Sahakar Maharshi Bhausaheb Santuji Thorat College Sangamner

DEPARTMENT OF COMPUTER SCIENCE

MATHEMATICS

Remark
Demonstrators
Signature
Date :- / /2023

Name :- Prem Vijay Vajare **Batch No. :- D** **Roll No:- 75** **Date:-** / /2023

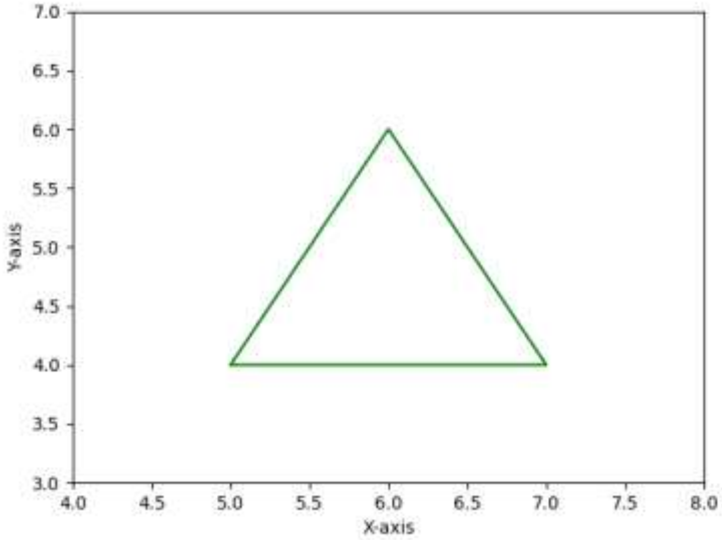
Title of the:- Practical 9 **Expt. No .** 9 **Class :-** S.Y.BCS

Q.1) Write n python program to Plot 2D X-axis and Y-axis black color and in the same diagram plot green triangle with vertices [5,4],[7,4],[6,6]

Syntax:

```
import matplotlib.pyplot as plt
# Define the vertices of the triangle
triangle_vertices = [[5, 4], [7, 4], [6, 6]]
# Extract the x and y coordinates of the triangle vertices
x = [vertex[0] for vertex in triangle_vertices]
y = [vertex[1] for vertex in triangle_vertices]
# Plot the X-axis and Y-axis in black color
plt.axhline(0, color='black')
plt.axvline(0, color='black')
# Plot the triangle with green color
plt.plot(x + [x[0]], y + [y[0]], 'g')
# Set the plot limits and labels
plt.xlim(4, 8)
plt.ylim(3, 7)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
# Show the plot
plt.show()
```

OUTPUT:



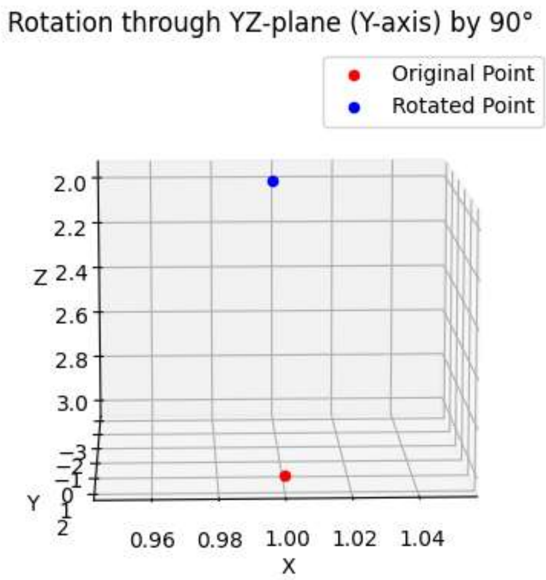
Q.2) Write n program in python to rotate the point. through YZ-plane in anticlockwise direction. (Rotation through Y-axis by an angle of 90°.)

Syntax:

```
import math
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Function to rotate a point (x, y, z) through YZ-plane by an angle of 90°
def rotate_yz_plane(point):
    x, y, z = point
    new_y = y * math.cos(math.radians(90)) - z * math.sin(math.radians(90))
    new_z = y * math.sin(math.radians(90)) + z * math.cos(math.radians(90))
    return [x, new_y, new_z]
# Point to rotate
point = [1, 2, 3]
# Call the rotation function
rotated_point = rotate_yz_plane(point)
# Original point coordinates
x_original, y_original, z_original = point
```

```
# Rotated point coordinates
x_rotated, y_rotated, z_rotated = rotated_point
# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot original point as a red dot
ax.scatter(x_original, y_original, z_original, color='red', label='Original Point')
# Plot rotated point as a blue dot
ax.scatter(x_rotated, y_rotated, z_rotated, color='blue', label='Rotated Point')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Rotation through YZ-plane (Y-axis) by 90°')
ax.legend()
plt.show()
```

OUTPUT:

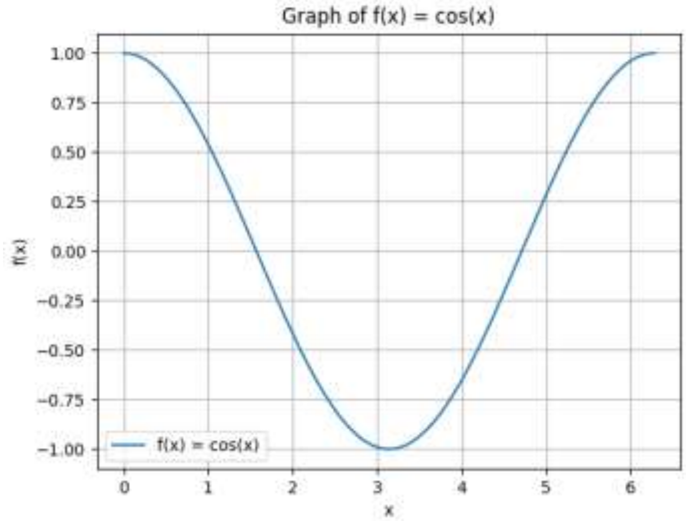


Q.3) Using Python plot the graph of function f(x) = cos(x) on the interval (0, 2*pi).

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Generate x values from 0 to 2*pi with a step of 0.01
x = np.arange(0, 2*np.pi, 0.01)
# Compute the corresponding y values for f(x) = cos(x)
y = np.cos(x)
# Create a plot
plt.plot(x, y, label='f(x) = cos(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Graph of f(x) = cos(x)')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:



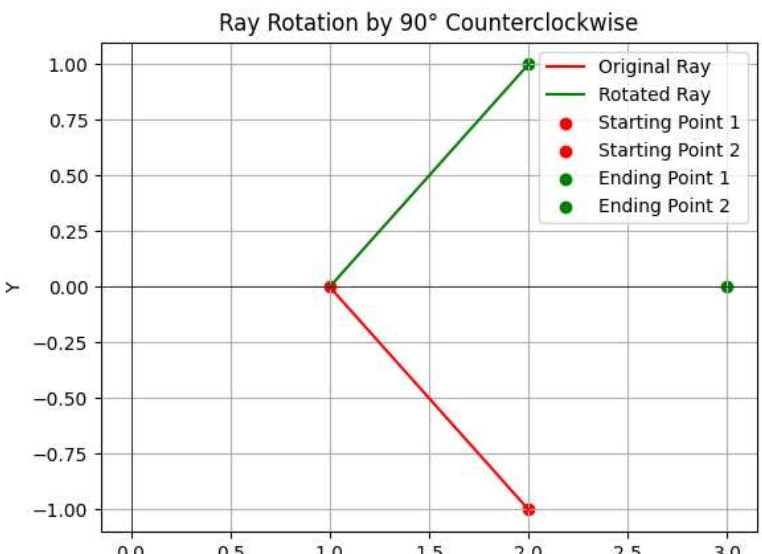
Q.4) Write a python program to rotate the ray by 90° having starting point (1,0) and (2,-1)

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the starting points of the ray
start_point_1 = np.array([1, 0])
```

```
start_point_2 = np.array([2, -1])
# Compute the direction vector of the ray
direction_vector = start_point_2 - start_point_1
# Perform the rotation by 90° counterclockwise
rotation_matrix = np.array([[0, -1], [1, 0]])
rotated_direction_vector = np.dot(rotation_matrix, direction_vector)
# Compute the ending point of the rotated ray
end_point_1 = start_point_1 + rotated_direction_vector
end_point_2 = start_point_2 + rotated_direction_vector
# Plot the original and rotated rays
plt.plot([start_point_1[0], start_point_2[0]], [start_point_1[1], start_point_2[1]], 'r', label='Original Ray')
plt.plot([start_point_1[0], end_point_1[0]], [start_point_1[1], end_point_1[1]], 'g', label='Rotated Ray')
plt.scatter(start_point_1[0], start_point_1[1], c='r', marker='o', label='Starting Point 1')
plt.scatter(start_point_2[0], start_point_2[1], c='r', marker='o', label='Starting Point 2')
plt.scatter(end_point_1[0], end_point_1[1], c='g', marker='o', label='Ending Point 1')
plt.scatter(end_point_2[0], end_point_2[1], c='g', marker='o', label='Ending Point 2')
plt.axhline(0, color='k', linewidth=0.5)
plt.axvline(0, color='k', linewidth=0.5)
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Ray Rotation by 90° Counterclockwise')
plt.grid(True)
plt.show()
```

Output:



Q.5) Using sympy declare the points A(0, 7), B(5, 2). Declare the line segment passing through them. Find the length and midpoint of the line segment passing through points A and B.

Syntax:

```
from sympy import Point, Line
# Declare the points A and B
A = Point(0, 7)
B = Point(5, 2)
# Declare the line passing through points A and B
line_AB = Line(A, B)
# Calculate the length of the line segment AB
length_AB = A.distance(B)
# Calculate the midpoint of the line segment AB
midpoint_AB = ((A[0] + B[0]) / 2, (A[1] + B[1]) / 2)
# Print the results
print("Point A: {}".format(A))
print("Point B: {}".format(B))
print("Line segment AB: {}".format(line_AB))
print("Length of line segment AB: {}".format(length_AB))
print("Midpoint of line segment AB: {}".format(midpoint_AB))
```

OUTPUT:

```
Point A: Point2D(0, 7)
Point B: Point2D(5, 2)
Line segment AB: Line2D(Point2D(0, 7), Point2D(5, 2))
Length of line segment AB: 5*sqrt(2)
Midpoint of line segment AB: (5/2, 9/2)
```

Q.6) Write a Python program to find the area and perimeter of the ABC, where A[0, 0] B[5, 0], C[3,3].

Synatx:

```
import numpy as np

# Define the vertices of the triangle

A = np.array([0, 0])
```

```
B = np.array([5, 0])
C = np.array([3, 3])
# Calculate the side lengths of the triangle
AB = np.linalg.norm(B - A)
BC = np.linalg.norm(C - B)
CA = np.linalg.norm(A - C)
# Calculate the semiperimeter
s = (AB + BC + CA) / 2
# Calculate the area using Heron's formula
area = np.sqrt(s * (s - AB) * (s - BC) * (s - CA))
# Calculate the perimeter
perimeter = AB + BC + CA
# Print the results
print("Triangle ABC:")
print("Side AB:", AB)
print("Side BC:", BC)
print("Side CA:", CA)
print("Area:", area)
print("Perimeter:", perimeter)
```

OUTPUT:

```
Triangle ABC:
Side AB: 5.0
Side BC: 3.605551275463989
Side CA: 4.242640687119285
Area: 7.5000000000000036
Perimeter: 12.848191962583275
```

Q.7) write a Python program to solve the following LPP

Max Z = 150x + 75y

Subjected to

4x + 6y <= 24

5x + 3y <= 15

x > 0 , y > 0

Syntax:

```
from pulp import *

# Create the LP problem as a maximization problem

problem = LpProblem("LPP", LpMaximize)

# Define the decision variables

x = LpVariable('x', lowBound=0, cat='Continuous')

y = LpVariable('y', lowBound=0, cat='Continuous')

# Define the objective function

problem += 150 * x + 75 * y, "Z"

# Define the constraints

problem += 4 * x + 6 * y <= 24, "Constraint1"

problem += 5 * x + 3 * y <= 15, "Constraint2"

# Solve the LP problem

problem.solve()

# Print the status of the solution

print("Status:", LpStatus[problem.status])

# Print the optimal values of x and y

print("Optimal x =", value(x))

print("Optimal y =", value(y))

# Print the optimal value of the objective function

print("Optimal Z =", value(problem.objective
```

OUTPUT:

Status: Optimal

Optimal x = 3.0

Optimal y = 0.0

Optimal Z = 450.0

Q.8) Write a python program to display the following LPP by using pulp module and simplex method. Find its optimal solution if exist.

Min Z = 4x + y + 3z + 5w
subject to
4x + 6y – 5z – 4w >= 20
-3x – 2y + 2z + w <= 10
-8x – 3y + 3z + 2w <= 20
x=>0, y=>0,z=> 0,w>=

Syntax:

```
from pulp import *
# Define the decision variables
x = LpVariable("x", lowBound=0)
y = LpVariable("y", lowBound=0)
z = LpVariable("z", lowBound=0)
w = LpVariable("w", lowBound=0)
# Define the objective function
objective = 4 * x + y + 3 * z + 5 * w
# Define the constraints
constraint1 = 4 * x + 6 * y - 5 * z - 4 * w >= 20
constraint2 = -3 * x - 2 * y + 2 * z + w <= 10
constraint3 = -8 * x - 3 * y + 3 * z + 2 * w <= 20
# Create the LP problem
problem = LpProblem("Linear_Programming_Problem", LpMinimize)
# Add the objective function and constraints to the problem
problem += objective
problem += constraint1
problem += constraint2
problem += constraint3
# Solve the LP problem
status = problem.solve()
# Check the status of the solution
if status == LpStatusOptimal:
```

```
    # Get the optimal values of the decision variables
    opt_x = value(x)
    opt_y = value(y)
    opt_z = value(z)
    opt_w = value(w)
    # Get the optimal value of the objective function
    opt_z = value(objective)
    # Print the optimal solution
    print("Optimal Solution:")
    print("x = {}".format(opt_x))
    print("y = {}".format(opt_y))
    print("z = {}".format(opt_z))
    print("w = {}".format(opt_w))
    print("Optimal value of the objective function: {}".format(opt_z))
else:
    print("No optimal solution found.")
```

OUTPUT:

```
Optimal Solution:
x = 0.0
y = 3.3333333
z = 3.3333333
w = 0.0
Optimal value of the objective function: 3.3333333
```

Q.9) Apply Python. Program in each of the following transformation on the point P[-2,4]

(I) Shearing in Y direction by 7 units.
(II)Scaling in X and Y direction by 7/2 and 7 unit respectively.
(III) Shearing in X and Y direction by 4 and 7 unit respectively.
(IV)Rotation about origin by an angle 60^{o}

Syntax:

```
import numpy as np
# Define the original point P
P = np.array([-2, 4])
# Transformation 1: Shearing in Y direction by 7 units
shearing_Y = np.array([[1, 0], [7, 1]])
P_transformed1 = np.dot(shearing_Y, P)
# Transformation 2: Scaling in X and Y direction by 7/2 and 7 units respectively
scaling_XY = np.array([[7/2, 0], [0, 7]])
P_transformed2 = np.dot(scaling_XY, P)
```

```
# Transformation 3: Shearing in X and Y direction by 4 and 7 units respectively
shearing_XY = np.array([[1, 4], [7, 1]])
P_transformed3 = np.dot(shearing_XY, P)
# Transformation 4: Rotation about origin by an angle of 60 degrees
angle = np.radians(60)
rotation = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
P_transformed4 = np.dot(rotation, P)
# Print the transformed points
print("Original Point P: {}".format(P))
print("Transformation 1: Shearing in Y direction by 7 units: {}".format(P_transformed1))
print("Transformation 2: Scaling in X and Y direction by 7/2 and 7 units respectively: {}".format(P_transformed2))
print("Transformation 3: Shearing in X and Y direction by 4 and 7 units respectively: {}".format(P_transformed3))
print("Transformation 4: Rotation about origin by an angle of 60 degrees: {}".format(P_transformed4))
```

OUTPUT:

```
Original Point P: [-2  4]
Transformation 1: Shearing in Y direction by 7 units: [ -2 -10]
Transformation 2: Scaling in X and Y direction by 7/2 and 7 units respectively: [-7. 28.]
Transformation 3: Shearing in X and Y direction by 4 and 7 units respectively: [ 14 -10]
Transformation 4: Rotation about origin by an angle of 60 degrees: [-4.46410162 0.26794919]
```

Q.10) Find the combined transformation of the line segment between the point A[5,3] & B[1, 4] by using Python program for the following sequence of transformation:-

(I) Rotate about origin through an angle pi/3.
(II) Uniform scaling by -.5 units
(III) scaling in Y – axis by 5 units
(IV) Shearing in X and Y direction by 3 and 4 nits respectively.

Syntax:

```
import numpy as np
# Define the original points A and B
```

```
A = np.array([5, 3])
B = np.array([1, 4])
# Transformation 1: Rotate about origin through an angle of pi/3
angle = np.pi / 3
rotation = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
A_transformed1 = np.dot(rotation, A)
B_transformed1 = np.dot(rotation, B)
# Transformation 2: Uniform scaling by -0.5 units
scaling_uniform = np.array([[-0.5, 0], [0, -0.5]])
A_transformed2 = np.dot(scaling_uniform, A_transformed1)
B_transformed2 = np.dot(scaling_uniform, B_transformed1)
# Transformation 3: Scaling in Y-axis by 5 units
scaling_Y = np.array([[1, 0], [0, 5]])
A_transformed3 = np.dot(scaling_Y, A_transformed2)
B_transformed3 = np.dot(scaling_Y, B_transformed2)
# Transformation 4: Shearing in X and Y direction by 3 and 4 units respectively
shearing_XY = np.array([[1, 3], [4, 1]])
A_transformed4 = np.dot(shearing_XY, A_transformed3)
B_transformed4 = np.dot(shearing_XY, B_transformed3)
# Print the transformed points
print("Original Point A: {}".format(A))
print("Original Point B: {}".format(B))
print("Transformation 1: Rotate about origin through an angle of pi/3")
print("A_transformed1: {}".format(A_transformed1))
print("B_transformed1: {}".format(B_transformed1))
print("Transformation 2: Uniform scaling by -0.5 units")
print("A_transformed2: {}".format(A_transformed2))
print("B_transformed2: {}".format(B_transformed2))
print("Transformation 3: Scaling in Y-axis by 5 units")
print("A_transformed3: {}".format(A_transformed3))
print("B_transformed3: {}".format(B_transformed3))
print("Transformation 4: Shearing in X and Y direction by 3 and 4 units respectively")
print("A_transformed4: {}".format(A_transformed4))
print("B_transformed4: {}".format(B_transformed4))
```

OUTPUT:

```
Original Point P: [-2  4]
Transformation 1: Shearing in Y direction by 7 units: [ -2 -10]
```

Transformation 2: Scaling in X and Y direction by 7/2 and 7 units respectively: [-7. 28.]
Transformation 3: Shearing in X and Y direction by 4 and 7 units respectively: [14 -10]
Transformation 4: Rotation about origin by an angle of 60 degrees: [-4.46410162 0.26794919]
PS E:\Python 2nd Sem Practical> python -u "e:\Python 2nd Sem Practical\tempCodeRunnerFile.py"
Original Point A: [5 3]
Original Point B: [1 4]
Transformation 1: Rotate about origin through an angle of pi/3
A_transformed1: [-0.09807621 5.83012702]
B_transformed1: [-2.96410162 2.8660254]
Transformation 2: Uniform scaling by -0.5 units
A_transformed2: [0.04903811 -2.91506351]
B_transformed2: [1.48205081 -1.4330127]
Transformation 3: Scaling in Y-axis by 5 units
A_transformed3: [0.04903811 -14.57531755]
B_transformed3: [1.48205081 -7.16506351]
Transformation 4: Shearing in X and Y direction by 3 and 4 units respectively
A_transformed4: [-43.67691454 -14.37916512]
B_transformed4: [-20.01313972 -1.23686028]