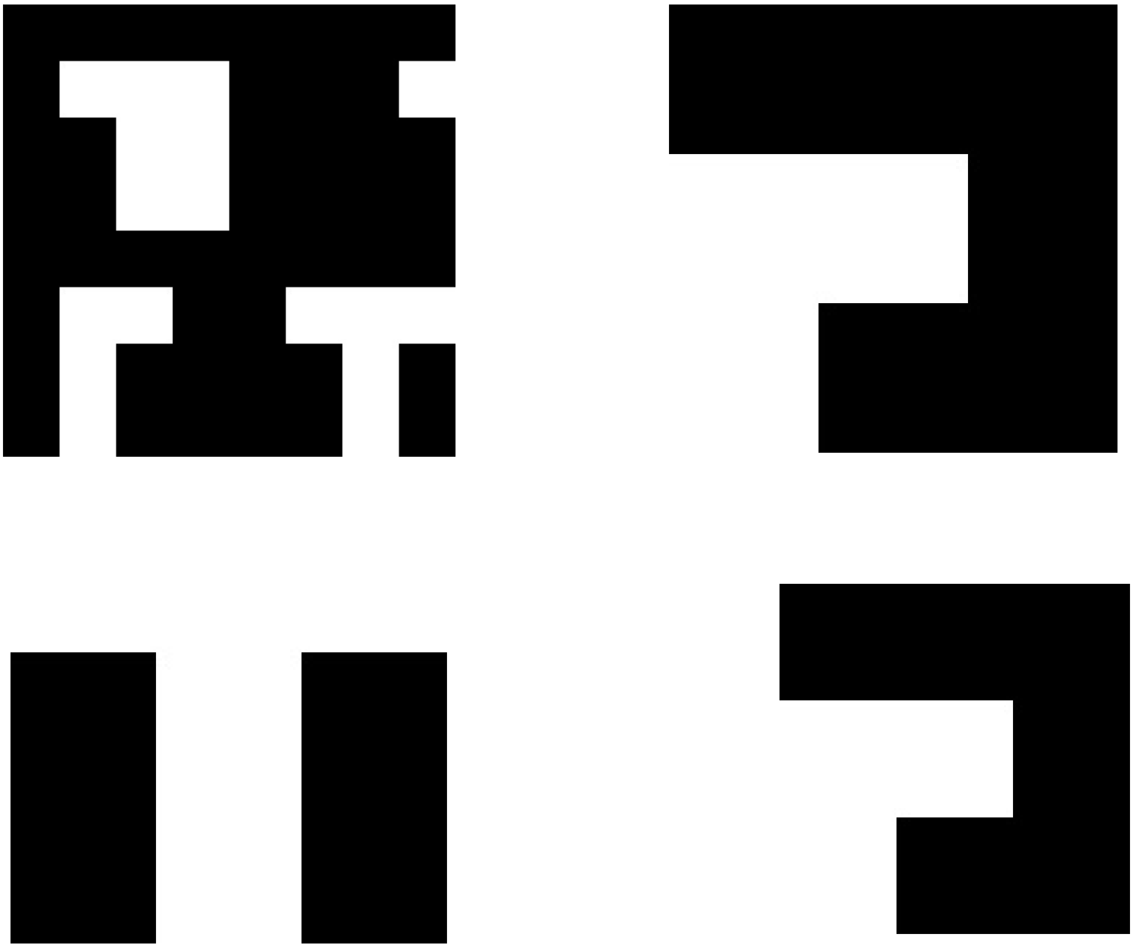
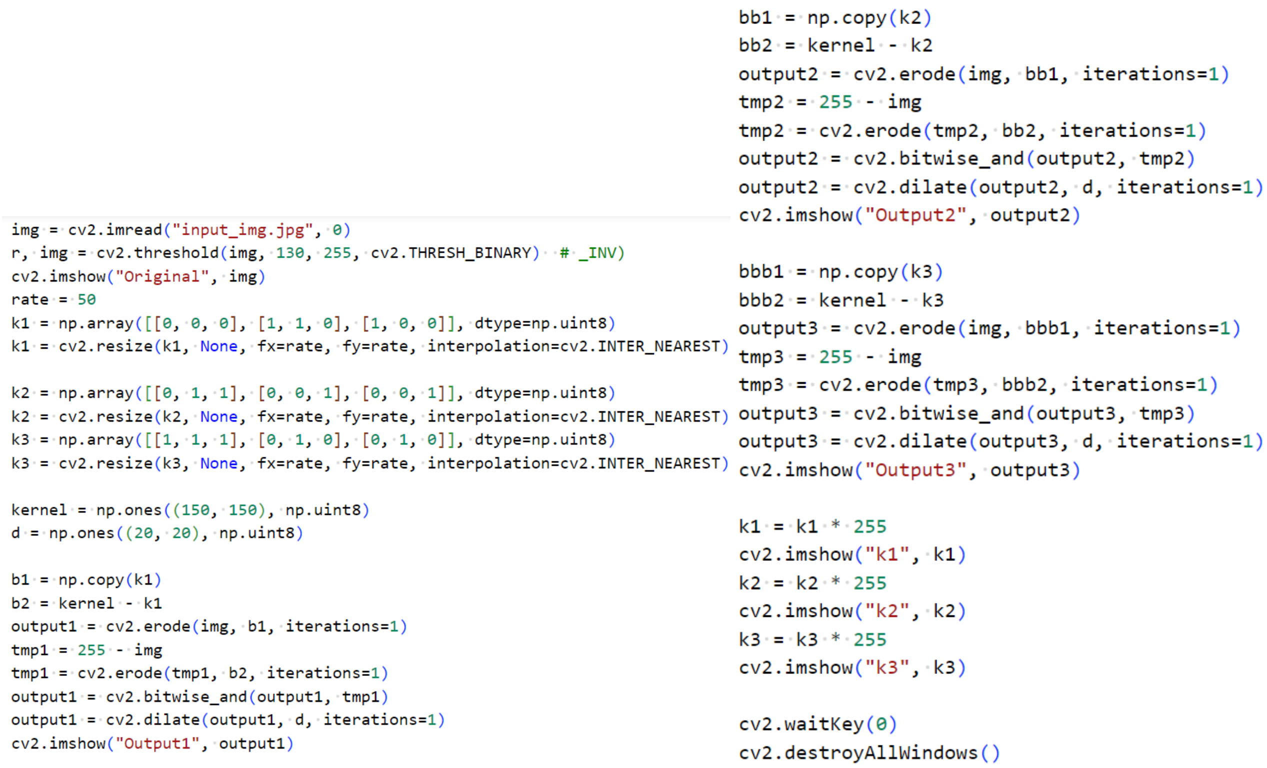


Fig 3.1: Input Image and Structural Elements



```python
img = cv2.imread("input_img.jpg", 0)
r, img = cv2.threshold(img, 130, 255, cv2.THRESH_BINARY)  # _INV)
cv2.imshow("Original", img)
rate = 50
k1 = np.array([[0, 0, 0], [1, 1, 0], [1, 0, 0]], dtype=np.uint8)
k1 = cv2.resize(k1, None, fx=rate, fy=rate, interpolation=cv2.INTER_NEAREST)

k2 = np.array([[0, 1, 1], [0, 0, 1], [0, 0, 1]], dtype=np.uint8)
k2 = cv2.resize(k2, None, fx=rate, fy=rate, interpolation=cv2.INTER_NEAREST)
k3 = np.array([[1, 1, 1], [0, 1, 0], [0, 1, 0]], dtype=np.uint8)
k3 = cv2.resize(k3, None, fx=rate, fy=rate, interpolation=cv2.INTER_NEAREST)

kernel = np.ones((150, 150), np.uint8)
d = np.ones((20, 20), np.uint8)

b1 = np.copy(k1)
b2 = kernel - k1
output1 = cv2.erode(img, b1, iterations=1)
tmp1 = 255 - img
tmp1 = cv2.erode(tmp1, b2, iterations=1)
output1 = cv2.bitwise_and(output1, tmp1)
output1 = cv2.dilate(output1, d, iterations=1)
cv2.imshow("Output1", output1)
```

```python
bb1 = np.copy(k2)
bb2 = kernel - k2
output2 = cv2.erode(img, bb1, iterations=1)
tmp2 = 255 - img
tmp2 = cv2.erode(tmp2, bb2, iterations=1)
output2 = cv2.bitwise_and(output2, tmp2)
output2 = cv2.dilate(output2, d, iterations=1)
cv2.imshow("Output2", output2)

bbb1 = np.copy(k3)
bbb2 = kernel - k3
output3 = cv2.erode(img, bbb1, iterations=1)
tmp3 = 255 - img
tmp3 = cv2.erode(tmp3, bbb2, iterations=1)
output3 = cv2.bitwise_and(output3, tmp3)
output3 = cv2.dilate(output3, d, iterations=1)
cv2.imshow("Output3", output3)

k1 = k1 * 255
cv2.imshow("k1", k1)
k2 = k2 * 255
cv2.imshow("k2", k2)
k3 = k3 * 255
cv2.imshow("k3", k3)

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

Fig 3.2: The code for Hit or Miss Operations

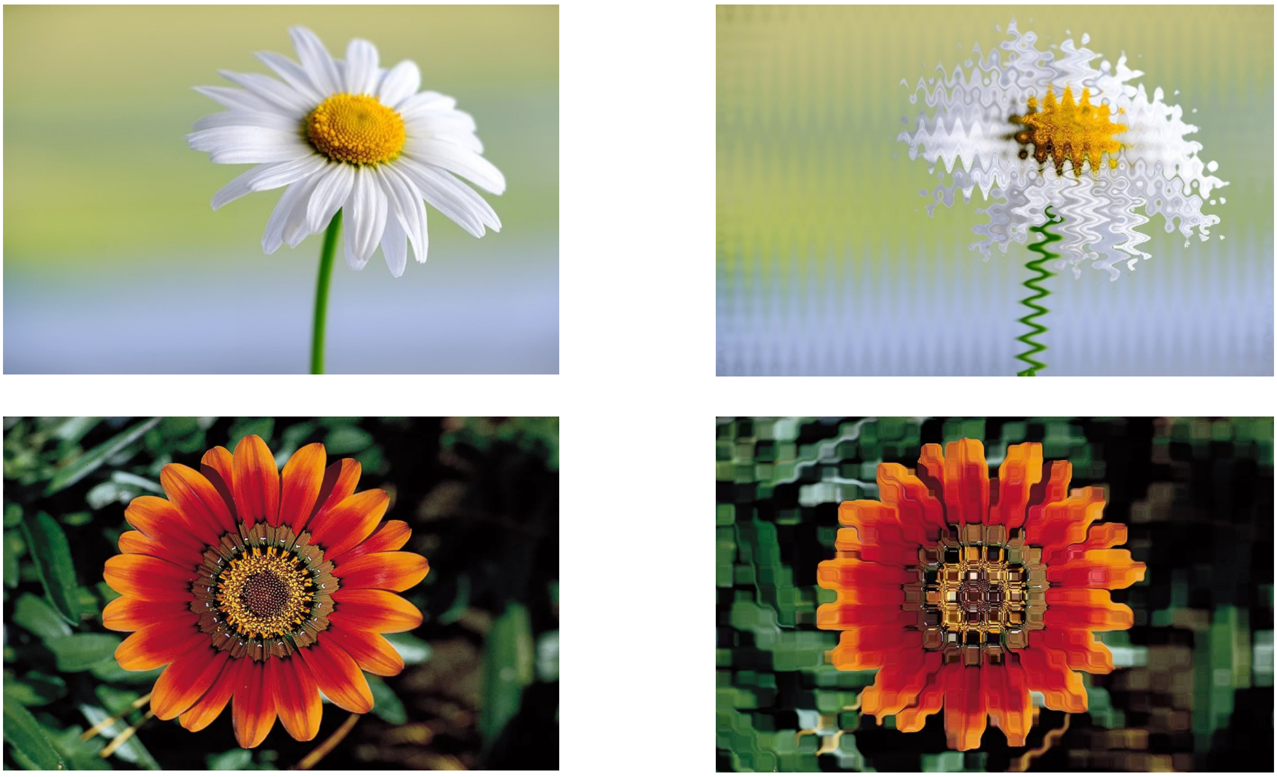
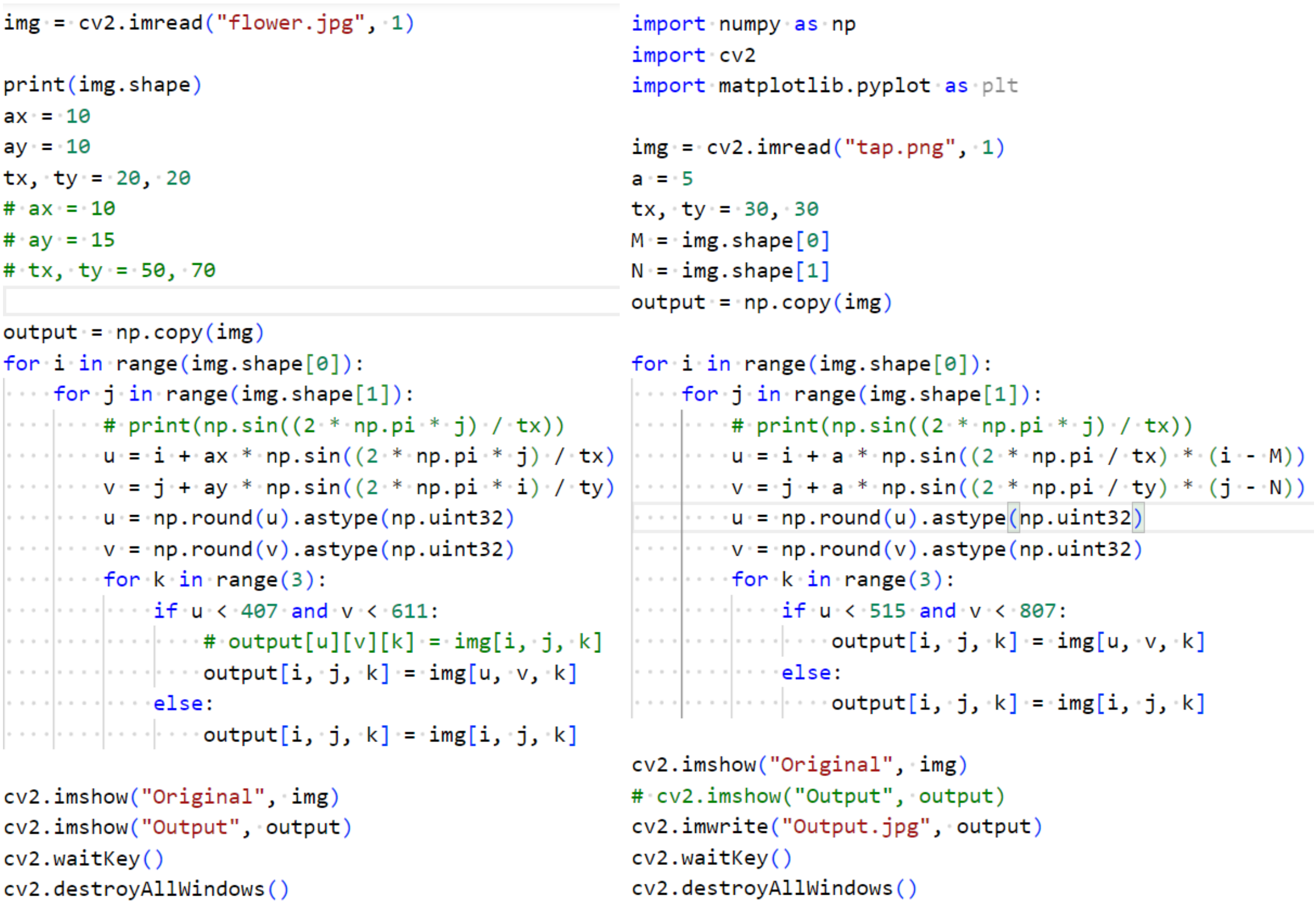


Fig 3.3: The input and output image for Ripple and Tapestry



```
img = cv2.imread("flower.jpg", 1)

print(img.shape)
ax = 10
ay = 10
tx, ty = 20, 20
# ax = 10
# ay = 15
# tx, ty = 50, 70

output = np.copy(img)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        # print(np.sin((2 * np.pi * j) / tx))
        u = i + ax * np.sin((2 * np.pi * j) / tx)
        v = j + ay * np.sin((2 * np.pi * i) / ty)
        u = np.round(u).astype(np.uint32)
        v = np.round(v).astype(np.uint32)
        for k in range(3):
            if u < 407 and v < 611:
                # output[u][v][k] = img[i, j, k]
                output[i, j, k] = img[u, v, k]
            else:
                output[i, j, k] = img[i, j, k]

cv2.imshow("Original", img)
cv2.imshow("Output", output)
cv2.waitKey()
cv2.destroyAllWindows()
```

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

img = cv2.imread("tap.png", 1)
a = 5
tx, ty = 30, 30
M = img.shape[0]
N = img.shape[1]
output = np.copy(img)

for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        # print(np.sin((2 * np.pi * j) / tx))
        u = i + a * np.sin((2 * np.pi / tx) * (i - M))
        v = j + a * np.sin((2 * np.pi / ty) * (j - N))
        u = np.round(u).astype(np.uint32)
        v = np.round(v).astype(np.uint32)
        for k in range(3):
            if u < 515 and v < 807:
                output[i, j, k] = img[u, v, k]
            else:
                output[i, j, k] = img[i, j, k]

cv2.imshow("Original", img)
# cv2.imshow("Output", output)
cv2.imwrite("Output.jpg", output)
cv2.waitKey()
cv2.destroyAllWindows()
```

Fig 3.3: The Code for Ripple and Tapestry Transformations