CG LAB Assignment 8

NAME: S.B.ASHRITH

Registration No: 20BCE7236

2D Transformations:

1. **Write programs to implement basic 2D transformations (translation, rotation, scaling, reflection, shear, etc).**

Code:

```
import numpy as np

# Define the translation
matrix tx = 2 # translation
in x-axis ty = 3 # translation
in y-axis
    [1, 0, tx],
    [0, 1, ty],
    [0, 0, 1]
])

# Define the original
coordinates
    [0, 0, 1],
    [1, 0, 1],
    [0, 1, 1]
])

# Apply the translation matrix to the original coordinates
translated_coords = np.matmul(translation_matrix,
original_coords.T).T print(translated_coords)
```

```
PS D:\Computer Graphics Lab> python -u "d:\Computer Graphics Lab\Translation.py"
[[2 3 1]
 [3 3 1]
 [2 4 1]]
PS D:\Computer Graphics Lab>
```

```
import numpy as np

# Define the rotation matrix
theta = np.pi/2 # rotation angle in
radians rotation_matrix = np.array([
        [np.cos(theta), -np.sin(theta),
    0],
        [np.sin(theta), np.cos(theta),
    0],
        [0, 0, 1]
])
    [0, 0, 1],
    [1, 0, 1],
    [0, 1, 1]
])

# Apply the rotation matrix to the original coordinates
rotated_coords = np.matmul(rotation_matrix,
original_coords.T).T print(rotated_coords)
```

```
                              > python -u "d:\Computer Graphics Lab\Rotation.py"
[[ 0.000000e+00  0.000000e+00  1.000000e+00]
 [ 6.123234e-17  1.000000e+00  1.000000e+00]
 [-1.000000e+00  6.123234e-17  1.000000e+00]]
PS D:\Computer Graphics Lab>
```

```
import numpy as np

# Define the scaling matrix
sx = 2  # scaling factor in x-axis
sy = 0.5 # scaling factor in y-axis
scaling_matrix = np.array([
    [sx, 0, 0],
    [0, sy, 0],
    [0, 0, 1]
```

```
])

# Define the original
coordinates original_coords =
    [0, 0, 1],
    [1, 0, 1],
    [0, 1, 1]
])

# Apply the scaling matrix to the original coordinates
scaled_coords = np.matmul(scaling_matrix,
original_coords.T).T print(scaled_coords)
```

```
> python -u "d:\Computer Graphics Lab\Scalling.py"
[[0.  0.  1. ]
 [2.  0.  1. ]
 [0.  0.5 1. ]]
PS D:\Computer Graphics Lab>
```

```
import numpy as np

# Define the reflection
matrix reflection_matrix =
np.array([
        [-1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]
])

# Define the original
    [0, 0, 1],
    [1, 0, 1],
    [0, 1, 1]
])

# Apply the reflection matrix to the original coordinates
reflected_coords = np.matmul(reflection_matrix,
original_coords.T).T print(reflected_coords)
```

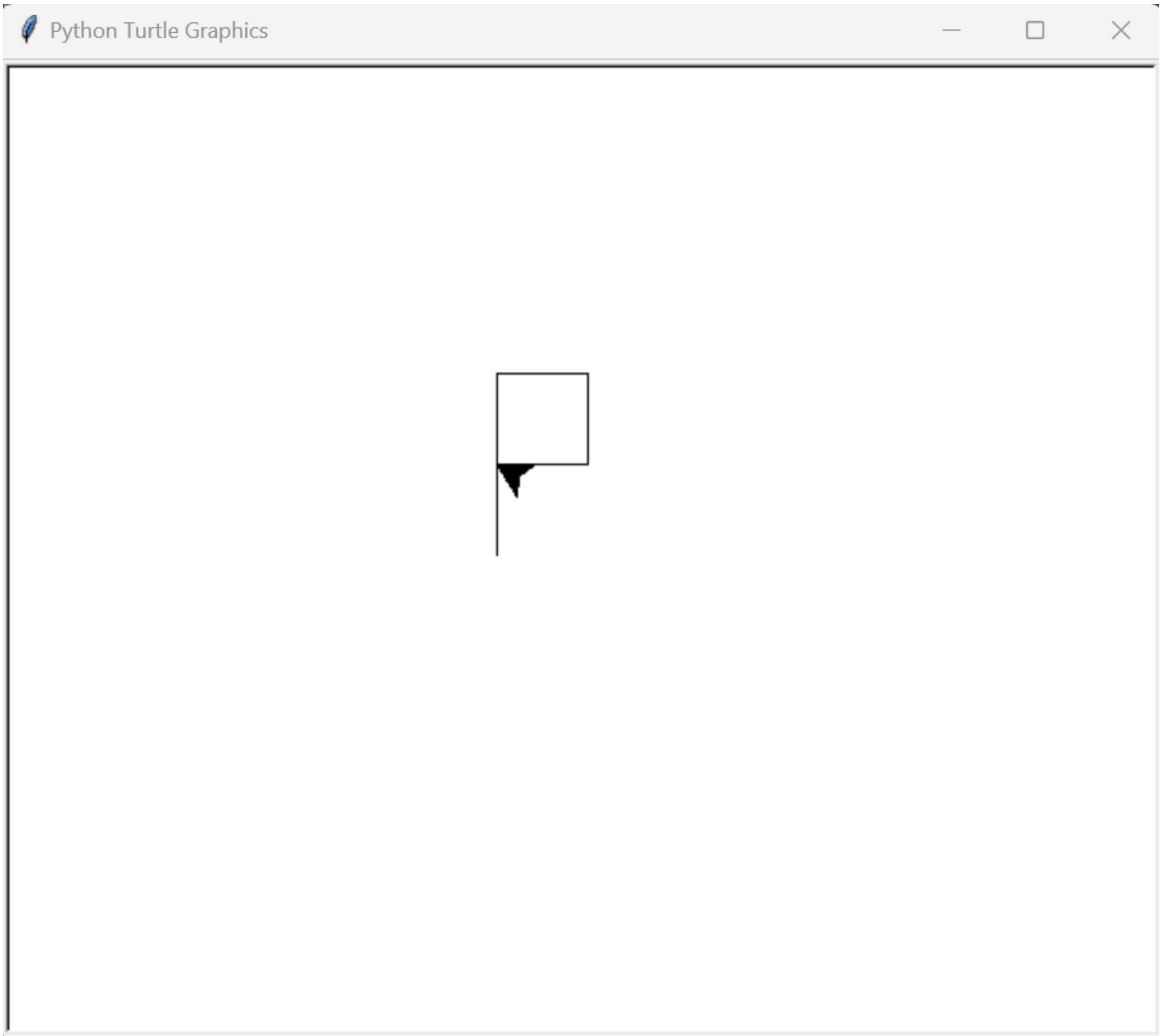
```
> python -u "d:\Computer Graphics Lab\Reflection.py"
[[ 0  0  1]
 [-1  0  1]
 [ 0  1  1]]
PS D:\Computer Graphics Lab> 
```

2. **Write programs to draw first letter of your name and rotate it by 30 degree clockwise and also do scaling 2 times.**

Code:

```
import turtle

# create a turtle object
t = turtle.Turtle()

# draw the letter P
t.penup()
t.goto(-50, 0)
t.pendown()
t.left(90)
t.forward(100)
t.right(90)
t.forward(50)
t.right(90)
t.forward(50)
t.right(90)
t.forward(50)

# rotate the turtle by 30 degrees clockwise
t.right(30)

# scale the turtle by a factor of 2
t.shapesize(2)

# keep the turtle window open until it is closed manually
turtle.done()
```

output:



3D Transformations:

3. **Write programs to implement basic 3D transformations (translation, rotation, scaling, reflection, shear, etc).**

Code:

```
import numpy as np

# Define a 3D object as a collection of
points object_points = np.array([
  [0, 0, 0],
  [0, 1, 0],
  [1, 1, 0],
  [1, 0, 0],
  [0, 0, 1],
  [0, 1, 1],
```

```
    [1, 1, 1],
    [1, 0, 1],
])

# Translation
translation_vector = np.array([2, 3, 4])
translation_matrix = np.eye(4)
translation_matrix[:3, 3] = translation_vector
translated_points = np.array([translation_matrix.dot(np.append(point, 1))[:3] for point
in object_points])

# Rotation
theta_x = np.pi / 4 # 45 degrees
theta_y = np.pi / 6 # 30 degrees
theta_z = np.pi / 3 # 60 degrees
rotation_x = np.array([
    [1, 0, 0, 0],
    [0, np.cos(theta_x), -np.sin(theta_x), 0],
    [0, np.sin(theta_x), np.cos(theta_x), 0],
    [0, 0, 0, 1]
])
rotation_y = np.array([
    [np.cos(theta_y), 0, np.sin(theta_y), 0],
    [0, 1, 0, 0],
    [-np.sin(theta_y), 0, np.cos(theta_y), 0],
    [0, 0, 0, 1]
])
rotation_z = np.array([
    [np.cos(theta_z), -np.sin(theta_z), 0, 0],
    [np.sin(theta_z), np.cos(theta_z), 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
])
rotation_matrix = rotation_z.dot(rotation_y).dot(rotation_x)
rotated_points = np.array([rotation_matrix.dot(np.append(point, 1))[:3] for point in
translated_points])

# Scaling
scaling_factor = 2
scaling_matrix = np.diag([scaling_factor, scaling_factor, scaling_factor, 1])
scaled_points = np.array([scaling_matrix.dot(np.append(point, 1))[:3] for point in
rotated_points])

# Reflection
reflection_plane = np.array([1, 0, 0])
reflection_matrix = np.eye(4) - 2 * np.outer(reflection_plane, reflection_plane)
reflected_points = np.array([reflection_matrix.dot(np.append(point, 1))[:3] for point
in scaled_points])
```

```
# Shearing
    shearing_matrix =
    np.array([ [1, 0, 0,
    0],
    [0.5, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
])
sheared_points = np.array([shearing_matrix.dot(np.append(point, 1))[:3] for point in
reflected_points])

print("Original points:\n", object_points)
print("Transformed points:\n", sheared_points)
```

4. **Write programs to draw a cube and rotate it by 45 degree anti-**

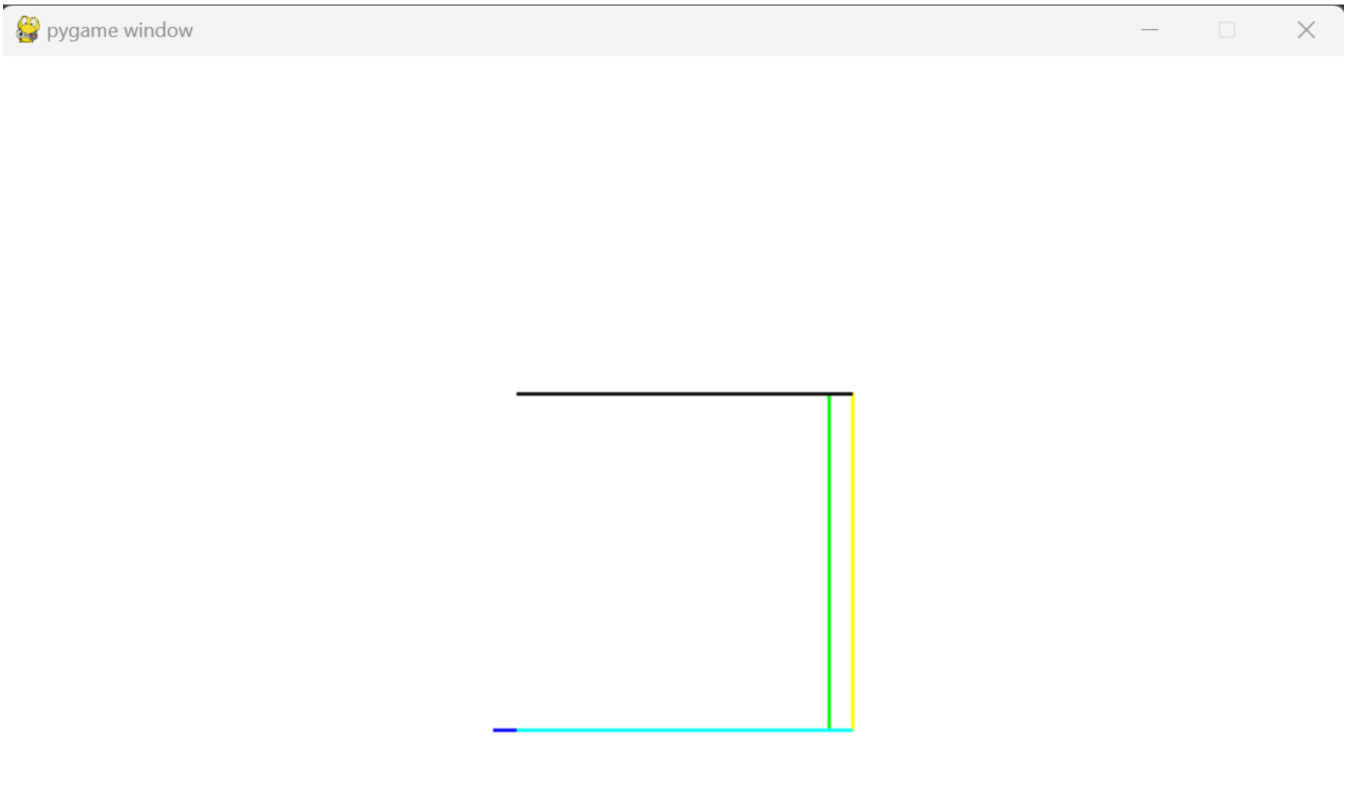
clockwise. Code:

```
import pygame
import math

# initialize
Pygame
pygame.init()

# set the screen size
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width,
screen_height))

  (100, 100, 100),
  (100, 100, -100),
  (100, -100, 100),
  (100, -100, -100),
  (-100, 100, 100),
  (-100, 100, -100),
  (-100, -100, 100),
  (-100, -100, -100)
]

# define the cube
edges cube_edges = [
  (0, 1),
  (0, 2),
  (0, 4),
  (1, 3),
  (1, 5),
```

```
    (2, 3),
    (2, 6),
    (3, 7),
    (4, 5),
    (4, 6),
    (5, 7),
    (6, 7)
]

# define the cube colors
cube_colors = [
    (255, 0, 0),
    (0, 255, 0),
    (0, 0, 255),
    (255, 255, 0),
    (0, 255, 255),
    (255, 0, 255),
    (255, 255, 255),
    (0, 0, 0)
]

# define a function to rotate a point around the origin
def rotate_point(point, angle):
    x, y, z = point
    radian = math.radians(angle)
    new_x = x * math.cos(radian) - z * math.sin(radian)
    new_z = z * math.cos(radian) + x * math.sin(radian)
    return (new_x, y, new_z)

# define a function to draw the cube
def draw_cube(vertices, edges, colors, angle):
    for edge, color in zip(edges, colors):
        start_vertex = vertices[edge[0]]
        end_vertex = vertices[edge[1]]
        start_rotated = rotate_point(start_vertex, angle)
        end_rotated = rotate_point(end_vertex, angle)
        start_projected = (int(start_rotated[0] + screen_width/2), int(start_rotated[1]
+ screen_height/2))
        end_projected = (int(end_rotated[0] + screen_width/2), int(end_rotated[1] +
screen_height/2))
        pygame.draw.line(screen, color, start_projected, end_projected, 2)

# set the initial angle
angle = 0

# game loop
while True:
    # handle events
```

```
        for event in
    pygame.event.get(): if
    event.type == pygame.QUIT:
        pygame.qui
        t() exit()

# fill the screen with white
color screen.fill((255, 255,
255))

# draw the cube
draw_cube(cube_vertices, cube_edges, cube_colors, angle)

# update the screen
pygame.display.update()

# increment the angle by 1
degree angle += 1

# rotate back to 0 degrees if greater than 45
degrees if angle > 45:
```

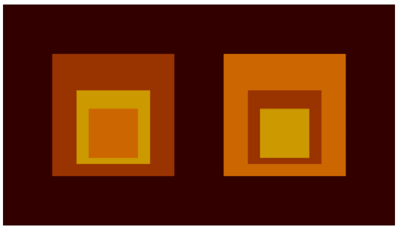
Output:

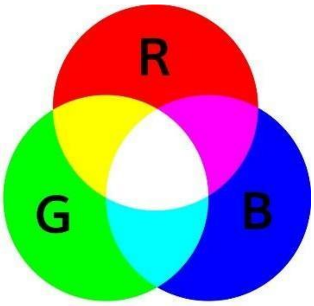


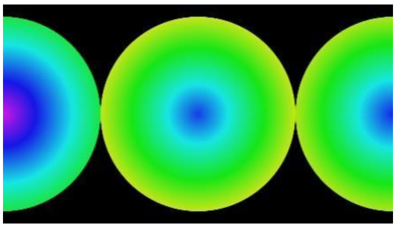
Realization of Color:

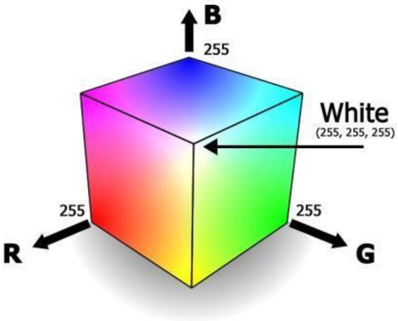
1. **Write programs to display any five(5) the following diagram using Color method in processing.**

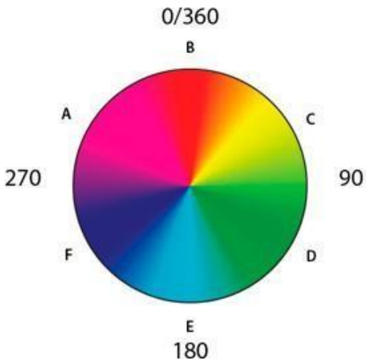
   [follow reference: https://processing.org/reference/color_.html]

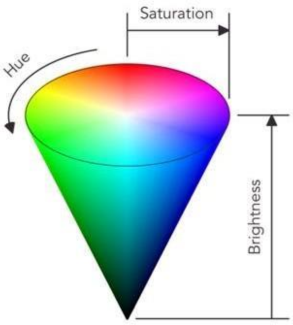
1 

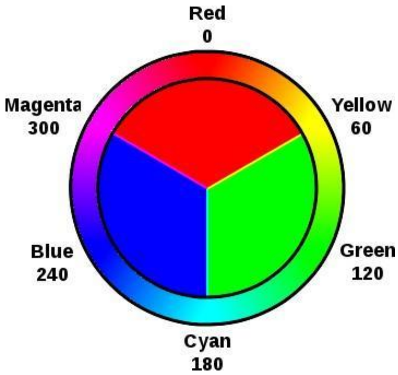
2 

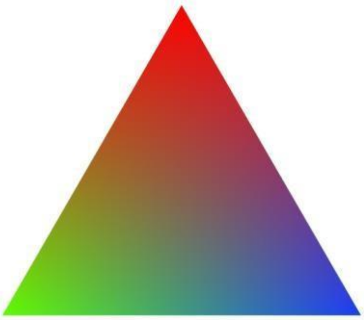
3 

4 

5 

6 

7 

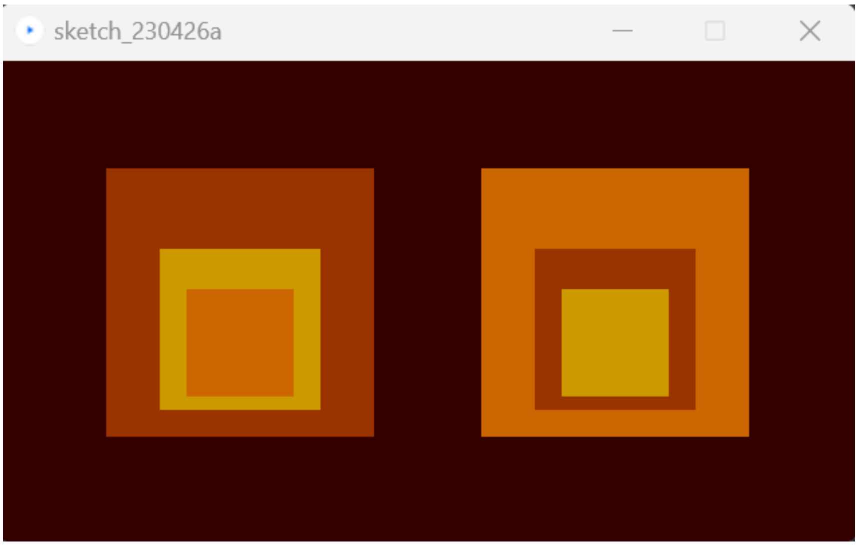
8 

Code:

```
/**
 * Color Variables (Homage to Albers).
 *
```

```
    This example creates variables for colors that may be referred to
    in the program by a name, rather than a number.
  */

size(640, 360);
noStroke();
background(51, 0,
0);

color inside = color(204, 102, 0);
color middle = color(204, 153, 0);
color outside = color(153, 51, 0);

// These statements are equivalent to the statements above.
// Programmers may use the format they prefer.
//color inside = #CC6600;
//color middle = #CC9900;
//color outside = #993300;

pushMatrix();
translate(80, 80);
fill(outside);
rect(0, 0, 200,
200);
fill(middle);
rect(40, 60, 120, 120);
fill(inside);
rect(60, 90, 80,
80);
popMatrix();

pushMatrix();
translate(360,
80);
fill(inside);
rect(0, 0, 200, 200);
fill(outside);
rect(40, 60, 120, 120);
```

Output:



Code:

```
/**
    Radial Gradient.
 *
    Draws a series of concentric circles to create a gradient
    from one color to another.
 */

int dim;

  void setup() {
  size(640, 360);
  dim = width/2;
  background(0);
  colorMode(HSB, 360, 100, 100);
  noStroke();
  ellipseMode(RADIUS);
  frameRate(1);
}

  void draw() {
  background(0);
      for (int x = 0; x <= width;
    x+=dim) { drawGradient(x,
    height/2);
  }
}

  void drawGradient(float x, float y)
  { int radius = dim/2;
  float h = random(0, 360);
```

```
    fill(h, 90, 90);
    ellipse(x, y, r,
    r); h = (h + 1) %
    360;
  }
}
```

Output: