Solutions to Homework 1

Question 1: 10 points. Write a program that solves for all positive integer pairs, i.e., a, $b \geq 0$,

$$\sqrt{a} + \sqrt{b} = \sqrt{n} \tag{1}$$

where n = 2025.

Hint: 2025 happens to be a perfect square, with the prime factorization of $3 \cdot 3 \cdot 3 \cdot 3 \cdot 5 \cdot 5$. You can use this fact and a little bit of number theory to see that there will only be 46 solutions (i.e., (a,b) pairs).

Python Source Code:

```
# ====================================================================
# TestIntegerSolutions2025.py: Compute positive integer solutions to:
#
#         math.sqrt(a) + math.sqrt(b) = math.sqrt(2025).
#
# where a and b are integers greater than or equal to zero.
#
# Notice: 2025 is a perfect square (3*3*5)*(3*3*5).
#
# Written by: Mark Austin                                  January, 2025
# ====================================================================

import math

# main method ...

def main():
    print("--- Enter TestIntegerSolutions2025.main()    ... ");
    print("--- ======================================== ... ");

    n = 2025

    # Part 1: Use nested for loop to exhaustively test all cases.
    #       : Naively test for equality ...

    print("--- ")
    print("--- Part 1: Nested loops, naive test for equality ...")

    i = 1
```

```
    for a in range(0, n+1):
        for b in range(0, n+1):
            error = math.sqrt(a) + math.sqrt(b) - math.sqrt(n)
            if error == 0:
               print("--- soln {:2d}: a = {:6d}, b = {:6d}: solution !! ...".format(i,a,b) );
               i = i + 1

    # Part 2: Use nested for loop to exhaustively test all cases.
    #       : Account for imprecise number representation ...

    print("--- ")
    print("--- Part 2: Nested loops, account for inexact arithmetic ...")

    i = 1
    for a in range(0, n+1):
        for b in range(0, n+1):
            error = math.sqrt(a) + math.sqrt(b) - math.sqrt(n)
            if math.fabs(error) < 0.00000000000001:
               print("--- soln {:2d}: a = {:6d}, b = {:6d}: solution !! ...".format(i,a,b) );
               i = i + 1

    # Part 3: Use number theory to reduce number of cases ...

    print("--- ")
    print("--- Part 3: Use number theory to reduce number of cases to evaluate ...")

    mmax = 45
    m = list( range(0, mmax + 1) );

    # Traverse list and verify expression values ...

    for i in m:
        j = mmax - i
        a = i*i
        b = j*j
        error = math.sqrt(a) + math.sqrt(b) - math.sqrt(n)
        if math.fabs(error) < 0.00000000000001:
           print("--- a = {:6d}, b = {:6d}: solution !! ...".format(a,b) );
        else:
           print("--- a = {:6d}, b = {:6d}: not a solution !! ...".format(a,b) );

    print("--- ======================================== ... ");
    print("--- Leave TestIntegerSolutions2025.main()    ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

Program Output: The abbreviated textual output is:

```
--- Enter TestIntegerSolutions2025.main()    ...
--- ======================================== ...
```

```
---
--- Part 1: Nested loops, naive test for equality ...
--- soln  1: a =      0, b =   2025: solution !! ...
--- soln  2: a =      1, b =   1936: solution !! ...
--- soln  3: a =      4, b =   1849: solution !! ...
--- soln  4: a =      9, b =   1764: solution !! ...
--- soln  5: a =     16, b =   1681: solution !! ...
--- soln  6: a =     25, b =   1600: solution !! ...
--- soln  7: a =     36, b =   1521: solution !! ...
--- soln  8: a =     49, b =   1444: solution !! ...
--- soln  9: a =     64, b =   1369: solution !! ...
--- soln 10: a =     81, b =   1296: solution !! ...
--- soln 11: a =    100, b =   1225: solution !! ...
--- soln 12: a =    121, b =   1156: solution !! ...
--- soln 13: a =    144, b =   1089: solution !! ...
--- soln 14: a =    169, b =   1024: solution !! ...
--- soln 15: a =    196, b =    961: solution !! ...
--- soln 16: a =    225, b =    900: solution !! ...
--- soln 17: a =    256, b =    841: solution !! ...
--- soln 18: a =    289, b =    784: solution !! ...
--- soln 19: a =    324, b =    729: solution !! ...
--- soln 20: a =    361, b =    676: solution !! ...
--- soln 21: a =    400, b =    625: solution !! ...
--- soln 22: a =    441, b =    576: solution !! ...
--- soln 23: a =    484, b =    529: solution !! ...
--- soln 24: a =    529, b =    484: solution !! ...
--- soln 25: a =    576, b =    441: solution !! ...
--- soln 26: a =    625, b =    400: solution !! ...
--- soln 27: a =    676, b =    361: solution !! ...
--- soln 28: a =    729, b =    324: solution !! ...
--- soln 29: a =    784, b =    289: solution !! ...
--- soln 30: a =    841, b =    256: solution !! ...
--- soln 31: a =    900, b =    225: solution !! ...
--- soln 32: a =    961, b =    196: solution !! ...
--- soln 33: a =   1024, b =    169: solution !! ...
--- soln 34: a =   1089, b =    144: solution !! ...
--- soln 35: a =   1156, b =    121: solution !! ...
--- soln 36: a =   1225, b =    100: solution !! ...
--- soln 37: a =   1296, b =     81: solution !! ...
--- soln 38: a =   1369, b =     64: solution !! ...
--- soln 39: a =   1444, b =     49: solution !! ...
--- soln 40: a =   1521, b =     36: solution !! ...
--- soln 41: a =   1600, b =     25: solution !! ...
--- soln 42: a =   1681, b =     16: solution !! ...
--- soln 43: a =   1764, b =      9: solution !! ...
--- soln 44: a =   1849, b =      4: solution !! ...
--- soln 45: a =   1936, b =      1: solution !! ...
--- soln 46: a =   2025, b =      0: solution !! ...
---
--- Part 2: Nested loops, account for inexact arithmetic ...
--- soln  1: a =      0, b =   2025: solution !! ...
--- soln  2: a =      1, b =   1936: solution !! ...

... lines of output removed ...
```

```
--- soln 44: a =   1849, b =      4: solution !! ...
--- soln 45: a =   1936, b =      1: solution !! ...
--- soln 46: a =   2025, b =      0: solution !! ...
---
--- Part 3: Use number theory to reduce number of cases to evaluate ...
--- a =      0, b =   2025: solution !! ...
--- a =      1, b =   1936: solution !! ...
--- a =      4, b =   1849: solution !! ...

... lines of output removed ...

--- a =   1849, b =      4: solution !! ...
--- a =   1936, b =      1: solution !! ...
--- a =   2025, b =      0: solution !! ...
--- ======================================== ...
--- Leave TestIntegerSolutions2025.main()    ...
```

Question 2: 10 points. A square of sheet metal having side length 2L cm has four pieces cut out symmetrically from the corners as shown in Figure 1.

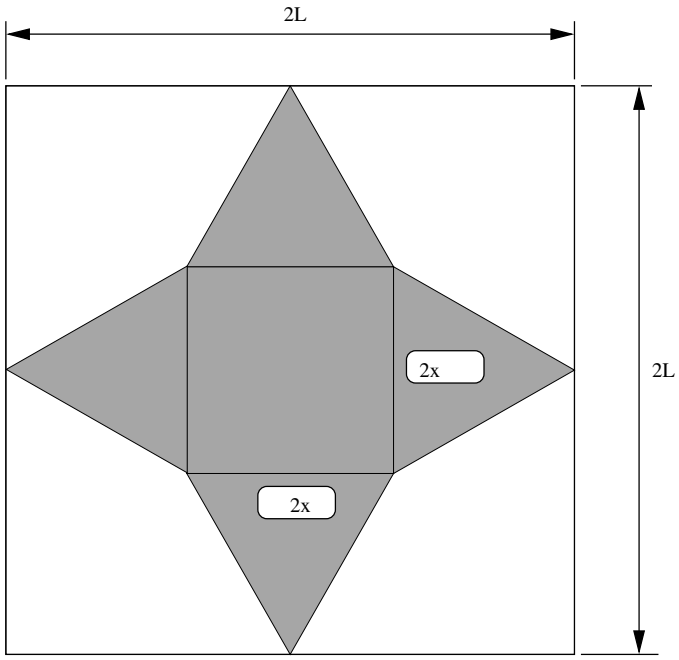

Figure 1: Sheetmetal schematic for a folded pyramid.

Assuming that L is a constant and L $>$ 2x, then the remaining metal can be folded into a pyramid having volume:

$$\text{Volume}(x) = \frac{4x^2}{3}\sqrt{(L^2 - 2Lx)} \quad \text{cm}^3. \tag{2}$$

The maximum volume occurs when x = 2L/5 cm.

Write a Python program that sets a value for length L = 10 cm, and then systematically computes and print volumes for appropriate values of x. Organize your output into a tidy table, e.g., something like:

```
--- Pyramid: L = 10.0 cm
+--------+---------------+
| X (cm) | Volume (cm^3) |
+--------+---------------+
|  0.00  |          0.00 |
|  0.50  |          3.16 |
|  1.00  |         11.93 |
|  1.50  |         25.10 |
|  2.00  |         41.31 |
|  2.50  |         58.93 |
```

```
|  3.00  |         75.89 |
|  3.50  |         89.46 |
|  4.00  |         95.41 |
|  4.50  |         85.38 |
|  5.00  |          0.00 |
+--------+---------------+
```

Python has a package called prettytable (i.e., pip3 install prettytable) which you might find useful. A small test program for pretty tables can be found in: python-code.d/basics/TestPrettyTable01.py.

Python Source Code:

```
# ============================================================
# TestPyramidVolume02.py: Compute pyramid volume. Display results
#                         in a prettytable.
#
# Written by: Mark Austin                              January, 2025
# ============================================================

import math
import numpy as np

from prettytable import PrettyTable

# Compute pyramid volume (for x <= 0.5L).

def PyramidVolume(L, x):
    return (4.0/3.0)*x*x*math.sqrt(L*L - 2*L*x);

# Main function ...

def main():
    print("--- Enter TestPyramidVolume.main()           ... ");
    print("--- ======================================== ... ");
    print("");

    # Create table of coordinates and pyramid volumes ...

    L = 10 # <--- L = 10 cm ...

    # Create array of x coordinates ...

    coords = np.linspace(0,5.0,11)

    # Compute pyramid volumes ...

    volume = []
    for x in coords:
        vol = PyramidVolume(L,x);
        volume.append( vol );

    print("--- ");
```

```
    print("--- Create table of pyramid volumes ...");
    print("--- ");

    table01 = PrettyTable()
    table01.add_column( "X (cm)", coords )
    table01.add_column( "Volume (cm^3)", volume )

    table01.float_format = ".2"           # print floating point numbers to 2 decimal places ...
    table01.align["Volume (cm^3)"] = "r"  # right alignment of volume column ...

    print("--- Pyramid: L = {:.1f} cm".format(L));
    print(table01)

    print("--- ");
    print("--- ======================================== ... ");
    print("--- Leave TestPyramidVolume02.main()         ... ");

# call the main method ...

main()
```

Program Output:

```
--- Pyramid: L = 10.0 cm
+--------+---------------+
| X (cm) | Volume (cm^3) |
+--------+---------------+
|  0.00  |          0.00 |
|  0.50  |          3.16 |
|  1.00  |         11.93 |
|  1.50  |         25.10 |
|  2.00  |         41.31 |
|  2.50  |         58.93 |
|  3.00  |         75.89 |
|  3.50  |         89.46 |
|  4.00  |         95.41 |
|  4.50  |         85.38 |
|  5.00  |          0.00 |
+--------+---------------+
```

Question 3: 10 points. Figure 2 shows a two-dimensional grid of masses.

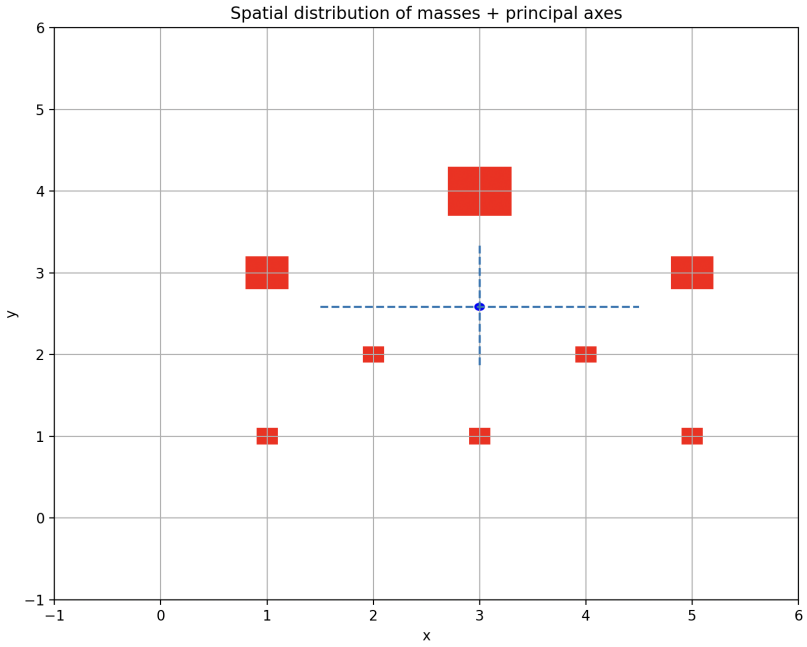

Figure 2: Two-dimensional grid of masses + principal axes.

If the total number of point masses is denoted by N, then the total mass of the grid, M, is given by

$$M = \sum_{i=1}^{N} m_i \tag{3}$$

The coordinates of the grid centroid, $(\bar{x}, \bar{y})$, are defined by:

$$M\bar{x} = \sum_{i=1}^{N} x_i \cdot m_i \quad \text{and} \quad M\bar{y} = \sum_{i=1}^{N} y_i \cdot m_i \tag{4}$$

The moments of inertia about the x- and y-axes are given by:

$$I_{xx} = \sum_{i=1}^{N} y_i^2 \cdot m_i \quad \text{and} \quad I_{yy} = \sum_{i=1}^{N} x_i^2 \cdot m_i \tag{5}$$

respectively. Similarly the cross moment of inertia is given by

$$I_{xy} = \sum_{i=1}^{N} x_i \cdot y_i \cdot m_i \tag{6}$$

With solutions to equations 4 - 6 in hand, the corresponding moments of inertia about the centroid are given by the parallel axes theorem (Google: parallel axis theorem moments of inertia). Finally, the orientation of the principle axes are given by

$$\tan(2\theta) = \left[\frac{2I_{xy}}{I_{xx} - I_{yy}} \right] \tag{7}$$

Now suppose that the (x,y) coordinates and masses are stored in two arrays;

```
mass  = np.array( [ 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 3.0, 2.0 ] );

coord = np.array( [ ( 1.0, 1.0 ),
                    ( 2.0, 2.0 ),
                    ( 3.0, 1.0 ),
                    ( 4.0, 2.0 ),
                    ( 5.0, 1.0 ),
                    ( 5.0, 3.0 ),
                    ( 3.0, 4.0 ),
                    ( 1.0, 3.0 ) ] );
```

Write a Python program to evaluate equations 3 – 7, and create a plot in Python similar to Figure 2. Add the centroid and principal axes (drawn with the appropriate orientation) to your plot.

Python Source Code:

```
# ============================================================
# TestMomentsInertia02.py: Compute moments of inertia about the
# origin, centroid (x,y), and orientation of principal axes.
#
# Written by: Mark Austin                          January, 2025
# ============================================================

import math
import numpy as np
import matplotlib.pyplot as plt

from matplotlib.patches import Rectangle
from matplotlib.patches import Circle
from matplotlib.lines   import Line2D

# Main function ...
```

```
def main():
    print("--- Enter TestMomentsInertia02.main()        ... ");
    print("--- ======================================== ... ");
    print("");

    # Homework 01: Mass and coordinate arrays

    mass  = np.array( [1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 3.0, 2.0])
    coord = np.array( [ ( 1.0, 1.0 ), ( 2.0, 2.0 ), ( 3.0, 1.0 ), ( 4.0, 2.0 ),
                        ( 5.0, 1.0 ), ( 5.0, 3.0 ), ( 3.0, 4.0 ), ( 1.0, 3.0 ) ] );

    # Computing inertias about the axes and origin.

    Ixx = 0.0; Iyy = 0.0; Ixy = 0.0;
    for i in range(len(mass)):
        Ixx = Ixx + mass[i]*coord[i][1]*coord[i][1]
        Iyy = Iyy + mass[i]*coord[i][0]*coord[i][0]
        Ixy = Ixy + mass[i]*coord[i][0]*coord[i][1]

    # Print results ...

    print("--- Ixx (about axis)   = {:.2f}  ...".format(Ixx) );
    print("--- Iyy (about axis)   = {:.2f}  ...".format(Iyy) );
    print("--- Ixy (about origin) = {:.2f}  ...".format(Ixy) );

    # Compute centroid of masses (x,y) ...

    M = sum (mass);
    print("--- Total mass = {:.2f}  ...".format(M) );

    fmMassX = 0.0; fmMassY = 0.0;
    for i in range(len(mass)):
        fmMassX = fmMassX + mass[i]*coord[i][0];
        fmMassY = fmMassY + mass[i]*coord[i][1];

    centroidX = fmMassX/M;
    centroidY = fmMassY/M;

    print("--- Centroid (x,y) = ({:.2f}, {:.2f})  ...".format(centroidX, centroidY ) );

    # Use parallel axis theorem to compute interias about centroid ...

    IxxCentroid = Ixx - M*centroidY*centroidY;
    IyyCentroid = Iyy - M*centroidX*centroidX;
    IxyCentroid = Ixy - M*centroidX*centroidY;

    print("--- Ixx (about centroid) = {:.2f}  ...".format(IxxCentroid) );
    print("--- Iyy (about centroid) = {:.2f}  ...".format(IyyCentroid) );
    print("--- Ixy (about centroid) = {:.2f}  ...".format(IxyCentroid) );

    # Compute orientation of principal axes ...

    angle = 0.0;
    if(Ixx != Iyy):
       angle = (1.0/2.0)*math.atan(IxyCentroid/(IxxCentroid - IyyCentroid));
```

```
print("--- Mohr's circle: angle = {:.3f} radians ...".format(angle) );

# Plot masses and coordinates ...
# Define Matplotlib figure and axis

fig, ax = plt.subplots()

# Draw masses as small/medium-sized rectangles ...

for i in range(len(mass)):

    xcoord = coord[i][0];
    ycoord = coord[i][1];

    dm = mass[i];
    if dm == 1:
       width = 0.2;
    elif dm == 2:
       width = 0.4;
    else:
       width = 0.6;

    ax.add_patch(Rectangle( (xcoord - width/2, ycoord-width/2), width, width, facecolor='red'))

# Draw centroid ...

radius = 0.05;
ax.add_patch( Circle( (centroidX, centroidY), radius, facecolor='blue') )

# Plot major principal axis ...

axislength = 1.5
x1 = centroidX - axislength*math.cos(angle);
y1 = centroidY - axislength*math.sin(angle);
x2 = centroidX + axislength*math.cos(angle);
y2 = centroidY + axislength*math.sin(angle);

xcoords = [ x1, x2 ]
ycoords = [ y1, y2 ]
ax.add_line( Line2D(xcoords, ycoords, linestyle='--') )

# Plot minor principal axis ...

x1 = centroidX - axislength/2.0*math.sin(angle);
y1 = centroidY + axislength/2.0*math.cos(angle);
x2 = centroidX + axislength/2.0*math.sin(angle);
y2 = centroidY - axislength/2.0*math.cos(angle);

xcoords = [ x1, x2 ]
ycoords = [ y1, y2 ]
ax.add_line( Line2D(xcoords, ycoords, linestyle='--') )

plt.title('Spatial distribution of masses + principal axes')
plt.ylabel('y')
```

```
    plt.xlabel('x')
    plt.ylim( -1, 6)
    plt.xlim( -1, 6)
    plt.grid(True)
    plt.show()

    print("--- ======================================== ... ");
    print("--- Leave TestMomentsInertia02.main()        ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

Program Output: The abbreviated textual output is as shown in the question description.

```
--- Enter TestMomentsInertia02.main()        ...
--- ======================================== ...

--- Ixx (about axis)   = 95.00  ...
--- Iyy (about axis)   = 134.00  ...
--- Ixy (about origin) = 93.00  ...
--- Total mass = 12.00  ...
--- Centroid (x,y) = (3.00, 2.58)  ...
--- Ixx (about centroid) = 14.92  ...
--- Iyy (about centroid) = 26.00  ...
--- Ixy (about centroid) = 0.00  ...
--- Mohr's circle: angle = -0.000 radians ...
--- ======================================== ...
--- Leave TestMomentsInertia02.main()        ...
```

The graphical output is shown in Figure 2.

Question 4: 10 points. Figure 3 shows the cross-section of a T-shaped beam (also called T-beam). Reinforced concrete T-beams are commonly found in buildings and highway bridges.

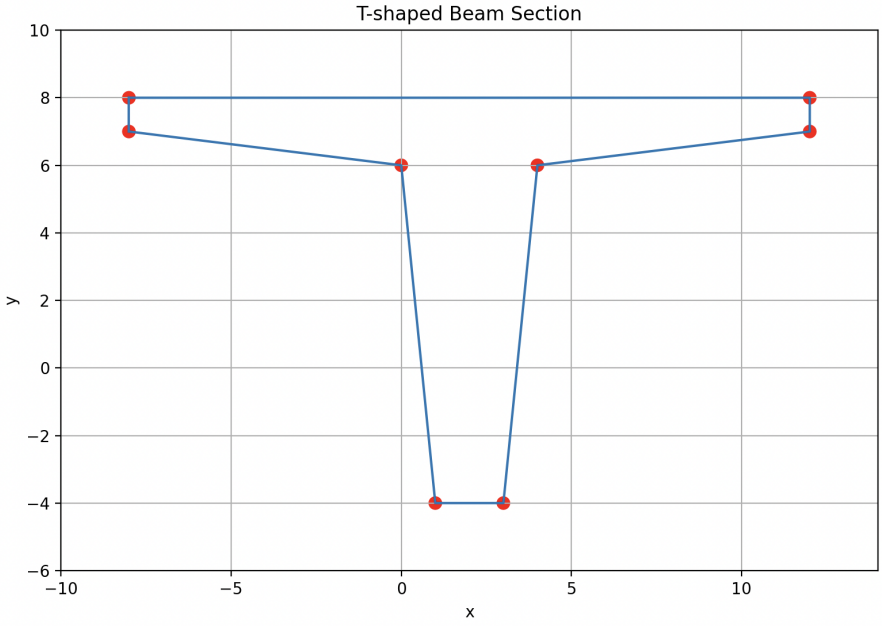

Figure 3: T-shaped beam cross section.

Under service load conditions, T-beams are expected to behave elastically, with very small displacements and no long-term damage. From a mechanics standpoint, the associated elastic analysis procedures require a knowledge of the section area and centroid, and moments of inertia. The purpose of this question is to take a first step toward the development of python code that will compute these section properties automatically. Later on (i.e., homeworks 2 and 3) we will step things up a bit by adding holes to the cross section, and modeling the whole cross section as an object.

Getting Started. The T-beam shown in Figure 3 has (x, y) coordinates stored as two columns of a numpy array:

```
coord = np.array( [ (  -8.0,   8.0 ),
                    (  12.0,   8.0 ),
                    (  12.0,   7.0 ),
                    (   4.0,   6.0 ),
                    (   3.0,  -4.0 ),
                    (   1.0,  -4.0 ),
                    (   0.0,   6.0 ),
                    (  -8.0 ,  7.0 ) ] );
```

Write a Python program that will:

1. Compute and print the minimum and maximum polygon coordinates in both the x and y directions.
2. Compute and print the minimum and maximum distance of the polygon vertices from the coordinate system origin.
3. Create a plot of the T-beam similar to Figure 3.
4. Write functions perimeter() and area() to compute the perimeter and area of the T-beam, respectively.

Python Source Code:

```
# ============================================================
# TestPolygonTshapedBeam01.py: Compute centroid and moments of
# inertia for a T-shaped beam cross section.
#
# Written by: Mark Austin                                January 2025
# ============================================================

import math
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from matplotlib.lines import Line2D

# Function to print two-dimensional matrices ...

def PrintMatrix(name, a):
    print("Matrix: {:s} ".format(name) );
    for row in a:
        for col in row:
            print("{:8.4f}".format(col), end=" ")
        print("")

# Compute polygon perimeter ...

def perimeter( coord ):
    norows = coord.shape[0];

    dperimeter = 0.0;
    for i in range(1,norows):
        dx = coord[i][0] - coord[i-1][0];
        dy = coord[i][1] - coord[i-1][1];
        dperimeter = dperimeter + math.sqrt( dx*dx + dy*dy );

    dx = coord[norows-1][0] - coord[0][0];
    dy = coord[norows-1][1] - coord[0][1];
    dperimeter = dperimeter + math.sqrt( dx*dx + dy*dy );

    return dperimeter;

# Compute polygon area ...

def area( coord ):
```

```
    norows = coord.shape[0];

    darea = 0.0;
    for i in range(1,norows):
        dx = coord[i][0] - coord[i-1][0];
        dy = coord[i][1] + coord[i-1][1];
        darea = darea + dx*dy/2.0;

    dx = coord[0][0] - coord[norows-1][0];
    dy = coord[0][1] + coord[norows-1][1];
    darea = darea + dx*dy/2.0;

    return darea;

# ================================================================
# main method ...
# ================================================================

def main():
    print("--- Enter TestPolygonTshapedBeam01.main()  ... ");
    print("--- ===================================== ... ");

    print("--- Part 1: Initialize t-beam coordinates ... ");

    coord = np.array( [ (  -8.0,  8.0 ),
                        (  12.0,  8.0 ),
                        (  12.0,  7.0 ),
                        (   4.0,  6.0 ),
                        (   3.0, -4.0 ),
                        (   1.0, -4.0 ),
                        (   0.0,  6.0 ),
                        (  -8.0 , 7.0 ) ] );

    PrintMatrix("T-beam Coordinates", coord);

    print("--- ");
    print("--- Part 2: Max/min coordinate positions ... ");
    print("--- ");

    print("--- Min x = {:8.3f} ...".format( min ( coord[:,0] ) ) );
    print("--- Max x = {:8.3f} ...".format( max ( coord[:,0] ) ) );
    print("--- Min y = {:8.3f} ...".format( min ( coord[:,1] ) ) );
    print("--- Max y = {:8.3f} ...".format( max ( coord[:,1] ) ) );

    print("--- ");
    print("--- Part 3: Compute and print distance of coords from origin ... ");
    print("--- ");

    distance = np.zeros( coord.shape[0] )
    for i in range( coord.shape[0] ):
        x = coord[i][0];
        y = coord[i][1];
        distance[i] = math.sqrt(x**2 + y**2)

    print("--- Min distance from origin = {:8.3f} ...".format( min ( distance ) ) );
```

```
    print("--- Max distance from origin = {:8.3f} ...".format( max ( distance ) ) );

    print("--- ");
    print("--- Part 4: Compute and print perimeter and area of polygon ... ");
    print("--- ");

    print("--- Section perimeter = {:8.3f} ...".format( perimeter(coord) ) );
    print("--- Section area      = {:8.3f} ...".format( area(coord) ) );

    print("--- ");
    print("--- Part 5: Plot section verticies and edges ...");
    print("--- ");

    # Define Matplotlib figure and axis

    fig, ax = plt.subplots()

    # Draw vertices as small circles ...

    width = 0.2;
    for i in range(len(coord)):
        xcoord = coord[i][0]; ycoord = coord[i][1];
        ax.add_patch( Circle( (xcoord, ycoord), width, facecolor='red') )

    # Draw section edges ...

    x1 = coord[:,0];
    y1 = coord[:,1];
    x2 = [ coord[0][0], coord[ len(coord)-1][0] ];
    y2 = [ coord[0][1], coord[ len(coord)-1][1] ];

    ax.add_line( Line2D(x1, y1) )
    ax.add_line( Line2D(x2, y2) )

    plt.title('T-shaped Beam Section')
    plt.ylabel('y')
    plt.xlabel('x')
    plt.ylim(  -6, 10)
    plt.xlim( -10, 14)
    plt.grid(True)
    plt.show()

    print("--- ======================================== ... ");
    print("--- Enter TestPolygonTshapedSection01.main()  ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

Program Output: The abbreviated textual output is:

```
--- Enter TestPolygonTshapedBeam01.main()  ...
--- ======================================= ...
--- Part 1: Initialize t-beam coordinates ...

Matrix: T-beam Coordinates
 -8.0000   8.0000
 12.0000   8.0000
 12.0000   7.0000
  4.0000   6.0000
  3.0000  -4.0000
  1.0000  -4.0000
  0.0000   6.0000
 -8.0000   7.0000

---
--- Part 2: Max/min coordinate positions ...
---
--- Min x =   -8.000 ...
--- Max x =   12.000 ...
--- Min y =   -4.000 ...
--- Max y =    8.000 ...
---
--- Part 3: Compute and print distance of coords from origin ...
---
--- Min distance from origin =    4.123 ...
--- Max distance from origin =   14.422 ...
---
--- Part 4: Compute and print perimeter and area of polygon ...
---
--- Section perimeter =   60.224 ...
--- Section area      =   62.000 ...
---
--- Part 5: Plot section verticies and edges ...
---
--- ======================================== ...
--- Enter TestPolygonTshapedSection01.main()  ...
```

Graphical output: see Figure 3.

Question 5: 10 points. Write a Python program that will compute and print a list of (x, y) pairs for:

$$y(x) = \left[\frac{(x^3 - 16x)}{(x-4)(x+5)\sin(x)} \right] \tag{8}$$

over the range $-10 \leq x \leq 10$ and in intervals of 0.25. You should find that $y(0)$ and $y(4)$ evaluate to not-a-number (NaN), and that $y(-5)$ evaluates to positive infinity.

Python 3 provides remarkably good builtin support for handling of run-time errors. Create a plot of $y(x)$ vs x – you should find that errors will be automatically handled within the matplotlib.pyplot environment.

Python Source Code:

```
# ============================================================
# TestDifficultFunction04.py: Compute difficult function and see
# how Python handles divide by zero and NaN at runtime.
#
# Written by: Mark Austin                              January 2025
# ============================================================

import math
import numpy as np
import matplotlib.pyplot as plt

# Implement test function, ignore difficulties ...

def testFunction(x):
    result = (x*x*x - 16*x)/(x-4)/(x+5)/math.sin(x)
    return result

# main method ...

def main():
    print("--- Enter TestDifficultFunction04.main()     ... ");
    print("--- ======================================== ... ");
    print("");

    print("==============================");
    print("          Coord          Value");
    print("            (x)            (y)");
    print("==============================");

    # Define problem parameters.

    xcoord = np.linspace(-10,10,81)

    # Create list for y coordinates ...

    ycoord = [];
```

```
    # Traverse xcoord array and compute y values ...

    for x in xcoord:
        result = testFunction(x)
        print("        {:7.2f}   {:12.3f}".format(x,result) );
        ycoord.append(result)

    print("==============================");
    print("");

    # Plot y vs x for test function ...

    plt.plot(xcoord,ycoord)
    plt.title('plot y vs x for difficult function')
    plt.ylabel('y')
    plt.xlabel('x')
    plt.xlim( -10, 10)
    plt.grid(True)
    plt.show()

    print("--- ======================================== ... ");
    print("--- Leave TestDifficultFunction04.main()     ... ");

# call the main method ...

if __name__ == "__main__":
   main()
```

Program Output: The abbreviated textual output is:

```
--- Enter TestDifficultFunction04.main()     ...
--- ======================================== ...

==============================
          Coord          Value
            (x)            (y)
==============================
         -10.00        -22.058
          -9.75        -36.939
          -9.50       -154.503

... lines of output removed ..

          -5.50        -23.386
          -5.25        -30.561
          -5.00            inf <--- divide by zero ...
          -4.75         14.260
          -4.50          4.603

... lines of output removed ..

          -0.50          0.811
          -0.25          0.798
```

```
            0.00            nan <-- Not a number ...
            0.25          0.818
            0.50          0.853

... lines of output removed ..

            3.50         -8.804
            3.75         -5.811
            4.00            nan <-- Not a number ...
            4.25         -4.235
            4.50         -4.119

... lines of output removed ..

            9.50       -117.694
            9.75        -28.446
           10.00        -17.156
==============================

--- ======================================== ...
--- Leave TestDifficultFunction04.main()     ...
```

The graphical output is: