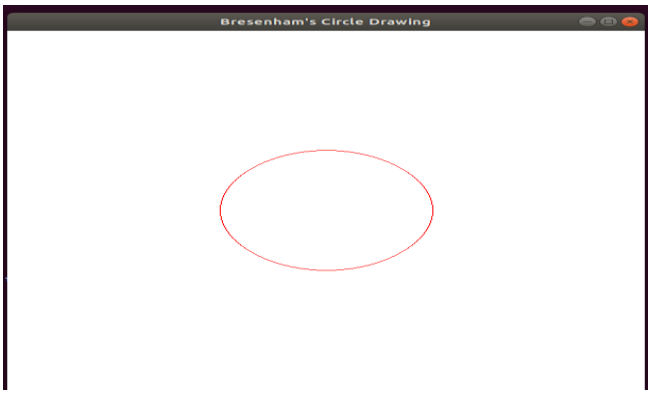
1.Develop an OpenGL program for Bresenhams Circle drawing algorithm with radius 10cm.

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
int xc, yc, r;
void display();
void init();
void plotCirclePoints(int, int, int, int);
void bresenhamCircle(int, int, int);

void main(int argc, char **argv) {
   printf("Enter center of circle (xc, yc): ");
   scanf("%d%d", &xc, &yc);
   printf("Enter radius of circle (in pixels  1cm=10pixels): ");
   scanf("%d", &r);
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(600, 600);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("Bresenham's Circle Drawing");
   init();
   glutDisplayFunc(display);
   glutMainLoop();
}
void init() {
   glClearColor(1, 1, 1, 0);
   glColor3f(1, 0, 0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-300, 300, -300, 300);
   glMatrixMode(GL_MODELVIEW);
}
void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   bresenhamCircle(xc, yc, r);
   glFlush();
```

```
}
void plotCirclePoints(int xc, int yc, int x, int y) {
   glBegin(GL_POINTS);
   glVertex2i(xc + x, yc + y);
   glVertex2i(xc - x, yc + y);
   glVertex2i(xc + x, yc - y);
   glVertex2i(xc - x, yc - y);
   glVertex2i(xc + y, yc + x);
   glVertex2i(xc - y, yc + x);
   glVertex2i(xc + y, yc - x);
   glVertex2i(xc - y, yc - x);
   glEnd();
}
void bresenhamCircle(int xc, int yc, int r) {
   int x = 0, y = r;
   int d = 3 - 2 * r;
   plotCirclePoints(xc, yc, x, y);
   while (x <= y) {
      x++;
      if (d < 0) {
         d = d + 4 * x + 6;
      } else {
         y--;
         d = d + 4 * (x - y) + 10;
      }
      plotCirclePoints(xc, yc, x, y);
   }
}
```

**Output:**



2. Develop an OpenGL program to demonstrates basic 2D transformations like shearing and reflection.

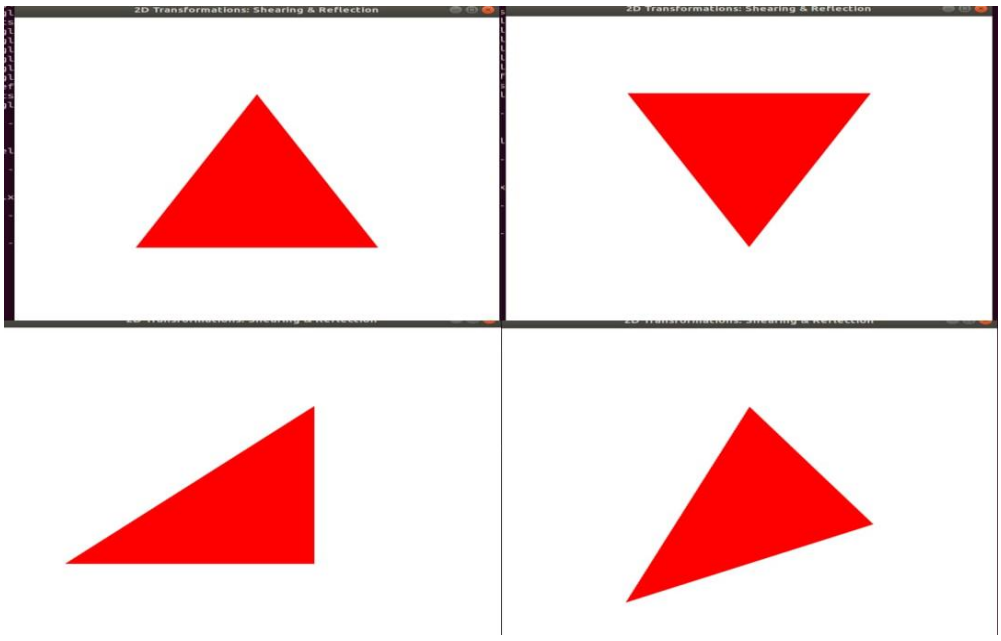
```
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

GLfloat triangle[3][2] = {
   {0.0f, 0.5f},
   {-0.5f, -0.5f},
   {0.5f, -0.5f}
};


int applyShearX = 0;
int applyShearY = 0;
int reflectX = 0;
int reflectY = 0;


void transformAndDraw() {
   glClear(GL_COLOR_BUFFER_BIT);

   glColor3f(1.0, 0.0, 0.0);
   glBegin(GL_TRIANGLES);
   for (int i = 0; i < 3; i++) {
      GLfloat x = triangle[i][0];
      GLfloat y = triangle[i][1];


      if (reflectX) y = -y;
      if (reflectY) x = -x;

      if (applyShearX) x += 0.5 * y;
      if (applyShearY) y += 0.5 * x;

      glVertex2f(x, y);
```

```
    }
    glEnd();
    glFlush();
}


void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 'r':
            reflectX = !reflectX;
            printf("Reflection over X-axis: %s\n", reflectX ? "ON" : "OFF");
            break;
        case 'e':
            reflectY = !reflectY;
            printf("Reflection over Y-axis: %s\n", reflectY ? "ON" : "OFF");
            break;
        case 's':
            applyShearX = !applyShearX;
            printf("Shearing in X-direction: %s\n", applyShearX ? "ON" : "OFF");
            break;
        case 't':
            applyShearY = !applyShearY;
            printf("Shearing in Y-direction: %s\n", applyShearY ? "ON" : "OFF");
            break;
        case 27:
            exit(0);
    }
    glutPostRedisplay();
}


void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-1, 1, -1, 1);
}
```

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("2D Transformations: Shearing & Reflection");

    init();
    glutDisplayFunc(transformAndDraw);
    glutKeyboardFunc(keyboard);
    glutMainLoop();

    return 0;
}
```

**Output:**

```
(base) sahyadri@sahyadri-HP:~$ ./a.out
Reflection over X-axis: ON
Reflection over X-axis: OFF
Reflection over Y-axis: ON
Reflection over Y-axis: OFF
Shearing in X-direction: ON
Shearing in X-direction: OFF
Shearing in Y-direction: ON
```

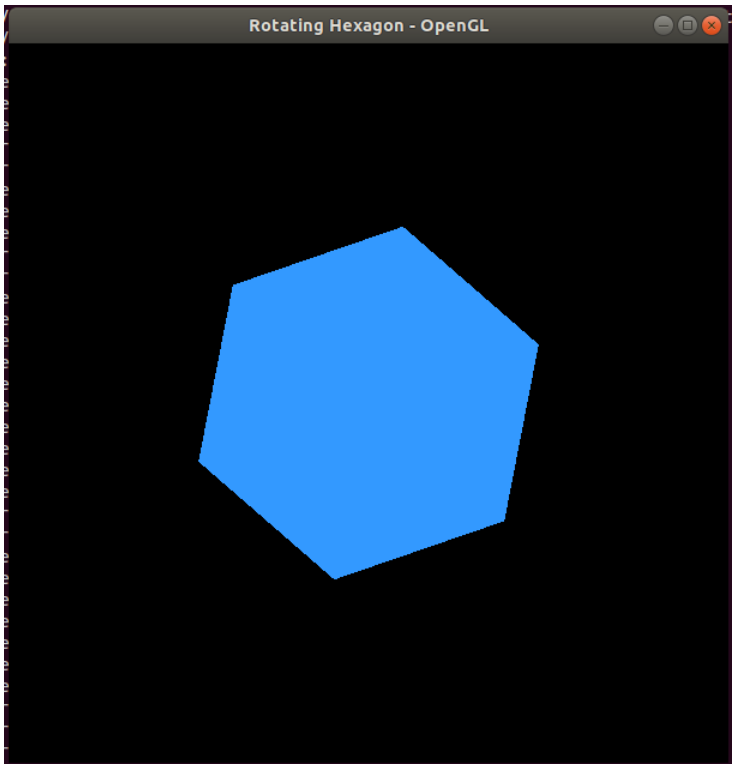


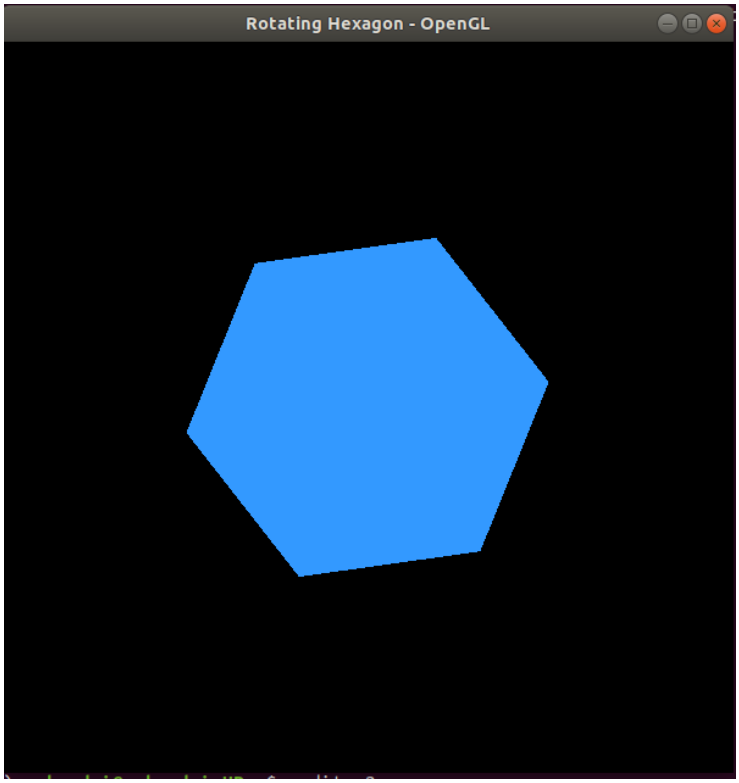
3.Develop an animation program which continuously rotates a regular hexagon in the xy plane about the z axis.

```
#include <GL/glut.h>
#include <math.h>

#define PI 3.14159265358979323846

float angle = 0.0f;

void drawHexagon() {
   int i;
   float radius = 0.5f;

   glBegin(GL_POLYGON);
   for (i = 0; i < 6; ++i) {
      float theta = 2.0f * PI * i / 6;
      float x = radius * cos(theta);
      float y = radius * sin(theta);
      glVertex2f(x, y);
   }
   glEnd();
}

void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   glLoadIdentity();

   glRotatef(angle, 0.0f, 0.0f, 1.0f);

   glColor3f(0.2f, 0.6f, 1.0f);
   drawHexagon();

   glutSwapBuffers();
}
```

```
void timer(int value) {
    angle += 1.0f;
    if (angle >= 360.0f) angle -= 360.0f;

    glutPostRedisplay();
    glutTimerFunc(16, timer, 0);
}

void init() {
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Rotating Hexagon - OpenGL");

    init();

    glutDisplayFunc(display);
    glutTimerFunc(0, timer, 0);

    glutMainLoop();
    return 0;
}
```

**Output:**



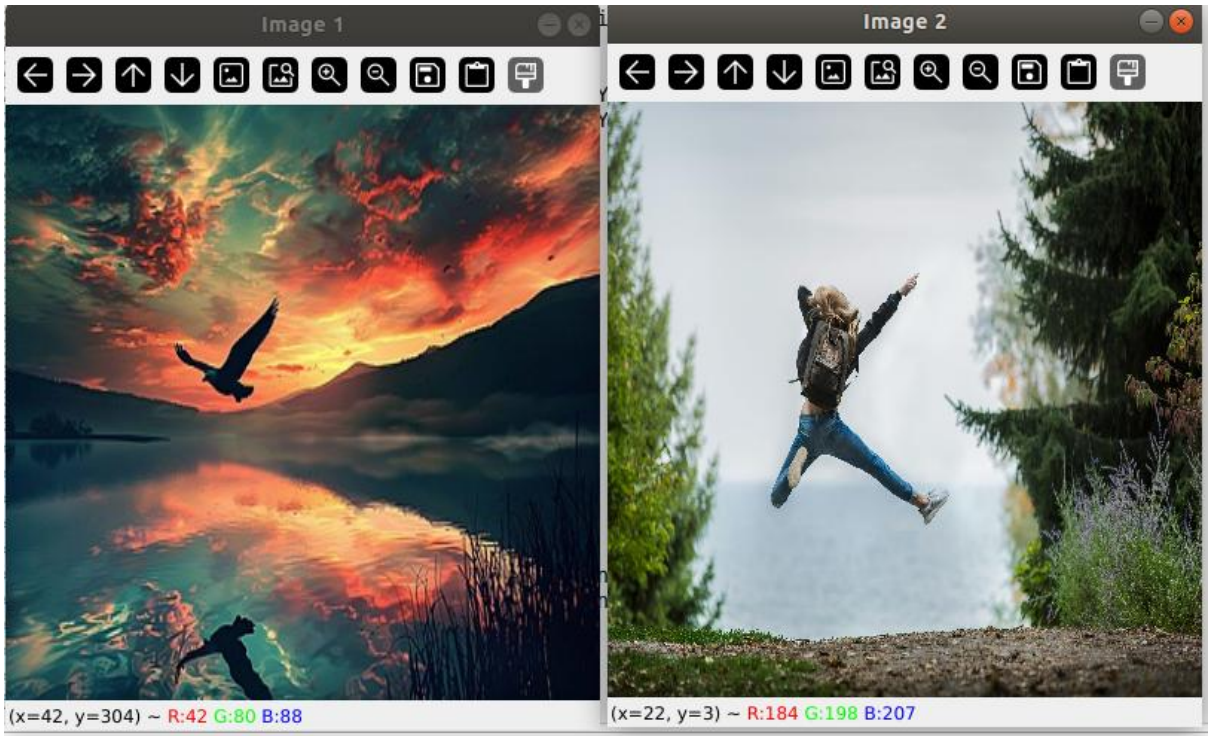



4.Develop a python program for performing arithmetic (add, subtract, multiply, divide) and logical(xor, or, and, not)operations on two image using OpenCV.

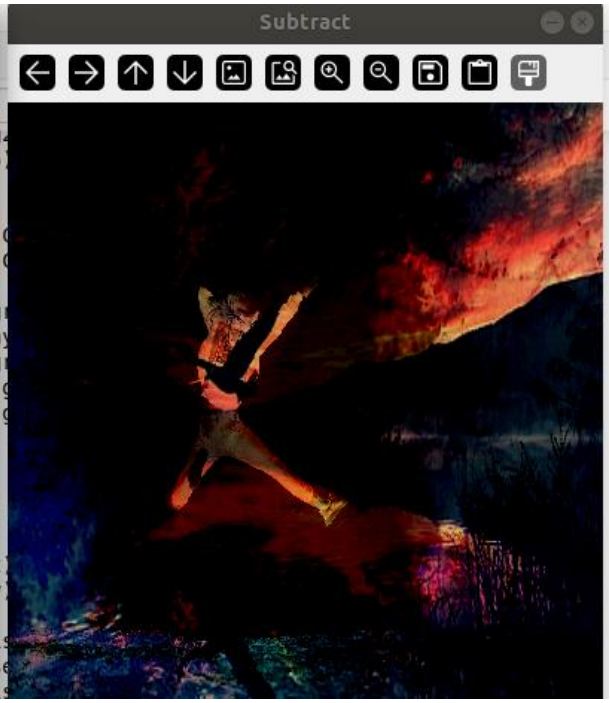
```
import cv2
import numpy as np


img1 = cv2.imread('photo2.jpg')
img2 = cv2.imread('photo3.jpg')


if img1 is None or img2 is None:
    print("Error: One or both images not found.")
    exit()

img1 = cv2.resize(img1, (400, 400))
img2 = cv2.resize(img2, (400, 400))

add = cv2.add(img1, img2)
subtract = cv2.subtract(img1, img2)
multiply = cv2.multiply(img1, img2)

img1_f = img1.astype(np.float32) + 1e-5
img2_f = img2.astype(np.float32) + 1e-5
divide = cv2.divide(img1_f, img2_f)
divide = np.clip(divide, 0, 255).astype(np.uint8)


gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

bitwise_and = cv2.bitwise_and(gray1, gray2)
bitwise_or = cv2.bitwise_or(gray1, gray2)
bitwise_xor = cv2.bitwise_xor(gray1, gray2)
bitwise_not1 = cv2.bitwise_not(gray1)
bitwise_not2 = cv2.bitwise_not(gray2)
```

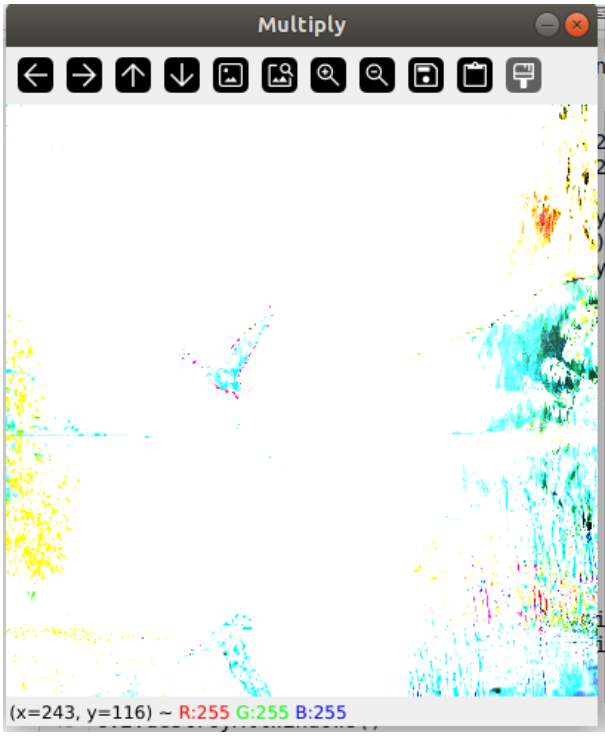
```
cv2.imshow('Image 1', img1)
cv2.imshow('Image 2', img2)
cv2.imshow('Add', add)
cv2.imshow('Subtract', subtract)
cv2.imshow('Multiply', multiply)
cv2.imshow('Divide', divide)
cv2.imshow('Bitwise AND', bitwise_and)
cv2.imshow('Bitwise OR', bitwise_or)
cv2.imshow('Bitwise XOR', bitwise_xor)
cv2.imshow('Bitwise NOT (Image 1)', bitwise_not1)
cv2.imshow('Bitwise NOT (Image 2)', bitwise_not2)

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

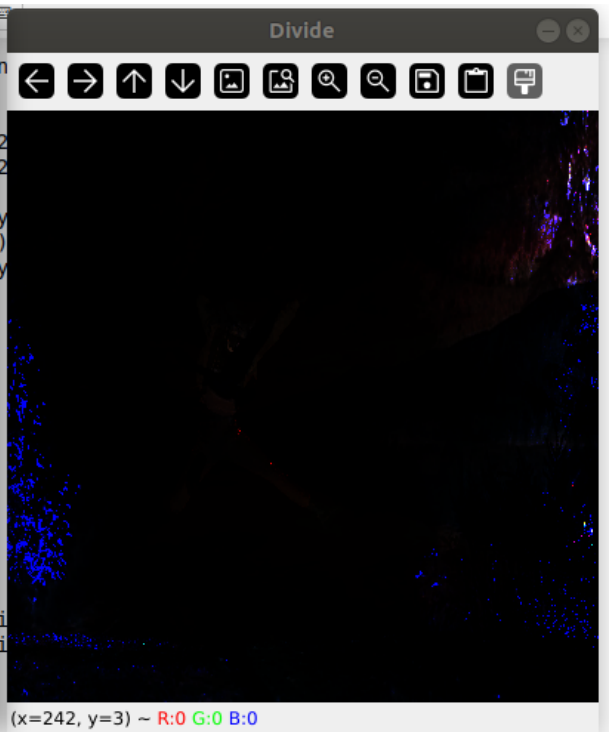
**Output:**




Add


Subtract


Multiply
(x=243, y=116) ~ R:255 G:255 B:255


Divide
(x=242, y=3) ~ R:0 G:0 B:0


Bitwise AND
(x=21, y=135) ~ L:32


Bitwise OR
(x=191, y=62) ~ L:223


Bitwise XOR
(x=11, y=253) ~ L:11

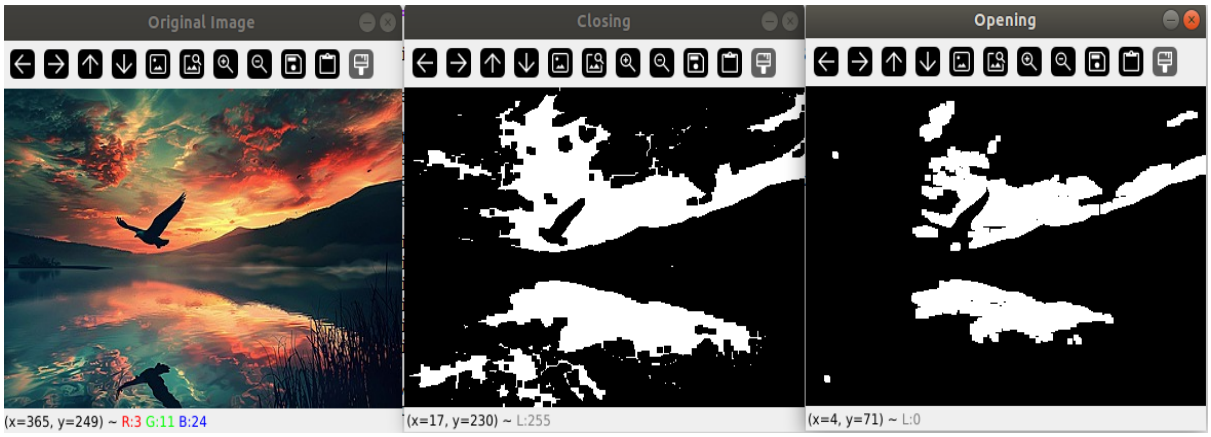

Bitwise NOT (Image 1)
(x=41, y=145) ~ L:200

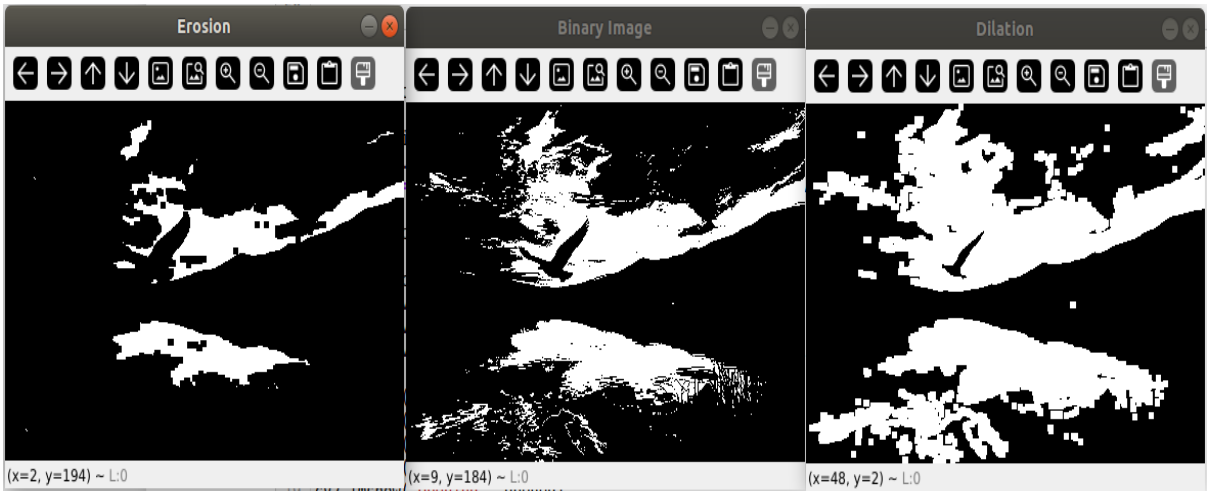

Bitwise NOT (Image 2)
(x=54, y=9) ~ L:56

5.Develop a python program to perform morphological operations (Opening, Closing, Dilation, Erosion)on an image using OpenCV.

```
import cv2
import numpy as np

img = cv2.imread('photo2.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

kernel = np.ones((5, 5), np.uint8)

dilated = cv2.dilate(binary, kernel, iterations=1)
eroded = cv2.erode(binary, kernel, iterations=1)
opened = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)
closed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)

cv2.imshow('Original Image', img)
cv2.imshow('Binary Image', binary)
cv2.imshow('Dilation', dilated)
cv2.imshow('Erosion', eroded)
cv2.imshow('Opening', opened)
cv2.imshow('Closing', closed)

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

**Output:**




Erosion
(x=2, y=194) ~ L:0
Binary Image
(x=9, y=184) ~ L:0
Dilation
(x=48, y=2) ~ L:0