S.Y.BSc Computer

Prof.Mamta Solanki
Prof.Vishal Khare

Python Programming Practical Workbook

(2 credits)

Name of Student:	
Roll Number:	

SSR College of Arts Commerce and Science

Silvassa

S.Y.BSc Computer Science

Python Programming Practical Workbook

Certificate

This is to certify that,

Mr./Ms. ______________________________ has satisfactorily completed _______ out of ________ assignments of Mathematics Practical in class S.Y.BSc Computer Semester I during the academic year _______________________.

Signature of instructor_________________ Date:__________

Signature of instructor_________________ Date:__________

INDEX

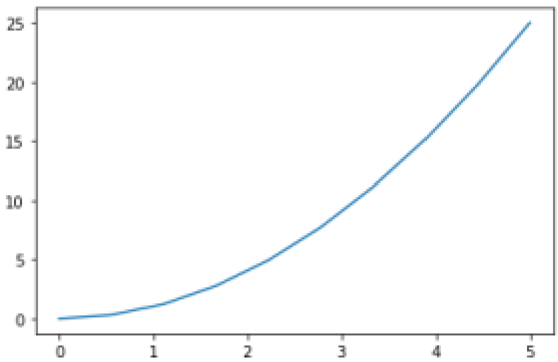
Sr. No.	Practical	Remark
1	Graph Plotting (Unit 1 – 1.1 – 1.3)	
2	Graph Plotting (Unit 1 – 1.4 – 1.7)	
3	Application to Computational Geometry (Unit 2 – 2.1)	
4	Application to Computational Geometry (Unit 2 – 2.2)	
5	Application to Computational Geometry (Unit 2 – 2.3)	
6	Study of Graphical aspects of Two-dimensional transformation matrix using matplotlib	
7	Study of Graphical aspects of Three-dimensional transformation matrix using matplotlib	
8	Study of Graphical aspects of Three-dimensional transformation matrix using matplotlib	
9	Study of effect of concatenation of Two dimensional and Three dimensional transformations	
10	Generation of Bezier curve using given control points	
11	Study of Operational Research in Python (Unit 3.1)	
12	Study of Operational Research in Python (Unit 3.2)	

PRACTICAL 1: GRAPH PLOTTING

#1. Plot the graph of f(x)=x2 in [0,5].

```
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0,5,10)
y=x**2
plot(x,y)
show()
```

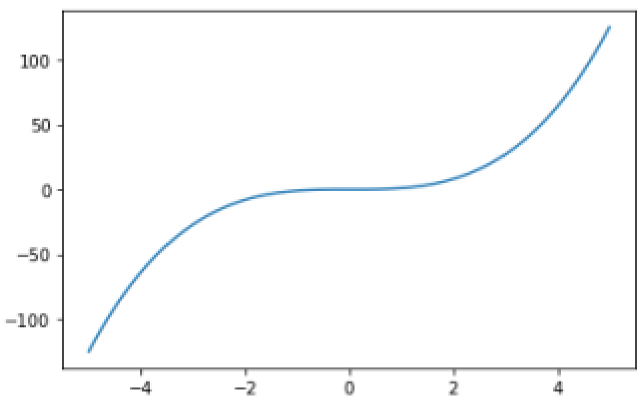
Output :



#2. Plot the graph of f(x)=x3 in [-5,5].

```
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-5,5,100)
y=x**3
plot(x,y)
show()
```

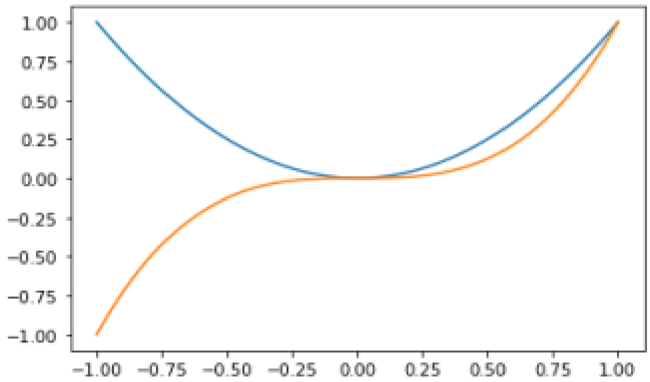
OUTPUT :



#3. Plot the graph of f(x)=x2 and g(x)=x3 in [-1,1].

```
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-1,1,100)
f=x**2 g=x**3
plot(x,f)
plot(x,g)
show()
```

OUTPUT:

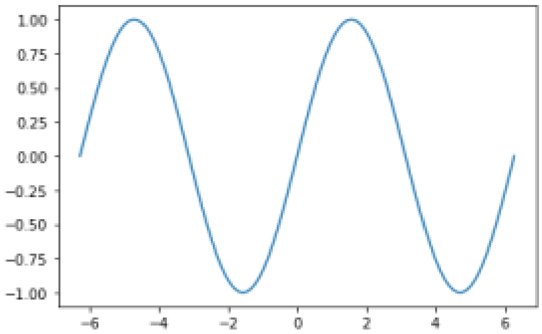


PRACTICAL 2: GRAPH PLOTTING

#1.Plot the graph of f(x)= sinx in [-2π,2π].

```
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-2*pi,2*pi,100)
f=np.sin(x)
plot(x,f)
show()
```

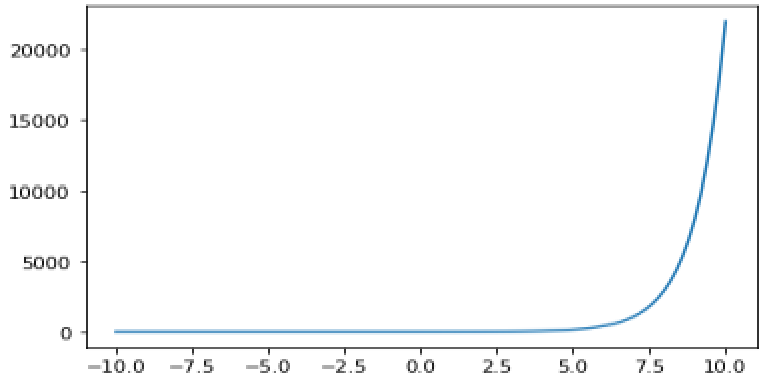
OUTPUT :



#2. Plot the graph of f(x) =ex in [-10,10].

```
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-10,10,100)
f=np.exp(x)
plot(x,f)
show()
```

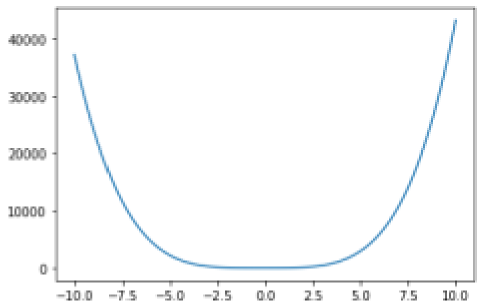
OUTPUT :



#3. Plot the graph of f(x)= 1+x+2x2+3x3+4x4 in [-10,10].

```
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-10,10,100)
f=-1+x+2*x**2+3*x**3+4*x**4
plot(x,f)
show()
```

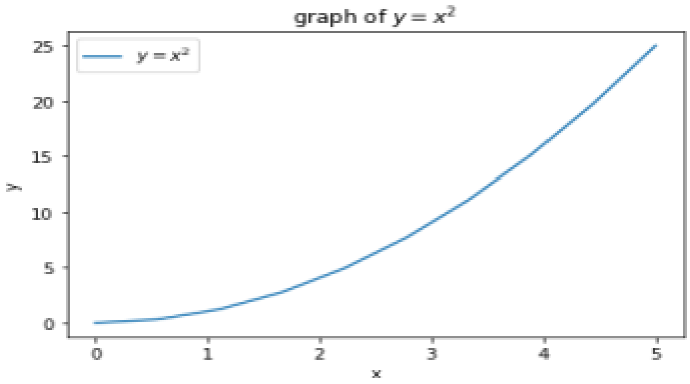
OUTPUT :



#4. Plot the graph of f(x) = x2 in [0,5].

```
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0,5,10)
y=x**2
plot(x,y,label="$y=x^2$")
xlabel('x')
ylabel('y')
title('graph of $y=x^2$')
legend()
show()
```

OUTPUT:



PRACTICAL 3: Application to Computational Geometry-I

#1. find the distance between points x and y,y and w,x and z,if x=[1,-1],y=[-2,4],z=[-1,-1] and w=[3,5]

```
import sympy as sp
x=sp.Point(1,-1)
y=sp.Point(-2,4)
z=sp.Point(-1,-1)
w=sp.Point(3,5)
print(x.distance(y))
print(y.distance(w))
print(x.distance(z))
```

OUTPUT:

```
sqrt(34)
sqrt(26)
2
```

#2.Reflect the given points through respective lines.

#(a) (−3, 6) , x + 2y = 0

#(b) (2, −6) , 2x + 3y = −1

#(c) (5, −2) , x − y = 5

#(d) (6.3, 3.6) , x − 4y = 1

#(e) (−5, 8) , 4x + 3y = 11

```
import sympy as sp
x,y=sp.symbols('x y')
P=sp.Point(-3,6)
print("a.",P.reflect(sp.Line(x+2*y)))
P=sp.Point(2,-6)
print("b.",P.reflect(sp.Line(2*x+3*y+1)))
P=sp.Point(5,-2)
print("c.",P.reflect(sp.Line(x-y-5)))
```

```
P=sp.Point(6.3,3.6)
print("d.",P.reflect(sp.Line(x-4*y-1)))
P=sp.Point(-5,8)
print("e.",P.reflect(sp.Line(4*x+3*y-11)))
```

OUTPUT:

```
a. Point2D(-33/5, -6/5)
b. Point2D(6, 0)
c. Point2D(3, 0)
d. Point2D(1253/170, -58/85)
e. Point2D(-69/25, 242/25)
```

#3. Reflect the point P(-3,6) through the line 3x/sqrt2 − 2y +4/3 = 0.

```
import sympy as sp
x,y=sp.symbols('x y')
P=sp.Point(-3,6)
print(P.reflect(sp.Line(3*x/sp.sqrt(2)-2*y+4/3)))
```

OUTPUT:

```
Point2D(3/17 + 64*sqrt(2)/17, 50/51 - 36*sqrt(2)/17)
```

Practical 4: Application to Computational Geometry-II

#1. Apply each of the following transformations on the point P = [2, −5].
#(a) Reflection through X-axis.
#(b) Scaling in X-coordinate by factor 4.
#(c) Scaling in Y-coordinate by factor 5.
#(d) Reflection through the line y = −2x.
#(e) Shearing in Y direction by 2 units.
#(f) Scaling in X and Y direction by 4/5 and 7 units respectively.
#(g) Shearing in both X and Y direction by −3 and 1 units respectively.
#(h) Rotation about origin by an angle 45 degrees.

```
import sympy as sp
P=sp.Point(2,-5)
x,y=sp.symbols('x y')
print("a.",P.transform(sp.Matrix([[1,0,0],[0,-1,0],[0,0,1]])))
print("b.",P.transform(sp.Matrix([[4,0,0],[0,1,0],[0,0,1]])))
print("c.",P.transform(sp.Matrix([[1,0,0],[0,5,0],[0,0,1]])))
print("d.",P.reflect(sp.Line(y+2*x)))
print("e.",P.transform(sp.Matrix([[1,2,0],[0,1,0],[0,0,1]])))
print("f.",P.transform(sp.Matrix([[4/5,0,0],[0,7,0],[0,0,1]])))
print("g.",P.transform(sp.Matrix([[1,1,0],[-3,1,0],[0,0,1]])))
print("h.",P.rotate(sp.pi/4))
```

```
OUTPUT:
a. Point2D(2, 5)
b. Point2D(8, -5)
c. Point2D(2, -25)
d. Point2D(14/5, -23/5)
e. Point2D(2, -1)
f. Point2D(8/5, -35)
g. Point2D(17, -3)
h. Point2D(7*sqrt(2)/2, -3*sqrt(2)/2)
```

#2. Apply each of the following transformations on the point P = [−2, 4].
#(a) Scaling in X and Y direction by 7/2 and 4 units respectively.
#(b) Shearing in both X and Y direction by 4 and 7 units respectively.
#(c) Reflection through the line y = 2x + 3.
#(d) Shearing in Y direction by 7 units.
#(e) Rotation about origin by an angle 48 degrees.
#(f) Reflection through line 3x + 4y = 5.
#(g) Scaling in X-coordinate by factor 6.
#(h) Scaling in Y-coordinate by factor 4.

```
import sympy as sp
import math as mt
P=sp.Point(-2,4)
x,y=sp.symbols('x y')
print("a.",P.transform(sp.Matrix([[7/2,0,0],[0,4,0],[0,0,1]])))
print("b.",P.transform(sp.Matrix([[1,7,0],[4,1,0],[0,0,1]])))
print("c.",P.reflect(sp.Line(-2*x+y-3)))
print("d.",P.transform(sp.Matrix([[1,7,0],[0,1,0],[0,0,1]])))
print("e.",P.rotate(angle))
print("f.",P.reflect(sp.Line(3*x+4*y-5)))
print("g.",P.transform(sp.Matrix([[6,0,0],[0,1,0],[0,0,1]])))
print("h.",P.transform(sp.Matrix([[1,0,0],[0,4,0],[0,0,1]])))
angle=mt.radians(48)
```

OUTPUT:

```
a. Point2D(-7, 16)
b. Point2D(14, -10)
c. Point2D(2, 2)
d. Point2D(-2, -10)
e. Point2D(-431084051462729/100000000000000, 929869355063/781250000000)
f. Point2D(-16/5, 12/5)
g. Point2D(-12, 4)
h. Point2D(-2, 16)
```

#3. Apply each of the following transformations on the point P = [−4, 1]
#(a) Scaling in X-coordinate by factor 3.
#(b) Scaling in Y-coordinate by factor 4.
#(c) Reflection through the line y = 2x − 3.
#(d) Shearing in Y direction by 7 units.
#(e) Scaling in X and Y direction by 5/2 and 4 units respectively.
#(f) Rotation about origin by an angle 35 degrees.
#(g) Reflection through line x + 4y = 0.

```
import sympy as sp
import math as mt
P=sp.Point(-4,1)
x,y=sp.symbols('x y')
print("a.",P.transform(sp.Matrix([[3,0,0],[0,1,0],[0,0,1]])))
print("b.",P.transform(sp.Matrix([[1,0,0],[0,4,0],[0,0,1]])))
print("c.",P.reflect(sp.Line(-2*x+y+3)))
print("d.",P.transform(sp.Matrix([[1,7,0],[0,1,0],[0,0,1]])))
print("g.",P.reflect(sp.Line(x+4*y)))
print("e.",P.transform(sp.Matrix([[5/2,0,0],[0,4,0],[0,0,1]])))
angle=mt.radians(35)
print("f.",P.rotate(angle))
```

OUTPUT:
```
a. Point2D(-12, 1)
b. Point2D(-4, 4)
c. Point2D(28/5, -19/5)
d. Point2D(-4, -27)
e. Point2D(-10, 4)
f. Point2D(-385018461350701/100000000000000, -
147515370111519/100000000000000)
g. Point2D(-4, 1)
```

Practical 5: Application to Computational Geometry-III

#1.Rotate the line passing through points A[5 1] and B[2 5] about origin through an angle 270 degrees

```
import sympy as sp
l=sp.Line((5,1),(2,5))
print(l.rotate(3*sp.pi/4))
```

OUTPUT:

Line2D(Point2D(-3*sqrt(2), 2*sqrt(2)), Point2D(-7*sqrt(2)/2, -3*sqrt(2)/2))

#2. Rotate the line passing through points A[0 −1] and B[2 −5] about origin through an angle 80 degrees

```
import sympy as sp
import math as mt
l=sp.Line((0,-1),(2,-5))
angle=mt.radians(80)
print(l.rotate(angle))
```

OUTPUT:

Line2D(Point2D(61550484563263/62500000000000, -17364817766693/100000000000000),
Point2D(52713351203949/10000000000000, 6883591360561/6250000000000))

#3. If the line segment joining the points A[−2 5], B[−4 3] is transformed to the line segment A∗B∗ by the transformation matrix, [T] =
#[2 −2],[3 −6] then find the midpoint and length of A∗B∗

```
import sympy as sp
import math as mt
A=sp.Point(-2,5)
B=sp.Point(-4,3)
A1=A.transform(sp.Matrix([[2,-2,0],[3,-6,0],[0,0,1]]))
B1=B.transform(sp.Matrix([[2,-2,0],[3,-6,0],[0,0,1]]))
L=sp.Segment(A1,B1)
print(L)
print(L.midpoint)
```

OUTPUT:

$Point2D$(6,−18)

```
#4. Reflect the line segment joining the points A[5, 2] and B[−3, 4]
#through the line y = 2x − 1.
import sympy as sp
A=sp.Point(5,2)
B=sp.Point(-3,4)
S=sp.Segment(A,B)
x,y=sp.symbols('x,y')
L=sp.Line(-2*x+y+1)
S.reflect(L)
```

OUTPUT:
$Segment2D(Point2D(-3/5,24/5),Point2D(29/5,-2/5))$

```
#5. Rotate the line by 75 degrees having two points (0, 0) and (0, 1). Also
#find its equation after applying rotation.
import sympy as sp
import math as mt
l=sp.Line((0,0),(0,1))
angle=mt.radians(75)
print(l.rotate(angle))
l1=l.rotate(angle)
print(l1.equation())
```

OUTPUT:

```
Line2D(Point2D(0, 0), Point2D(-241481456572267/250000000000000,
258819045102521/1000000000000000))
-258819045102521*x/1000000000000000 -
241481456572267*y/250000000000000
```

```
#6. Rotate the segment by 180 degrees having end points (1, 0) and (2, −1)
import sympy as sp
l=sp.Line((1,0),(2,-1))
print(l.rotate(sp.pi))
```

OUTPUT:

```
Line2D(Point2D(-1, 0), Point2D(-2, 1))
```

```
#7. Rotate the ray by 90 degrees having starting point (0, 0) in the direction of (4, 4).
import sympy as sp
import math as mt
R=sp.Ray(sp.Point(0,0),sp.Point(4,4))
print(R.rotate(mt.pi/2))
```

OUTPUT:

```
Ray2D(Point2D(0, 0), Point2D(-4, 4))
```

Practical 6: Study of Graphical aspects of Two-dimensional transformation matrix using matplotlib

```
#1. Reflect the line 4x + 3y = 5 through line x + y = 0 and find the
#equation of reflected line.
import sympy as sp
l=sp.Line(4*x+3*y-5)
l1=sp.Line(x+y)
reflected_line=l.reflect(l1)
eqn=reflected_line.equation()
print(eqn)
```

OUTPUT:
x+4*y/3+5/3

```
#2. Reflect the segment having two endpoints (2, 3),(4, 6) through line
#7x + 6y = 3.
import sympy as sp
A=sp.Point(2,3)
B=sp.Point(4,6)
S=sp.Segment(A,B)
x,y=sp.symbols('x,y')
L=sp.Line(7*x+6*y-3)
print(S.reflect(L))
```

OUTPUT:
Segment2D(Point2D(-236/85, -93/85), Point2D(-514/85, -222/85))

```
#3. Reflect the line segment having starting point (0, 0) in the direction of
#(2, 4) through line x − 2y = 3.
import numpy as np
A=sp.Point(0,0)
B=sp.Point(2,4)
S=sp.Ray(A,B)
x,y=sp.symbols('x,y')
L=sp.Line(x-2*y-3)
print(S.reflect(L))
```

OUTPUT:
$Ray2D(Point2D$(6/5,−12/5),$Point2D$(28/5,−16/5))

Practical 7: Study of Graphical aspects of Three-dimensional transformation matrix using matplotlib

#1. If the line with points A[2 1], B[4 −1] is transformed by the transformation matrix, [T] = [1 2],[2 1] then find the equation of transformed line.

```
import sympy as sp
import math as mt
A=sp.Point(2,1)
B=sp.Point(4,-1)
print(A.transform(sp.Matrix([[1,2,0],[2,1,0],[0,0,1]])))
print(B.transform(sp.Matrix([[1,2,0],[2,1,0],[0,0,1]])))
```

OUTPUT:
Point2D(4, 5)
Point2D(2, 7)

#2. Rotate the line passing through points A[1 1] and B[5 5] about origin through an angle 90 degrees.

```
import sympy as sp
import math as mt
l=sp.Line((1,1),(5,5))
print(l.rotate(mt.pi/2))
```

OUTPUT:
Line2D(Point2D(-1, 1), Point2D(-5, 5))

#3. If the line segment joining the points A[2 5], B[4 3] is transformed to the line segment A∗B∗ by the transformation matrix, [T] =[2 −3],[4 1] then find the midpoint of A∗B∗.

```
import sympy as sp
l=sp.Segment((2,5),(4,3))
l1=l.midpoint
print(l1.transform(sp.Matrix([[2,-3,0],[4,1,0],[0,0,1]])))
```

OUTPUT:
Point2D(22, -5)

#4. Reflect the line segment joining the points A[5 −3] and B[1 4] through the line y = 2x + 3.

```
x,y=sp.symbols('x,y')
l=sp.Line((5,-3),(1,4))
print(l.reflect(sp.Line(-2*x+y-3)))
```

OUTPUT:
Line2D(Point2D(-39/5, 17/5), Point2D(1/5, 22/5))

#5. Suppose that the line segment between the points A[1 4] and B[3 6] is transformed to the line segment A∗B∗ using the transformation matrix [T] = [2 −1],[1 −3] Find slope of the transformed line segment A∗B∗

```
import sympy as sp
import math as mt
A=sp.Point(1,4)
B=sp.Point(3,6)
L=sp.Line((A.transform(sp.Matrix([[2,-1,0],[1,-
3,0],[0,0,1]]))),(B.transform(sp.Matrix([[2,-1,0],[3,6,0],[0,0,1]]))))
print(L.slope)
```

OUTPUT:
23/9

#6. If the two lines 2x − y = 5 and x + 3y = −1 are transformed using the transformation matrix [T] = [−2 3],[1 1] then find the point of intersection of the transformed lines.

```
import sympy as sp
sp.symbols('x,y')
l1=sp.Line(2*x-y-5)
l2=sp.Line(x+3*y+1)
P=l1.intersection(l2)
p=P[0]
print(p.transform(sp.Matrix([[-2,3,0],[1,1,0],[0,0,1]])))
```

OUTPUT:
Point2D(-5, 5)

#7. If we apply shearing on the line 2x + y = 3 in x and y directions by 2 and -3 units respectively, then find the equation of the resulting line.

```
import sympy as sp
sp.symbols('x,y')
l=sp.Line(2*x+y-3)
p=l.points[0]
q=l.points[1]
l1=sp.Line((p.transform(sp.Matrix([[1,-
3,0],[2,1,0],[0,0,1]]))),(q.transform(sp.Matrix([[1,-3,0],[2,1,0],[0,0,1]]))))
print(l1.equation())
```

OUTPUT:
5*x - 3*y - 21

Practical 8:Study of Graphical aspects of Three-dimensional transformation matrix using matplotlib

#1. Write a python programme to draw a polygon with vertices (0, 1),(1, 0),(−2, 2),(1, −4) and find its area and perimeter

```
import sympy as sp
p=sp.Polygon((0,0),(1,0),(-2,2),(1,-4))
print(p.area)
print(p.perimeter)
```

OUTPUT:
4
1 + sqrt(13) + sqrt(17) + 3*sqrt(5)

#2. Write a python programme to draw a regular polygon with 8 sides and radius 6 centered at origin and find its area and perimeter

```
import sympy as sp
p=sp.Polygon((0,0),6,n=8)
print(p.area)
print(p.perimeter)
```

OUTPUT:
(576 - 288*sqrt(2))/(-4 + 4*sqrt(2))
48*sqrt(2 - sqrt(2))

#3. Write a python programme to draw a regular polygon with 6 sides andradius 1 centered at (1, 2) and find its area and perimeter

```
import sympy as sp
p=sp.Polygon((1,2),1,n=6)
print(p.area)
print(p.perimeter)
```

OUTPUT:
3*sqrt(3)/2
6

#4. Write a python programme to draw a regular polygon with 7 sides and radius 1.5 centered at (2, 2) and reflect it through line x − y = 5

```
import sympy as sp
x,y=sp.symbols('x,y')
p=sp.Polygon((2,2),1.5,n=7)
l=sp.Line(x-y-5)
p.reflect(l)
```

OUTPUT:

```
RegularPolygon(Point2D(7, -3), -1.50000000000000, 7, 3*pi/14)
```

#5. Write a python programme to draw a polygon with vertices (0, 0),(2, 0),(2, 3),(1, 6) and rotate by 180 degrees and find internal angle at

```
# each vertex.
import sympy as sp
import math as mt
A=sp.Point(0,0)
B=sp.Point(2,0)
C=sp.Point(2,3)
D=sp.Point(1,6)
P=sp.Polygon(A,B,C,D)
print(P.rotate(mt.pi/2))
print(P.angles[A])
print(P.angles[B])
print(P.angles[C])
print(P.angles[D])
```

OUTPUT:

```
Polygon(Point2D(0, 0),
Point2D(24492935982947/200000000000000000000000000000, 2), Point2D(-3, 2),
Point2D(-6, 1))
acos(sqrt(37)/37)
pi/2
acos(-3*sqrt(10)/10)
acos(17*sqrt(370)/370)
```

#6. Write a python programme to draw a polygon with vertices (0, 0),(1, 0),(2, 2),(1, 4) and find its area and perimeter.

```
import sympy as sp
p=sp.Polygon((0,0),(1,0),(2,2),(1,4))
print(p.area)
print(p.perimeter)
```

OUTPUT:

```
4
1 + sqrt(17) + 2*sqrt(5)
```

#7. Write a python programme to draw a regular polygon with 4 sides and radius 6 centered at origin and find its area and perimeter.

```
import sympy as sp
p=sp.Polygon((0,0),6,n=4)
print(p.area)
print(p.perimeter)
```

OUTPUT:

```
72
24*sqrt(2)
```

#8. Write a python programme to draw a regular polygon with 8 sides and radius 2 centered at (−1, 2) and find its area and perimeter

```
import sympy as sp
p=sp.Polygon((-1,2),2,n=8)
print(p.area)
print(p.perimeter)
```

OUTPUT:

```
(64 - 32*sqrt(2))/(-4 + 4*sqrt(2))
16*sqrt(2 - sqrt(2))
```

#9. Write a python programme to draw a regular polygon with 7 sides and radius 6 centered at (−2, 2) and reflect it through line x − 2y = 5

```
import sympy as sp
x,y=sp.symbols('x,y')
p=sp.Polygon((-2,2),6,n=7)
l=sp.Line(x-2*y-5)
p.reflect(l)
```

OUTPUT:

$RegularPolygon(Point2D(12/5,-34/5),-6,7,-2\pi/7+\text{atan}(4/3))$

#10. Write a python programme to draw a polygon with vertices (0, 0),(−2, 0),(5, 5),(1, −6) and rotate by 180 degrees and find internal angle

```
#at each vertex.
import sympy as sp
import math as mt
A=sp.Point(0,0)
B=sp.Point(-2,0)
C=sp.Point(5,5)
D=sp.Point(1,-6)
P=sp.Polygon(A,B,C,D)
print(P.rotate(mt.pi))
print(P.angles[A])
print(P.angles[B])
print(P.angles[C])
print(P.angles[D])
```

OUTPUT:

```
Polygon(Point2D(0, 0), Point2D(2, -
244929359829471/1000000000000000000000000000000), Point2D(-5, -5),
Point2D(-1, 6))
acos(-sqrt(37)/37)
-acos(7*sqrt(74)/74) + 2*pi
-acos(83*sqrt(10138)/10138) + 2*pi
-acos(62*sqrt(5069)/5069) + 2*pi
```

Practical 9:Study of effect of concatenation of Two dimensional and Three dimensional transformations

#1. Reflect the pol ABC through the line y = 3, where A[1 0], B[2 − 1],C[−1 3].

```
import sympy  as sp
x,y=sp.symbols('x,y')
P=sp.Polygon((1,0),(2,-1),(-1,3))
P1=sp.Point(0,3)
Q1=sp.Point(1,3)
l=sp.Line(P1,Q1)
print(P.reflect(l))
```

OUTPUT:

```
Triangle(Point2D(1, 6), Point2D(2, 7), Point2D(-1, 3))
```

#2. Rotate the triangle ABC by 90 degree, where A[1 2], B[2 −2], C[−1 2].

```
import sympy  as sp
import math as mt
P=sp.Polygon((1,2),(2,-2),(-1,2))
print(P.rotate(mt.pi/2))
```

OUTPUT:

```
Triangle(Point2D(-2, 1), Point2D(2, 2), Point2D(-2, -1))
```

#3. Find the area and perimeter of the triangle ABC, where A[0 0], B[5 0],C[3 3].

```
import sympy  as sp
P=sp.Polygon((0,0),(5,0),(3,3))
print(P.area)
print(P.perimeter)
```

OUTPUT:

```
15/2
sqrt(13) + 3*sqrt(2) + 5
```

#4. Find the angle at each vertices of the triangle ABC, where A[0 0],B[2 2], C[0 2].

```
import sympy  as sp
A=sp.Point(0,0)
B=sp.Point(2,2)
C=sp.Point(0,2)
T=sp.Triangle(A,B,C)
print(T.angles[A])
print(T.angles[B])
print(T.angles[C])
```

OUTPUT:
pi/4
pi/4
pi/2

#5. Find the angle at each vertices of the triangle PQR, where P[1 0],Q[2 3], R[0 − 2].import sympy as sp

```
P=sp.Point(1,0)
Q=sp.Point(2,3)
R=sp.Point(0,-2)
T=sp.Triangle(P,Q,R)
print(T.angles[P])
print(T.angles[Q])
print(T.angles[R])
```

OUTPUT:
acos(-7*sqrt(2)/10)
acos(17*sqrt(290)/290)
acos(12*sqrt(145)/145)

#6. Reflect the triangle ABC through the line y = −3, where A[1 1],B[2 − 3], C[−1 5].

```
import sympy as sp
x,y=sp.symbols('x,y')
P=sp.Polygon((1,1),(2,-3),(-1,5))
P1=sp.Point(0,-3)
Q1=sp.Point(1,-3)
l=sp.Line(P1,Q1)
print(P.reflect(l))
```

OUTPUT:
Triangle(Point2D(1, -7), Point2D(2, -3), Point2D(-1, -11))

#7. Rotate the triangle ABC by 90 degree, where A[1 − 2], B[4 − 6],C[−1 4].

```
import sympy as sp
import math as mt
P=sp.Polygon((1,-2),(4,-6),(-1,4))
print(P.rotate(mt.pi/2))
```

OUTPUT:
Triangle(Point2D(2, 1), Point2D(6, 4), Point2D(-4, -1))

#8. Find the area and perimeter of the triangle ABC, where A[0 1], B[−5 0],C[3 − 3].

```
import sympy  as sp
P=sp.Polygon((0,0),(-5,0),(3,-3))
print(P.area)
print(P.perimeter)
```

OUTPUT:

```
15/2
3*sqrt(2) + 5 + sqrt(73)
```

#9. Find the angle at each vertices of the triangle ABC, where A[1 1],B[1 2], C[0 1].

```
import sympy  as sp
A=sp.Point(1,1)
B=sp.Point(1,2)
C=sp.Point(0,1)
T=sp.Triangle(A,B,C)
print(T.angles[A])
print(T.angles[B])
print(T.angles[C])
```

OUTPUT:

```
pi/2
pi/4
pi/4
```

#10. Reflect the triangle ABC through the line y = x + 3, where A[−1 0],B[2 − 1], C[1 3].

```
import sympy  as sp
x,y=sp.symbols('x,y')
P=sp.Polygon((-1,0),(2,-1),(1,3))
P1=sp.Point(0,3)
Q1=sp.Point(1,4)
l=sp.Line(P1,Q1)
print(P.reflect(l))
```

OUTPUT:

```
Triangle(Point2D(-3, 2), Point2D(-4, 5), Point2D(0, 4))
```

#11. Rotate the triangle ABC by 270 degree, where A[−1 2], B[2 − 5],C[−1 7].

```
import sympy  as sp
import math as mt
P=sp.Polygon((-1,2),(2,-5),(-1,7))
print(P.rotate(3*mt.pi/2))
```

OUTPUT:

```
Triangle(Point2D(2, 1), Point2D(-5, -2), Point2D(7, 1))
```

#12. Find the area and perimeter of the triangle ABC, where A[0 1], B[−5 0],C[−3 3].

```
import sympy  as sp
P=sp.Polygon((0,1),(-5,0),(-3,3))
print(P.area)
print(P.perimeter)
```

OUTPUT:

```
-13/2
sqrt(26) + 2*sqrt(13)
```

Practical 10:Generation of Bezier curve using given control points

```
#1. If a 2×2 transformation matrix [T] =[1 7][−2 5] is used to transform a line L,
#then the equation of transformed line is y∗ = x∗ + 3.
#Find the equation of original line.
import sympy as sp
sp.symbols('x,y')
l=sp.Line(x-y+3)
p=l.points[0]
q=l.points[1]
m=sp.Matrix([[1,7,0],[-2,5,0],[0,0,1]])
n=m.inv()
p1=p.transform(n)
q1=q.transform(n)
l1=sp.Line(p1,q1)
print(l1.equation())
```

OUTPUT:

6*x/19 + 7*y/19 - 3/19

```
#2. Find the combined transformation of the line segment between the points A[−4
1]  and B[3 0] for the following sequence of transformations:
#  first rotation about origin through an angle π degrees; followed by scaling in x
coordinate by 2 units; followed by reflection through the
#line y =-x
import sympy as sp
import math as mt
a=sp.Point(-4,1)
b=sp.Point(3,0)
s=sp.Segment(a,b)
s=s.rotate(mt.pi)
s=s.scale(2,0)
p=s.points[0]
q=s.points[1]
p1=p.transform(sp.Matrix([[0,-1,0],[-1,0,0],[0,0,1]]))
q1=q.transform(sp.Matrix([[0,-1,0],[-1,0,0],[0,0,1]]))
print(sp.Segment(p1,q1))
```

OUTPUT:

Segment2D(Point2D(0, -8), Point2D(6/19, 3/19))

#3. Suppose that the line segment between the points A[1 −4] and B[5 −6] is transformed to the line segment A∗B∗ using the
transformation matrix [T] = [3 −1],[−5 3]. Find slope of the transformed linesegment A∗B∗.

```
import sympy as sp
import math as mt
A=sp.Point(1,-4)
B=sp.Point(5,-6)
L=sp.Line((A.transform(sp.Matrix([[-3,-1,0],[-
1,5,3],[0,0,1]]))),(B.transform(sp.Matrix([[3,-1,0],[-5,3,0],[0,0,1]]))))
print(L.slope)
```

OUTPUT:
-1/22

#4. If the two lines 2x + y = 0 and x − 3y = 1 are transformed using
#the transformation matrix [T] = [2 −3],[−1 −1] then find the point of
#intersection of the transformed lines.

```
import sympy as sp
x,y=sp.symbols('x,y')
l1=sp.Line(2*x+y)
l2=sp.Line(x-3*y-1)
P=l1.intersection(l2)
p=P[0]
print(p.transform(sp.Matrix([[2,-3,0],[-1,1,0],[0,0,1]])))
```

OUTPUT:
Point2D(4/7, -5/7)

#5. If we apply shearing on the line 2x − y = 8 in x and y directions by 4 and 6 units respectively, then find the equation of the resulting line.

```
import sympy as sp
sp.symbols('x,y')
l=sp.Line(2*x-y-8)
p=l.points[0]
q=l.points[1]
l1=sp.Line((p.transform(sp.Matrix([[1,6,0],[4,1,0],[0,0,1]]))),(q.transform(sp.Matrix([
[1,6,0],[4,1,0],[0,0,1]]))))
print(l1.equation())
```

OUTPUT:
-8*x + 9*y - 184

#6. If a −3 × 2 transformation matrix [T] = [1 3],[2 2] is used to transform a line L, then the equation of transformed line is y∗ = x∗− 3.

```
# Find the equation of original line.
import sympy as sp
x,y=sp.symbols('x,y')
l=sp.Line(x-y-3)
p=l.points[0]
q=l.points[1]
m=sp.Matrix([[1,3,0],[2,2,0],[0,0,1]])
n=m.inv()
p1=p.transform(n)
q1=q.transform(n)
l1=sp.Line(p1,q1)
print(l1.equation())
```

OUTPUT:

x + 3/2

#7. Find the combined transformation of the line segment between the points A[4 1] and B[−3 0] for the following sequence of transformations:

```
# first rotation about origin through an angle π degrees; followed by scaling in x coordinate by 3 units; followed by reflection through the
#line y=3*x+9
import sympy as sp
import math as mt
a=sp.Point(4,1)
b=sp.Point(-3,0)
s=sp.Segment(a,b)
s=s.rotate(mt.pi)
s=s.scale(3,0)
s=s.reflect(sp.Line(3*x-y+9))
print(s)
```

OUTPUT:

Segment2D(Point2D(21/5, -27/5), Point2D(-63/5, 36/5))

Practical 11:Study of Operational Research in Python (Unit 5-5.1)

```
#1. Solve the following LLP:
#Max Z = 150x + 75y
#subject to,
#4x + 6y ≤ 24
#5x + 3y ≤ 15
#x ≥ 0, y ≥ 0
import pulp as pl
model=pl.LpProblem(sense=pl.LpMaximize)
x=pl.LpVariable(name='x',lowBound=0)
y=pl.LpVariable(name='y',lowBound=0)
model +=(4*x+6*y<=24)
model +=(5*x+3*y<=15)
model += 150*x+75*y
print(model)
print(model.solve())
print(model.objective.value())
print(x.value())
print(y.value())
```

OUTPUT:

```
NoName:
MAXIMIZE
150*x + 75*y + 0
SUBJECT TO
_C1: 4 x + 6 y <= 24

_C2: 5 x + 3 y <= 15

VARIABLES
x Continuous
y Continuous

1
450.0
3.0
0.0
```

```
#2. Solve the following LLP:
#Min. Z = 3.5x + 2y
#subject to,
#x + y ≥ 5
#x ≥ 4
#y ≤ 2
#x ≥ 0, y ≥0
import pulp as pl
model=pl.LpProblem(sense=pl.LpMinimize)
x=pl.LpVariable(name='x',lowBound=0)
y=pl.LpVariable(name='y',lowBound=0)
model +=(x+y>=5)
model +=(x>=4)
model +=(y<=2)
model += 3.5*x+2*y
print(model)
print(model.solve())
print(model.objective.value())
print(x.value())
print(y.value())
```

OUTPUT:

```
NoName:
MINIMIZE
3.5*x + 2*y + 0.0
SUBJECT TO
_C1: x + y >= 5

_C2: x >= 4

_C3: y <= 2

VARIABLES
x Continuous
y Continuous

1
16.0
4.0
1.0
```

```
#3. Solve the following LLP:
#Max. Z = 3x + 5y + 4z
#subject to,
#2x + 3y ≤ 8
#2y + 5z ≤ 10
#3x + 2y + 4z ≤ 15
#x ≥ 0, y ≥ 0 z ≥ 0
import pulp as pl
model=pl.LpProblem(sense=pl.LpMaximize)
x=pl.LpVariable(name='x',lowBound=0)
y=pl.LpVariable(name='y',lowBound=0)
z=pl.LpVariable(name='z',lowBound=0)
model +=(2*x+3*y<=8)
model +=(2*x+5*z<=10)
model +=(3*x+2*y+4*z<=15)
model += 3*x+5*y+4*z
print(model)
print(model.solve())
print(model.objective.value())
print(x.value())
print(y.value())
print(z.value())
```

OUTPUT:

```
NoName:
MAXIMIZE
3*x + 5*y + 4*z + 0
SUBJECT TO
_C1: 2 x + 3 y <= 8

_C2: 2 x + 5 z <= 10

_C3: 3 x + 2 y + 4 z <= 15

VARIABLES
x Continuous
y Continuous
z Continuous

1
21.33335
0.0
2.66667
2.0
```

```
#4. Solve the following LLP:
#Min. Z = x + 2y + z
#subject to,
#x +1/2y +1/2z ≤ 1
#3/2x + 2y + z ≥ 8
#x ≥ 0, y ≥ 0, z ≥ 0
import pulp as pl
model=pl.LpProblem(sense=pl.LpMinimize)
x=pl.LpVariable(name='x',lowBound=0)
y=pl.LpVariable(name='y',lowBound=0)
z=pl.LpVariable(name='z',lowBound=0)
model +=(x+1/2*y+1/2*z<=1)
model +=(3*x/2+2*y>=8)
model += x+2*y+z
print(model)
print(model.solve())
print(model.objective.value())
print(x.value())
print(y.value())
print(z.value())
```

OUTPUT:

```
NoName:
MINIMIZE
1*x + 2*y + 1*z + 0
SUBJECT TO
_C1: x + 0.5 y + 0.5 z <= 1

_C2: 1.5 x + 2 y >= 8

VARIABLES
x Continuous
y Continuous
z Continuous

-3
0.0
0.0
0.0
0.0
```

```
#5. Solve the following LLP:
#Min. Z = x + y
#subject to,
#x ≥ 6
#y ≥ 6
#x + y ≤ 11
#x ≥ 0, y ≥ 0
import pulp as pl
model=pl.LpProblem(sense=pl.LpMinimize)
x=pl.LpVariable(name='x',lowBound=0)
y=pl.LpVariable(name='y',lowBound=0)
model +=(x>=6)
model +=(y>=6)
model +=(x+y<=11)
model += x+y
print(model)
print(model.solve())
print(model.objective.value())
print(x.value())
print(y.value())
```

OUTPUT:

```
NoName:
MINIMIZE
1*x + 1*y + 0
SUBJECT TO
_C1: x >= 6

_C2: y >= 6

_C3: x + y <= 11

VARIABLES
x Continuous
y Continuous

-3
0.0
0.0
0.0
```

```
#6. Solve the following LLP:
#Max. Z = x + y
#subject to,
#x − y ≥ 1
#x + y ≥ 2
import pulp as pl
model=pl.LpProblem(sense=pl.LpMaximize)
x=pl.LpVariable(name='x',lowBound=0)
y=pl.LpVariable(name='y',lowBound=0)
model +=(x-y>=1)
model +=(x+y>=2)
model += x+y
print(model)
print(model.solve())
print(model.objective.value())
print(x.value())
print(y.value())
```

OUTPUT:

```
NoName:
MAXIMIZE
1*x + 1*y + 0
SUBJECT TO
_C1: x - y >= 1

_C2: x + y >= 2

VARIABLES
x Continuous
y Continuous

-3
0.0
0.0
0.0
```

```
#7. Solve the following LLP:
#Max. Z = 4x + y + 3z + 5w
#subject to,
#4x + 6y − 5z − 4x ≥ −20
#−3x − 2y + 4z + w ≤ 10
#−8x − 3y + 3z + 2w ≤ 20
#x ≥ 0, y ≥ 0, z ≥ 0, w ≥ 0
import pulp as pl
model=pl.LpProblem(sense=pl.LpMaximize)
x=pl.LpVariable(name='x',lowBound=0)
y=pl.LpVariable(name='y',lowBound=0)
z=pl.LpVariable(name='z',lowBound=0)
w=pl.LpVariable(name='w',lowBound=0)
model +=(4*x+6*y-5*z>=-20)
model +=(-3*x-2*y+4*z+w<=10)
model +=(-8*x-3*y+3*z+2*w<=20)
model += 4*x+y+3*z+5*w
print(model)
print(model.solve())
print(model.objective.value())
print(x.value())
print(y.value())
```

OUTPUT:

```
NoName:
MAXIMIZE
5*w + 4*x + 1*y + 3*z + 0
SUBJECT TO
_C1: 4 x + 6 y - 5 z >= -20

_C2: w - 3 x - 2 y + 4 z <= 10

_C3: 2 w - 8 x - 3 y + 3 z <= 20

VARIABLES
w Continuous
x Continuous
y Continuous
z Continuous

-3
0.0
0.0
0.0
```

Practical 12:

```
#1. A beer company has two warehouses from which it distributes beer to four carefully chosen shops. At the start of every week, each barsends
#  an order to the brewery’s head office for so many crates of beer,which is then dispatched from the appropriate warehouse to the bar.The
#  brewery would like to have an interactive computer program whichthey can run week by week to tell them which warehouse should supply which
#  bar so as to minimize the costs of the whole operation. For example, suppose that at the start of a given week the brewery has 2000 cases
#  at warehouse A, and 4500 cases at warehouse B, and that the bars require 600, 1000, 1600, 150 and 800 cases respectively. Transportation costs
#  (dollars per crate) is given in the following table.
# From Warehouse to Bar A B
#            1 3 2
#            2 4 3
#            3 6 3
#            4 2 1
#Which warehouse should supply which bar?
import pulp as pl
warehouses=["a","b"]
supply={"a":2000,"b":4500}
bars=["1","2","3","4"]
demand={"1":600,"2":1000,"3":1600,"4":800}
costs={"a":{"1":3,"2":4,"3":6,"4":2},"b":{"1":2,"2":3,"3":3,"4":1}}
prob=pl.LpProblem("beer distribution problem",pl.LpMinimize)
routes=[(w,b) for  w in warehouses for b in bars]
vars=pl.LpVariable.dicts("Route",(warehouses,bars),0,None,pl.LpInteger)
prob +=pl.lpSum([vars[w][b]*costs[w][b] for (w,b) in routes])
for w in warehouses:
   prob +=pl.lpSum([vars[w][b] for b in bars])<=supply[w]
for b in bars:
   prob +=pl.lpSum([vars[w][b] for w in warehouses])>=demand[b]
print(prob.writeLp())
print(prob.objective.value())
```

```
OUTPUT:
MINIMIZE
2*Route_a_1 + 4*Route_a_2 + 5*Route_a_3 + 2*Route_a_4 + 1*Route_a_5 +
3*Route_b_1 + 1*Route_b_2 + 3*Route_b_3 + 2*Route_b_4 + 3*Route_b_5 + 0
SUBJECT TO
_C1: Route_a_1 + Route_a_2 + Route_a_3 + Route_a_4 + Route_a_5 <= 1000

_C2: Route_b_1 + Route_b_2 + Route_b_3 + Route_b_4 + Route_b_5 <= 4000

_C3: Route_a_1 + Route_b_1 >= 500

VARIABLES
0 <= Route_a_1 Integer
0 <= Route_a_2 Integer
0 <= Route_a_3 Integer
0 <= Route_a_4 Integer
0 <= Route_a_5 Integer
0 <= Route_b_1 Integer
0 <= Route_b_2 Integer
0 <= Route_b_3 Integer
0 <= Route_b_4 Integer
0 <= Route_b_5 Integer
beer_distribution_problem:
MINIMIZE
2*Route_a_1 + 4*Route_a_2 + 5*Route_a_3 + 2*Route_a_4 + 1*Route_a_5 +
3*Route_b_1 + 1*Route_b_2 + 3*Route_b_3 + 2*Route_b_4 + 3*Route_b_5 + 0
...
0 <= Route_b_3 Integer
0 <= Route_b_4 Integer
0 <= Route_b_5 Integer
9600
```