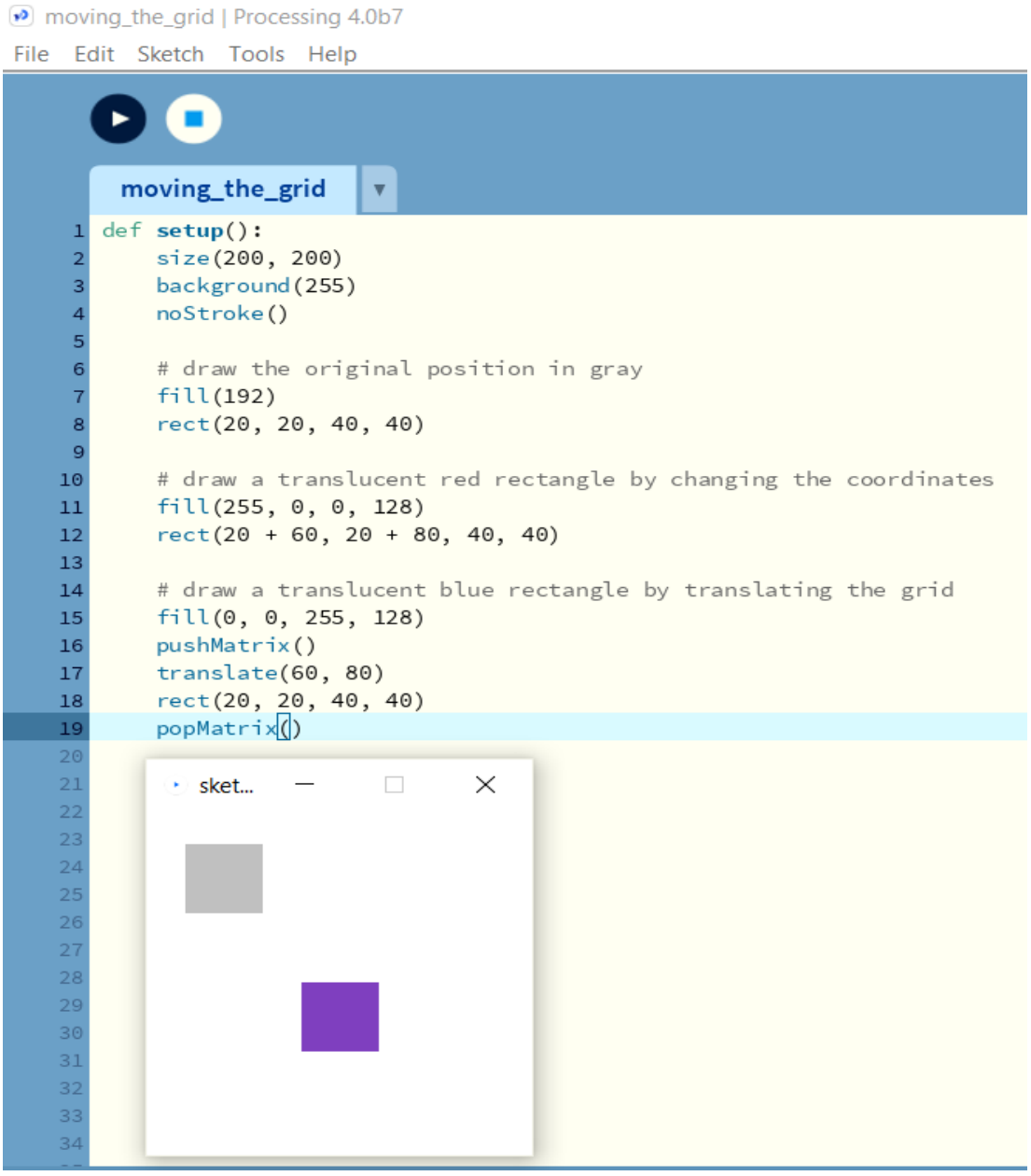
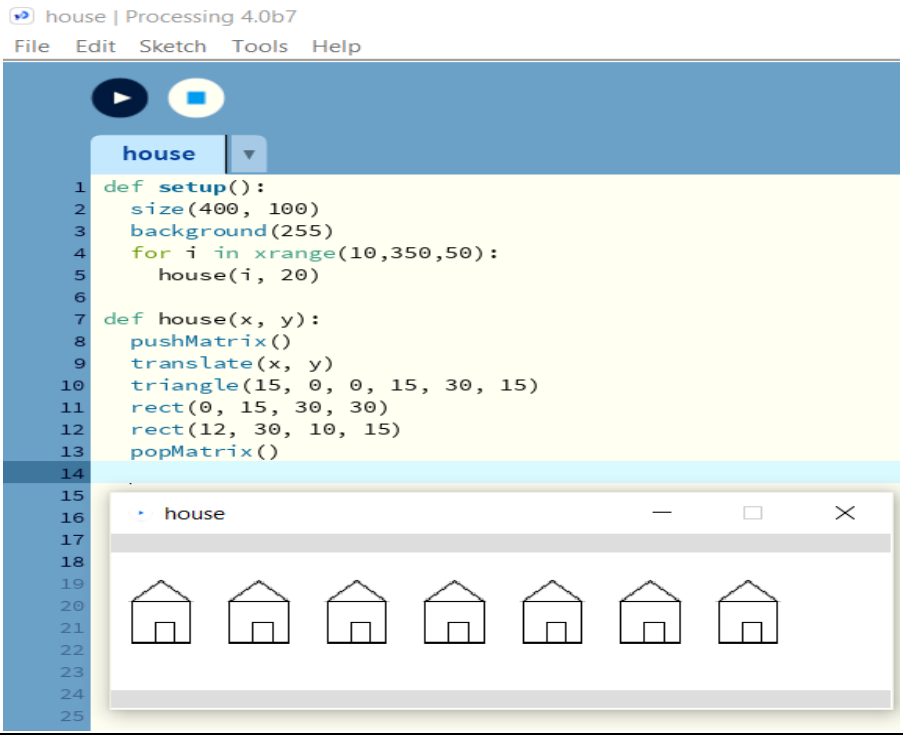
LINEAR ALGEBRA ASSIGNMENT

Unit 3

Implementation of Linear Transformation Techniques:

1.	Apply Linear Transformation on 2D Objects.
a) code and output	Translation: • Moving the Grid 

- drawing the house

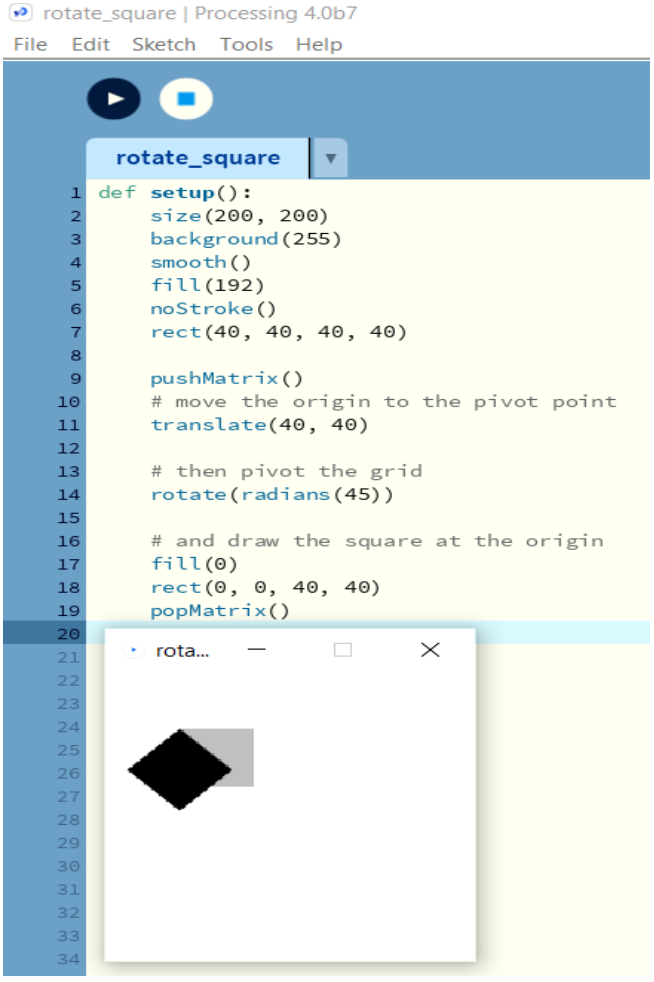


b)

code and output

Rotation

- rotate the square



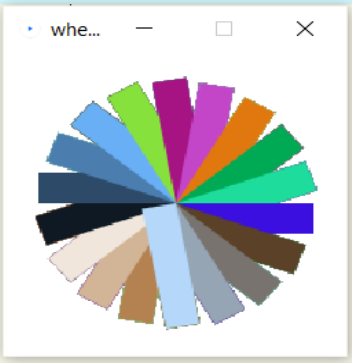
- wheel of colors

wheel_of_colors | Processing 4.0b7

File Edit Sketch Tools Help

wheel_of_colors

```
def setup():
    size(200, 200)
    background(255)
    smooth()
    noStroke()

def draw():
    if (frameCount % 10 == 0):
        fill(frameCount * 3 % 255, frameCount * 5 % 255,
          frameCount * 7 % 255)
        pushMatrix()
        translate(100, 100)
        rotate(radians(frameCount * 2  % 360))
        rect(0, 0, 80, 20)
        popMatrix()
```



c)

code and output

Scaling

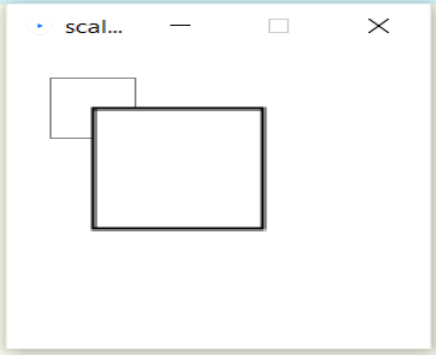
- scaling the grid

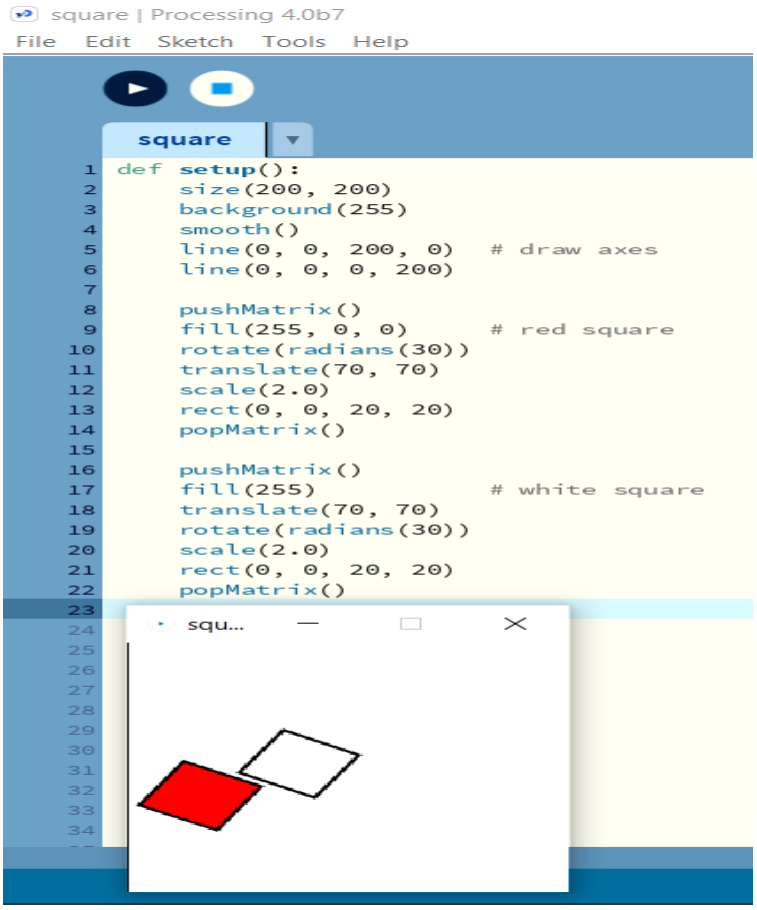
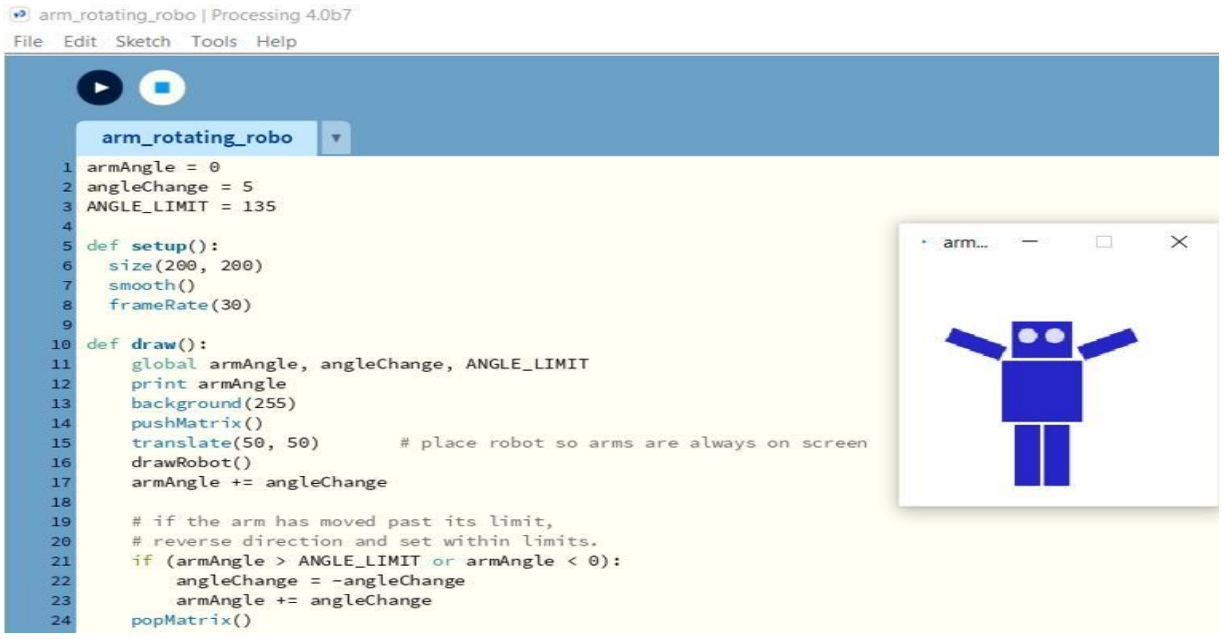
scaling_the_grid | Processing 4.0b7

File Edit Sketch Tools Help

scaling_the_grid

```
def setup():
    size(200,200)
    background(255)

    stroke(128)
    rect(20, 20, 40, 40)

    stroke(0)
    pushMatrix()
    scale(2.0)
    rect(20, 20, 40, 40)
    popMatrix()
```



d)	multiple transformations • A rotation followed by a translate followed by a scale will not give the same results as a translate followed by a rotate by a scale. 
e)	Three-dimensional Transforms • An Arm-Waving Robot 

```
square | Processing 4.0b7
File  Edit  Sketch  Tools  Help
square
def setup():
    size(200, 200)
    background(255)
    smooth()
    line(0, 0, 200, 0)   # draw axes
    line(0, 0, 0, 200)

    pushMatrix()
    fill(255, 0, 0)      # red square
    rotate(radians(30))
    translate(70, 70)
    scale(2.0)
    rect(0, 0, 20, 20)
    popMatrix()

    pushMatrix()
    fill(255)            # white square
    translate(70, 70)
    rotate(radians(30))
    scale(2.0)
    rect(0, 0, 20, 20)
    popMatrix()
```

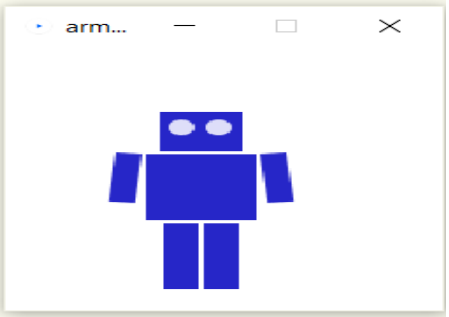
```
arm_rotating_robo | Processing 4.0b7
File  Edit  Sketch  Tools  Help
arm_rotating_robo
armAngle = 0
angleChange = 5
ANGLE_LIMIT = 135

def setup():
  size(200, 200)
  smooth()
  frameRate(30)

def draw():
    global armAngle, angleChange, ANGLE_LIMIT
    print armAngle
    background(255)
    pushMatrix()
    translate(50, 50)       # place robot so arms are always on screen
    drawRobot()
    armAngle += angleChange

    # if the arm has moved past its limit,
    # reverse direction and set within limits.
    if (armAngle > ANGLE_LIMIT or armAngle < 0):
        angleChange = -angleChange
        armAngle += angleChange
    popMatrix()
```

arm_rotating_robo | Processing 4.0b7

File Edit Sketch Tools Help

arm_rotating_robo

```
def drawRobot():
    noStroke()
    fill(38, 38, 200)
    rect(20, 0, 38, 30)      # head
    rect(14, 32, 50, 50)     # body
    drawLeftArm()
    drawRightArm()
    rect(22, 84, 16, 50)     # left leg
    rect(40, 84, 16, 50)     # right leg

    fill(222, 222, 249)
    ellipse(30, 12, 12, 12) # left eye
    ellipse(47, 12, 12, 12) # right eye

def drawLeftArm():
    global armAngle
    pushMatrix()
    translate(12, 32)

    rotate(radians(armAngle))

    rect(-12, 0, 12, 37)     # left arm
    popMatrix()

def drawRightArm():
    global armAngle
    pushMatrix()
    translate(66, 32)

    rotate(radians(-armAngle))

    rect(0, 0, 12, 37)      # right arm
    popMatrix()
```



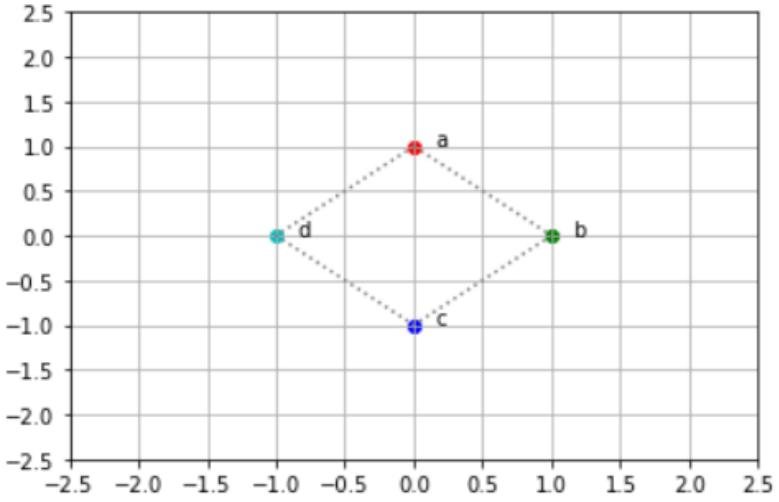
Data Augmentation by applying Linear Transformation.

- Affine transformations

a)

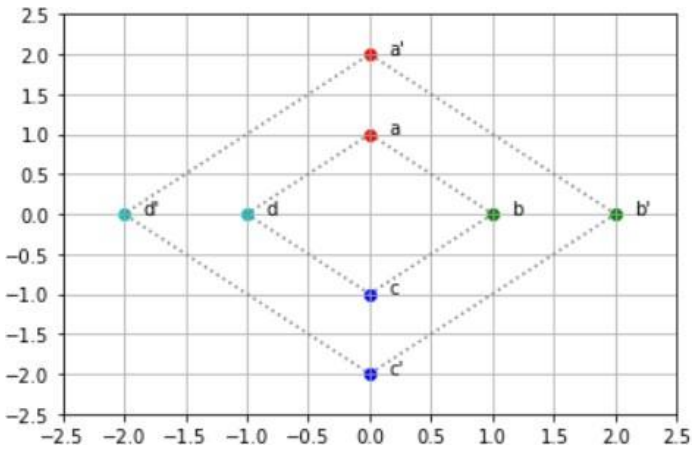
In [1]:

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

# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# 3x3 Identity transformation matrix
I = np.eye(3)
color_lut = 'rgbc'
fig = plt.figure()
ax = plt.gca()
xs = []
ys = []
for row in A:
    output_row = I @ row
    x, y, i = output_row
    xs.append(x)
    ys.append(y)
    i = int(i) # convert float to int for indexing
    c = color_lut[i]
    plt.scatter(x, y, color=c)
    plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
xs.append(xs[0])
ys.append(ys[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



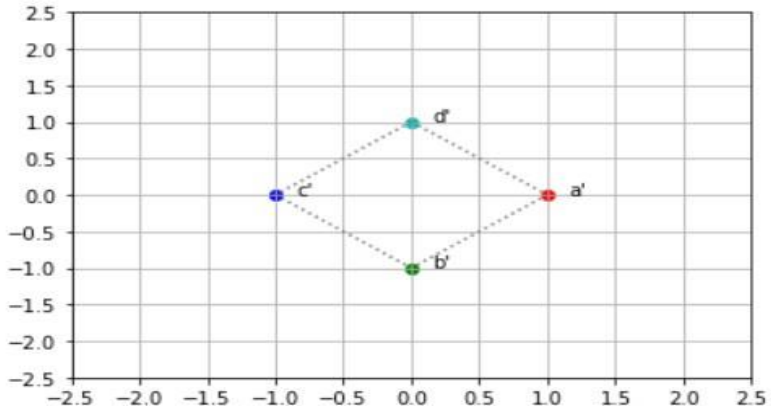
b)

In [2]:
```
# create the scaling transformation matrix
T_s = np.array([[2, 0, 0], [0, 2, 0], [0, 0, 1]])

fig = plt.figure()
ax = plt.gca()
xs_s = []
ys_s = []
for row in A:
    output_row = T_s @ row
    x, y, i = row
    x_s, y_s, i_s = output_row
    xs_s.append(x_s)
    ys_s.append(y_s)
    i, i_s = int(i), int(i_s) # convert float to int for indexing
    c, c_s = color_lut[i], color_lut[i_s] # these are the same but, its good to be explicit
    plt.scatter(x, y, color=c)
    plt.scatter(x_s, y_s, color=c_s)
    plt.text(x + 0.15, y, f"{string.ascii_letters[int(i)]}")
    plt.text(x_s + 0.15, y_s, f"{string.ascii_letters[int(i_s)]}'")

xs_s.append(xs_s[0])
ys_s.append(ys_s[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
plt.plot(xs_s, ys_s, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



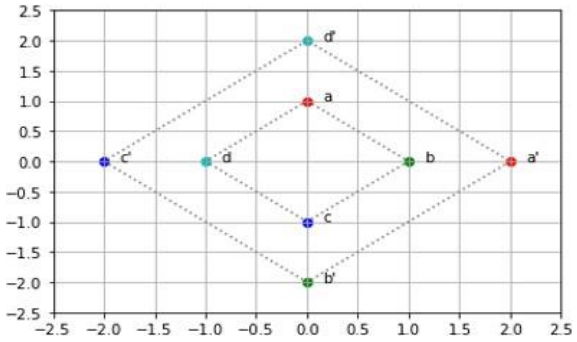
c)

In [3]:
```
# create the rotation transformation matrix
T_r = np.array([[0, 1, 0], [-1, 0, 0], [0, 0, 1]])

fig = plt.figure()
ax = plt.gca()
for row in A:
    output_row = T_r @ row
    x_r, y_r, i_r = output_row
    i_r = int(i_r) # convert float to int for indexing
    c_r = color_lut[i_r] # these are the same but, its good to be explicit
    letter_r = string.ascii_letters[i_r]
    plt.scatter(x_r, y_r, color=c_r)
    plt.text(x_r + 0.15, y_r, f"{letter_r}'")

plt.plot(xs, ys, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



d)

In [4]:
```
# create combined tranformation matrix
T = T_s @ T_r

fig = plt.figure()
ax = plt.gca()

xs_comb = []
ys_comb = []
for row in A:
    output_row = T @ row
    x, y, i = row
    x_comb, y_comb, i_comb = output_row
    xs_comb.append(x_comb)
    ys_comb.append(y_comb)
    i, i_comb = int(i), int(i_comb) # convert float to int for indexing
    c, c_comb = color_lut[i], color_lut[i_comb] # these are the same but, its good to be explicit
    letter, letter_comb = string.ascii_letters[i], string.ascii_letters[i_comb]
    plt.scatter(x, y, color=c)
    plt.scatter(x_comb, y_comb, color=c_comb)
    plt.text(x + 0.15 , y, f"{letter}")
    plt.text(x_comb + 0.15, y_comb, f"{letter_comb}'")
xs_comb.append(xs_comb[0])
ys_comb.append(ys_comb[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
plt.plot(xs_comb, ys_comb, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```

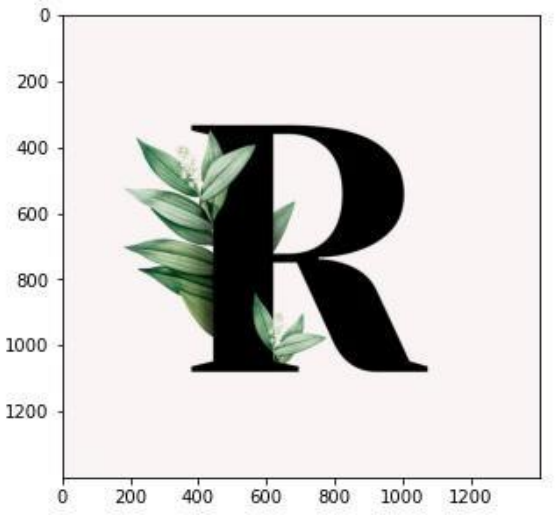


➢ Working With an Image

a)

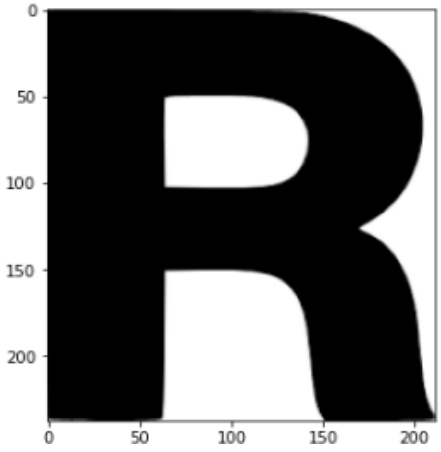
```
In [7]: img = plt.imread('letterR.jpg')
        img.shape #  (1000, 1000, 4)
        plt.figure(figsize=(5, 5))
        plt.imshow(img)
```

```
Out[7]: <matplotlib.image.AxesImage at 0x2f48fa68a30>
```



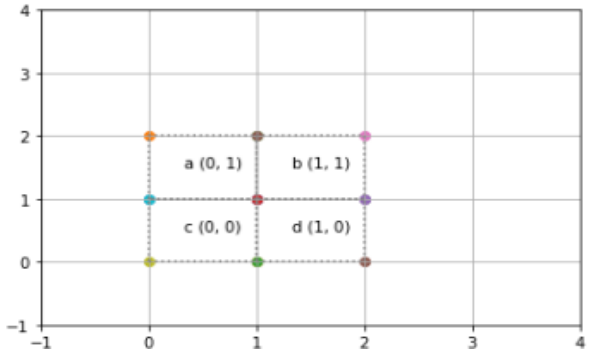
```
img = plt.imread('letterR.png')
img.shape #  (1000, 1000, 4)

plt.figure(figsize=(5, 5))
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x1ad5b22c460>
```



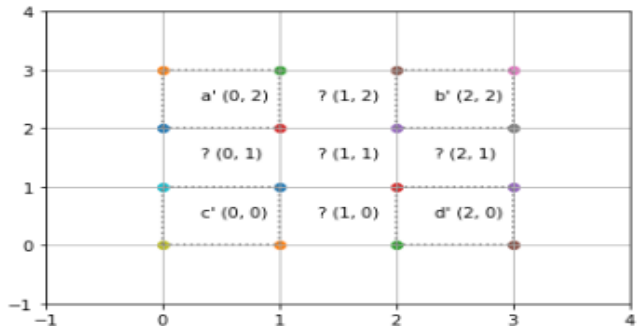
b)

```
def plot_box(plt, x0, y0, txt, w=1, h=1):
    plt.scatter(x0, y0)
    plt.scatter(x0, y0 + h)
    plt.scatter(x0 + w, y0 + h)
    plt.scatter(x0 + w, y0)
    plt.plot([x0, x0, x0 + w, x0 + w, x0], [y0, y0 + h, y0 + h, y0, y0], color="gray", linestyle='dotted')
    plt.text(x0 + (.33 * w), y0 + (.5 * h), txt)

#             x0, y0, letter
a = np.array((0,  1,  0))
b = np.array((1,  1,  1))
c = np.array((0,  0,  2))
d = np.array((1,  0,  3))

A = np.array([a, b, c, d])
fig = plt.figure()
ax = plt.gca()
for pt in A:
    x0, y0, i = I @ pt
    x0, y0, i = int(x0), int(y0), int(i)
    plot_box(plt, x0, y0, f"{string.ascii_letters[int(i)]} ({x0}, {y0})")

ax.set_xticks(np.arange(-1, 5, 1))
ax.set_yticks(np.arange(-1, 5, 1))
plt.grid()
plt.show()
```



c)

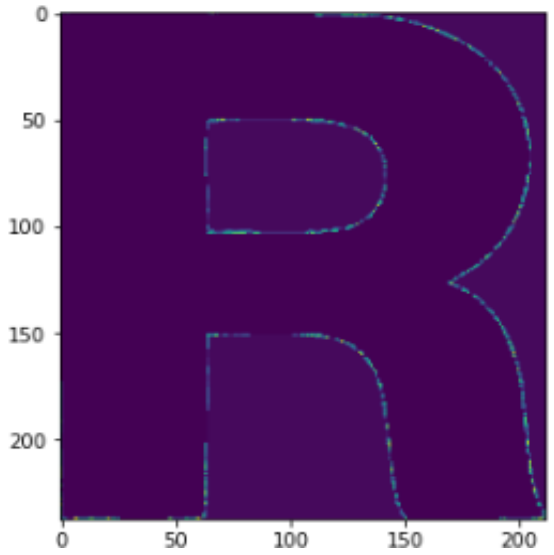
```
fig = plt.figure()
ax = plt.gca()
for pt in A:
    xt, yt, i = T_s @ pt
    xt, yt, i = int(xt), int(yt), int(i)
    plot_box(plt, xt, yt, f"{string.ascii_letters[i]}' ({xt}, {yt})")

delta_w, delta_h = 0.33, 0.5
plt.text(0 + delta_w, 1 + delta_h, "? (0, 1)")
plt.text(1 + delta_w, 0 + delta_h, "? (1, 0)")
plt.text(1 + delta_w, 1 + delta_h, "? (1, 1)")
plt.text(1 + delta_w, 2 + delta_h, "? (1, 2)")
plt.text(2 + delta_w, 1 + delta_h, "? (2, 1)")

ax.set_xticks(np.arange(-1, 5, 1))
ax.set_yticks(np.arange(-1, 5, 1))
plt.grid()
plt.show()
```



d) Using pillows

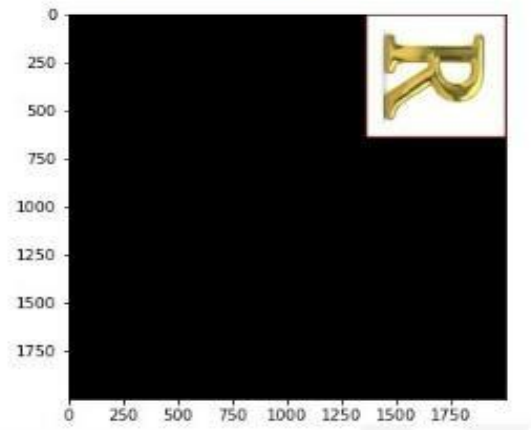
```
from PIL import Image
img = Image.open('letterR.png')
plt.figure(figsize=(5, 5))
plt.imshow(np.asarray(img))
```

```
<matplotlib.image.AxesImage at 0x1ad5e325a90>
```




```
from PIL import Image
img = Image.open('letterR.jpg')
plt.figure(figsize=(5, 5))
plt.imshow(np.asarray(img))
# recenter resultant image
T_pos1000 = np.array([
    [1, 0, 1000],
    [0, 1, 1000],
    [0, 0, 1]])
# rotate - opposite angle
T_rotate = np.array([
    [0, -1, 0],
    [1, 0, 0],
    [0, 0, 1]])
# scale
T_scale = np.array([
    [2, 0, 0],
    [0, 2, 0],
    [0, 0, 1]])
# center original to 0,0
T_neg500 = np.array([
    [1, 0, -500],
    [0, 1, -500],
    [0, 0, 1]])
T = T_pos1000 @ T_rotate @ T_scale @ T_neg500
T_inv = np.linalg.inv(T)
img_transformed = img.transform((2000, 2000), Image.AFFINE, data=T_inv.flatten()[:6], resample=Image.NEAREST)
plt.imshow(np.asarray(img_transformed))
```

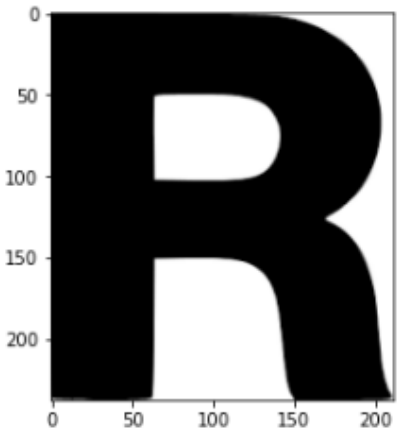
```
<matplotlib.image.AxesImage at 0x223b1363040>
```



e) Using CV2

```
import cv2
img = cv2.imread('letterR.png')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x1ad5e86c6d0>
```




```
T_opencv = np.float32(T.flatten()[:6].reshape(2,3))
img_transformed = cv2.warpAffine(img, T_opencv, (2000, 2000))
plt.imshow(cv2.cvtColor(img_transformed, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x1ad5e847760>
```