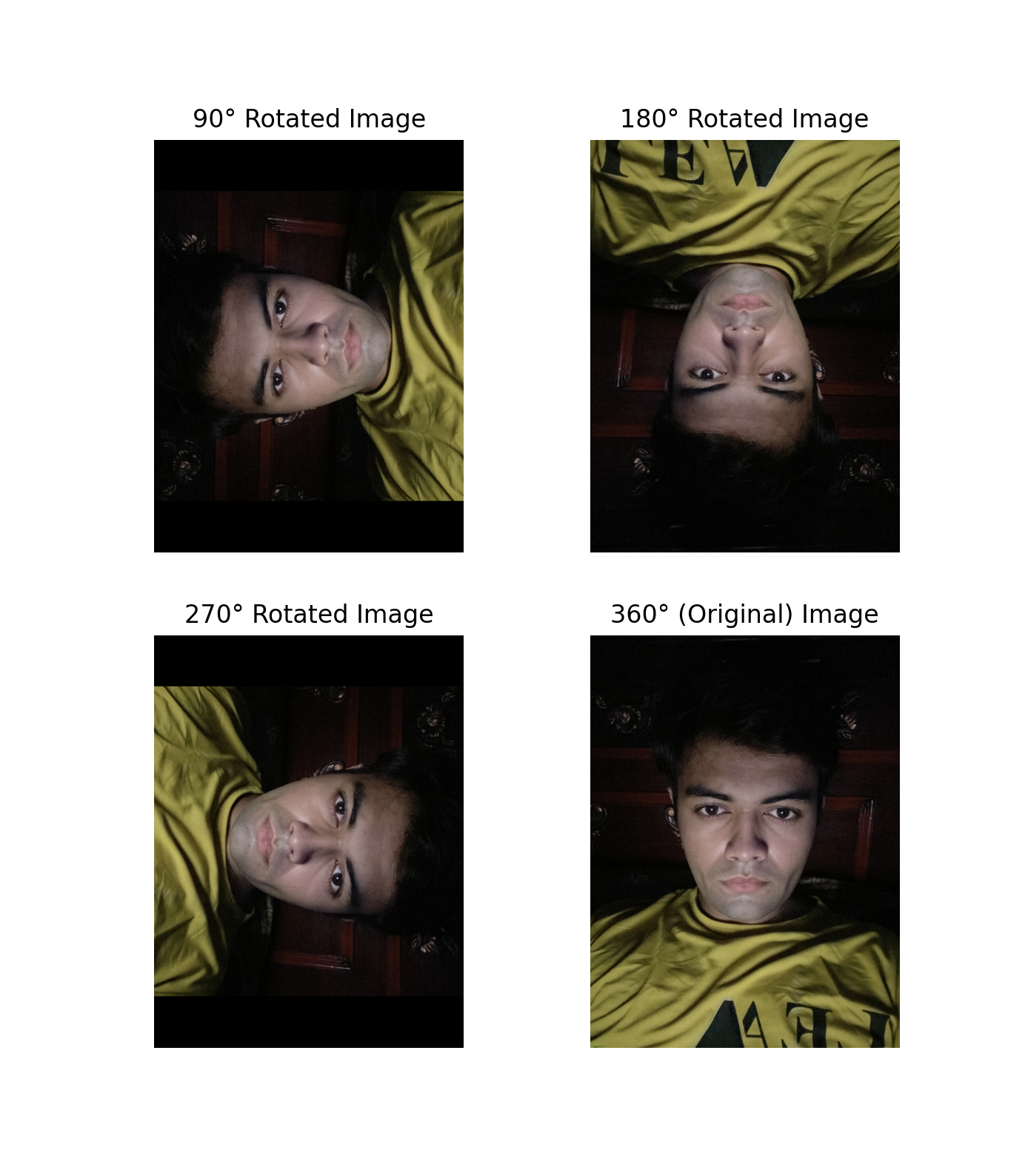
****

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

**Task#6**:

1. Concatenate the original and flipped images into 1 image.
2. Similarly concatenate all the rotated images into 1 image.





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

s

**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)

1. Rotate by 180 degrees clockwise: cv2.ROTATE\_180
2. 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
  1. **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)

* 1. **AND:** A bitwise AND is true *if and only if* both pixels are greater than zero.
  2. **OR:** A bitwise OR is true *if either* of the two pixels is greater than zero.
  3. **XOR:** A bitwise XOR is true *if and only if* one of the two pixels is greater than zero, *but not both.*
  4. **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)