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:- 04** **Date:-** / /2023

Title of the:- Practical 1 **Expt. No .** 1 **Class :-** S.Y.BCS

Q.1)Write a Python program to plot 2D graph of the functions $f(x) = x^2$ and $g(x) = x^3$ in [-1, 1]

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
def f(x):
    return x**2
def g(x):
    return x**3
# Generate x values in the range [-1, 1]
x = np.linspace(-1, 1, 100)
# Calculate y values for f(x) and g(x)
y_f = f(x)
y_g = g(x)
# Create a figure and axes
fig, ax = plt.subplots()
# Plot f(x) and g(x) on the same graph
ax.plot(x, y_f, label='f(x) = x^2')
ax.plot(x, y_g, label='g(x) = x^3')
# Add labels and legend
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend()
```

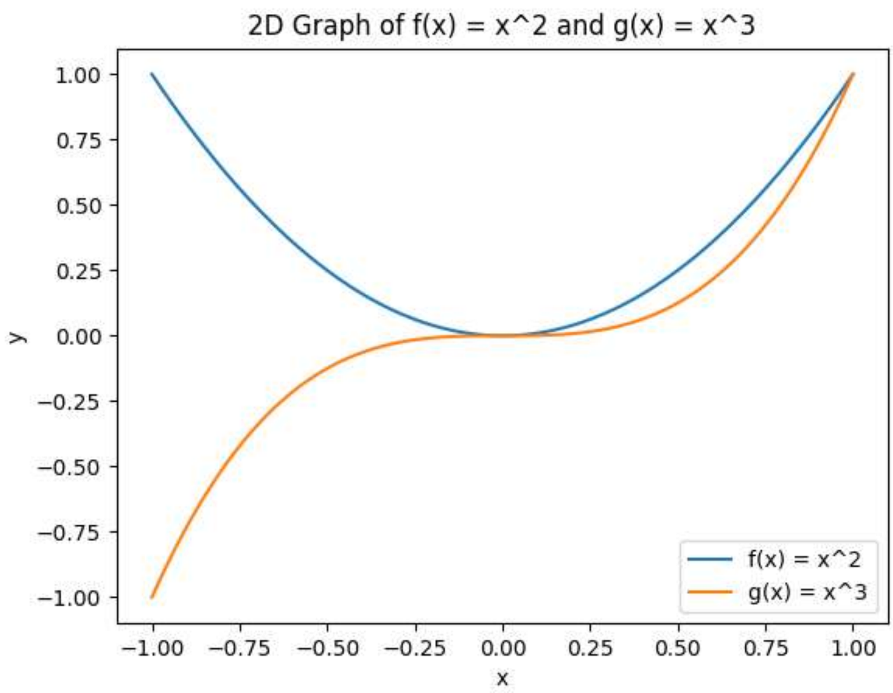
Set title

ax.set_title('2D Graph of f(x) = x^2 and g(x) = x^3')

Show the plot

plt.show()

OUTPUT:



Q.2) Write a Python program to plot 3D graph of the function f(x) = e**x**3 in [-5, 5] with green dashed points line with upward pointing triangle.

Syntax:

import numpy as np

import matplotlib.pyplot as plt

Generate x values

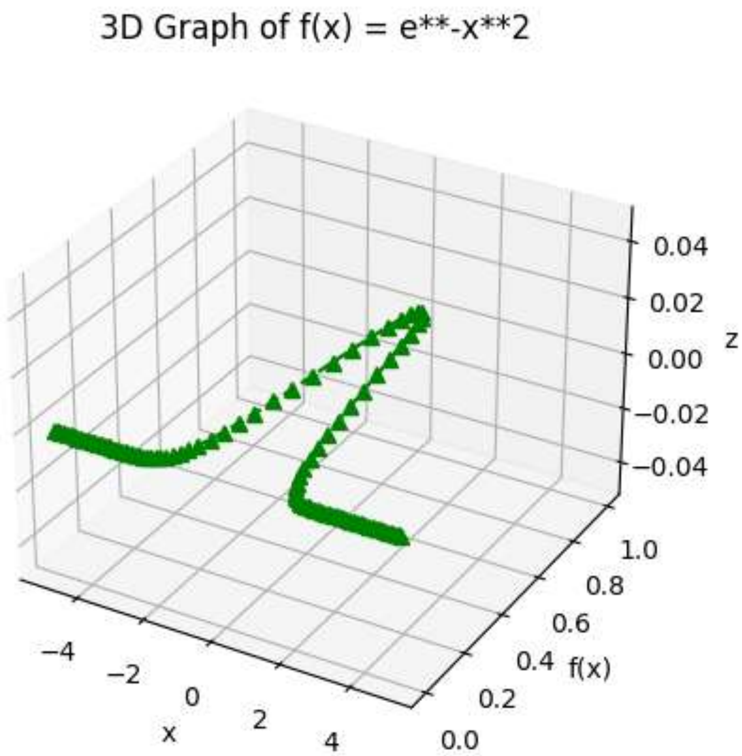
x = np.linspace(-5, 5, 100)

Compute y values using the given function

y = np.exp(-x**2)

```
# Create 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the points with green dashed line and upward-pointing triangles
ax.plot(x, y, np.zeros_like(x), linestyle='dashed', color='green', marker='^')
# Set labels for axes
ax.set_xlabel('x')
ax.set_ylabel('f(x)')
ax.set_zlabel('z')
# Set title for the plot
ax.set_title('3D Graph of f(x) = e**-x**2')
# Show the plot
plt.show()
```

OUTPUT:



Q.3) Using python, represent the following information using a bar graph (in green color)

Item	Clothing	Food	Rent	Petrol	Misc
Expenditure in Rs	60	4000	2000	1500	700

Syntax:

import matplotlib.pyplot as plt

left = [1,2,3,4,5]

height = [600,4000,200,1500,]

tick_label=[‘clothing’,’food’,’rent’,’petrol’,’Misc’]

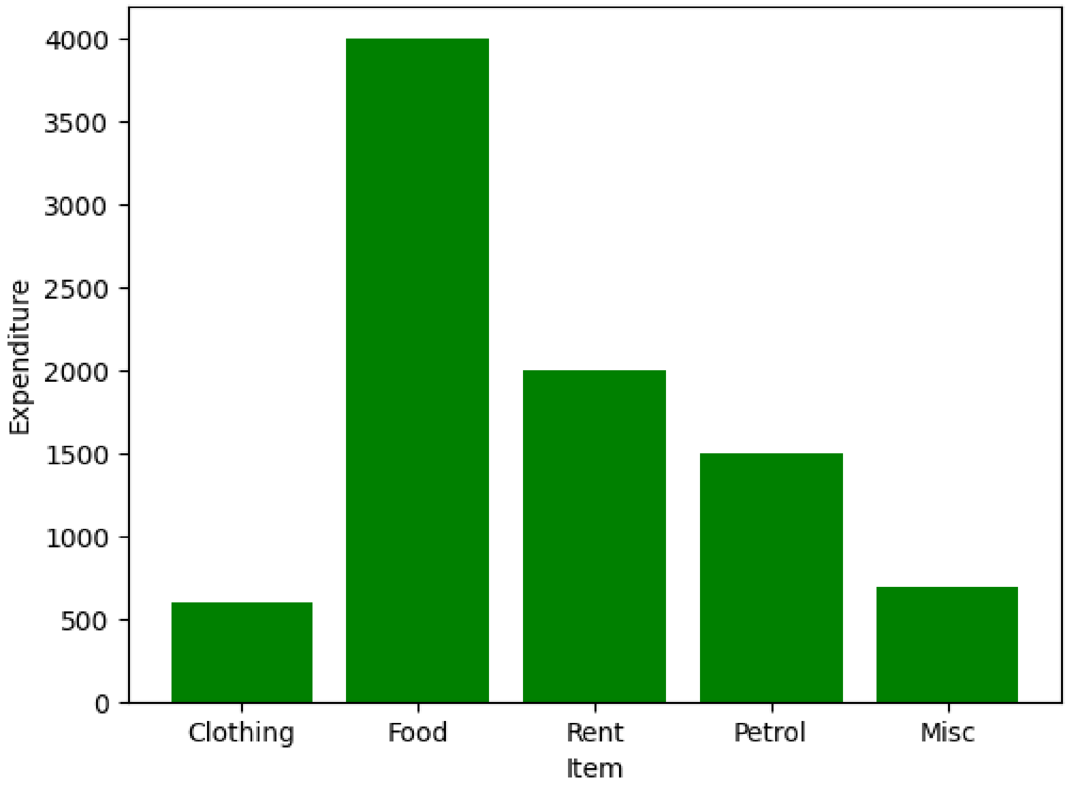
plt.bar (left,height,tick_label = tick_label,width = 0.8 ,color = ['green','green'])

plt.xlabel('Item')

plt.ylabel('Expenditure')

plt. show()

OUTPUT:



Q.4) write a Python program to reflect the line segment joining the points A[5, 3] and B[l, 4] through the line y = x + 1.

Syntax:

```
import numpy as np
# Define the points A and B
A = np.array([5, 3])
B = np.array([1, 4])
# Define the equation of the reflecting line
def reflect(line, point):
    m = line[0]
    c = line[1]
    x, y = point
    x_reflect = (2 * m * (y - c) + x * (m ** 2 - 1)) / (m ** 2 + 1)
    y_reflect = (2 * m * x + y * (1 - m ** 2) + 2 * c) / (m ** 2 + 1)
    return np.array([x_reflect, y_reflect])
# Define the equation of the reflecting line y = x + 1
line = np.array([1, -1])
# Reflect points A and B through the reflecting line
A_reflected = reflect(line, A)
B_reflected = reflect(line, B)
# Print the reflected points
print("Reflected Point A':", A_reflected)
print("Reflected Point B':", B_reflected)
```

Output:

```
Reflected Point A': [4. 4.]
Reflected Point B': [5. 0.]
```

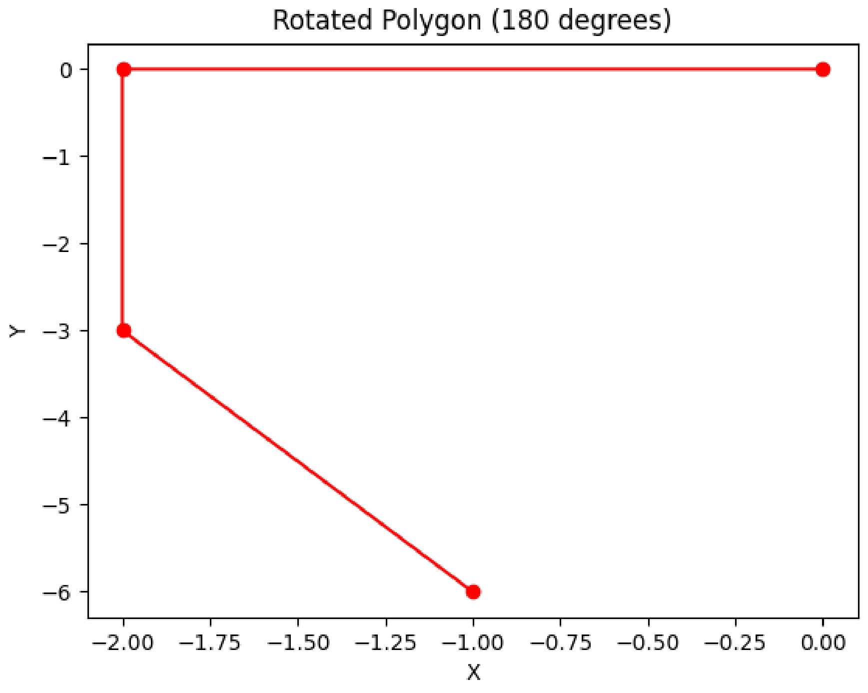
Q.5) Write a Python program to draw a polygon with vertices (0, 0), (2, 0), (2, 3) and (1, 6) and rotate it by 180° .

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the vertices of the polygon
vertices = np.array([[0, 0], [2, 0], [2, 3], [1, 6]])
```

```
# Plot the original polygon
plt.figure()
plt.plot(vertices[:, 0], vertices[:, 1], 'bo-')
plt.title('Original Polygon')
plt.xlabel('X')
plt.ylabel('Y')
# Define the rotation matrix for 180 degrees
theta = np.pi  # 180 degrees
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                            [np.sin(theta), np.cos(theta)]])
# Apply rotation to the vertices
vertices_rotated = np.dot(vertices, rotation_matrix)
# Plot the rotated polygon
plt.figure()
plt.plot(vertices_rotated[:, 0], vertices_rotated[:, 1], 'ro-')
plt.title('Rotated Polygon (180 degrees)')
plt.xlabel('X')
plt.ylabel('Y')
# Show the plots
plt.show()
```

OUTPUT:



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

Transformed Point A: [38. 10.]

Transformed Point B: [35. 8.]

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 = x+y
subject to
x => 6
y => 6
x + y <= 11
x=>0, y=>0

Syntax:

```
from pulp import *
# Create the LP problem as a minimization problem
problem = LpProblem("LPP", LpMinimize)
# Define the decision variables
x = LpVariable('x', lowBound=0, cat='Continuous')
y = LpVariable('y', lowBound=0, cat='Continuous')
# Define the objective function
problem += x + y, "Z"
# Define the constraints
problem += x >= 6, "Constraint1"
problem += y >= 6, "Constraint2"
problem += x + y <= 11, "Constraint3"
# Solve the LP problem using the simplex method
```

problem.solve(PULP_CBC_CMD(msg=False))
# Print the status of the solution
print("Status:", LpStatus[problem.status])
# If the problem has an optimal solution
if problem.status == LpStatusOptimal:
    # 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
Status: Infeasible

Q.9) Apply Python. Program in each of the following transformation on the point P[3,-1]
(I)Refection through X-axis.
(II)Scaling in X-co-ordinate by factor 2.
(III) Scaling in Y-co-ordinate by factor 1.5.
(IV) Reflection through the line y = x
Syntax:
Original point
x = 3
y = -1
print("Original point: ({}, {})".format(x, y))
# Transformation 1: Reflection through X-axis
x_reflected = x
y_reflected = -y
print("After reflection through X-axis: ({}, {})".format(x_reflected, y_reflected))
# Transformation 2: Scaling in X-coordinate by factor 2
x_scaled = x * 2
y_scaled = y
print("After scaling in X-coordinate by factor 2: ({}, {})".format(x_scaled, y_scaled))
# Transformation 3: Scaling in Y-coordinate by factor 1.5
x_scaled = x
y_scaled = y * 1.5
print("After scaling in Y-coordinate by factor 1.5: ({}, {})".format(x_scaled, y_scaled))
# Transformation 4: Reflection through the line y = x
x_reflected = y

```
y_reflected = x
print("After reflection through the line y = x: ({}, {})".format(x_reflected, y_reflected))
```

OUTPUT:
Original point: (3, -1)
After reflection through X-axis: (3, 1)
After scaling in X-coordinate by factor 2: (6, -1)
After scaling in Y-coordinate by factor 1.5: (3, -1.5)
After reflection through the line y = x: (-1, 3)

Q.10) Find the combined transformation of the line segment between the point A[5, -2] & B[4, 3] by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle pi.
(II) Scaling in X-Coordinate by 2 units.
(III) Reflection trough he line y = x
(IV) Shearing in X – Direction by 4 unit

Syntax:

```
import numpy as np
# Input points A and B
A = np.array([5, -2])
B = np.array([4, 3])
# Transformation 1: Rotation about origin through an angle pi
def rotation(pi, point):
    rotation_matrix = np.array([[np.cos(pi), -np.sin(pi)],
                               [np.sin(pi), np.cos(pi)]])
    return np.dot(rotation_matrix, point)
A = rotation(np.pi, A)
B = rotation(np.pi, B)
# Transformation 2: Scaling in X-coordinate by 2 units
def scaling_x(sx, point):
    scaling_matrix = np.array([[sx, 0],
                              [0, 1]])
    return np.dot(scaling_matrix, point)
A = scaling_x(2, A)
B = scaling_x(2, B)
# Transformation 3: Reflection through the line y = -x
def reflection(line, point):
```

```
    reflection_matrix = np.array([[-line[0]**2 + line[1]**2, 2*line[0]*line[1]],
                                  [2*line[0]*line[1], -line[0]**2 + line[1]**2]]) / (line[0]**2 +
line[1]**2)
    return np.dot(reflection_matrix, point)
A = reflection(np.array([1, -1]), A)
B = reflection(np.array([1, -1]), B)
# Transformation 4: Shearing in X direction by 4 units
def shearing_x(shx, point):
    shearing_matrix = np.array([[1, shx],
                                [0, 1]])
    return np.dot(shearing_matrix, point)
A = shearing_x(4, A)
B = shearing_x(4, B)
# Print the transformed points A and B
print("Transformed Point A:", A)
print("Transformed Point B:", B)
```

OUTPUT:

```
Status: Infeasible
Transformed Point A: [38. 10.]
Transformed Point B: [35.  8.]
```