Solutions to Homework 1

Question 1: 10 points. The four fours puzzle is most commonly formulated as follows: Given no more than 4 instances of the digit "4," represent all integers using a finite number of mathematical symbols and operators in common use. Acceptable symbols and operators include:

```
+,-      addition subtraction
*,/      multiplication, division
sqrt,^   square root, power
!        factorial, eg: n! = n * (n-1) * (n-2) * ... * 2 * 1
.        decimal point, e.g., .4
.4'      repeating decimal, eg: .4' = .444444444...
```

There are, however, a few constraints – you cannot use representations for other numbers (e.g., use of π as in $sin(\pi/4) = 1$) or an infinite number of operations (e.g., sqrt(sqrt(sqrt(...4))) = 1).

It can be shown that all integers from 0 through 100 can be represented in this way. Some expressions are far from obvious, for example,

$$\text{the integer 73 corresponds to} \qquad \sqrt{\sqrt{\sqrt{4^{4!}}} + \frac{4}{.4}}. \tag{1}$$

Solutions to the first few integers are shown in Table 1. Notice that these expressions are not unique. For example, zero and one can also be expressed as (4-4)/(4+4) and (4+4)/(4+4), respectively.

Write a Python program to evaluate and print the arithmetic expressions for the four-fours problem for integers 0 through 19 and 73. The abbreviated output might look like:

```
--- Value  Expression                    Evaluation ...
--- ============================================== ...
---     0: 4/4 - 4/4                 -->   0.000000 ...
---     1: 44/44                     -->   1.000000 ...

... lines of output removed ...

--- ============================================== ...
```

Number	Expression	Number	Expression
0	$4/4 - 4/4$	10	$(44 - 4)/4$
1	$44/44$	11	$4/.4 + 4/4$
2	$4/4 + 4/4$	12	$4! - (4 + 4) - 4$
3	$\sqrt{4 * 4} - 4/4$	13	$(4!\sqrt{4} + 4)/4$
4	$4! - (4 * 4) - 4$	14	$4/.4 + \sqrt{4} + \sqrt{4}$
5	$\sqrt{4 * 4} + 4/4$	15	$(4 * 4) - 4/4$
6	$(4 + 4)/4 + 4$	16	$(4 + 4 + 4 + 4)$
7	$4!/(4 + 4) + 4$	17	$(4 * 4) + 4/4$
8	$(4 + 4) * 4/4$	18	$44 * .4 + .4$
9	$(4 + 4) + (4/4)$	19	$4! - 4 - 4/4$

Table 1: Solutions to the four fours problem

Hint: Python has a factorial function ... so you can write `math.factorial(4)` in place of 4!. Also, while the expressions in Table 1 are written in terms of integers, in practice you will need to work with double precision numbers (e.g., to handle 4/.4). You should expect tiny errors because the ensuing arithmetic calculations may not be exact.

Python Source Code:

```
# ====================================================================
# TestFourFours.py: Compute four-fours expressions for 0 -- 19 and 73.
#
# Written by: Mark Austin                                September 2025
# ====================================================================

import math

# main method ...

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

    print("--- ");
    print("--- Value  Expression                    Evaluation ... ");
    print("--- ================================================ ... ");

    # Four-fours values and expressions for 0 through 19 ...
```

```
value      = 4/4 - 4/4;
expression = "4/4 - 4/4";
print("---     0: {:25s} --> {:10.6f} ...".format(expression, value));

value      = 44/44;
expression = "44/44";
print("---     1: {:25s} --> {:10.6f} ...".format(expression, value));

value      = 4/4 + 4/4;
expression = "4/4 + 4/4";
print("---     2: {:25s} --> {:10.6f} ...".format(expression, value));

value      = math.sqrt(4*4) - 4/4;
expression = "sqrt(4*4) - 4/4";
print("---     3: {:25s} --> {:10.6f} ...".format(expression, value));

value      = math.factorial(4) - 4*4 - 4;
expression = "4! - 4*4 - 4";
print("---     4: {:25s} --> {:10.6f} ...".format(expression, value));

value      = math.sqrt(4*4) + 4/4;
expression = "sqrt(4*4) + 4/4";
print("---     5: {:25s} --> {:10.6f} ...".format(expression, value));

value      = (4+4)/4 + 4;
expression = "(4+4)/4 + 4";
print("---     6: {:25s} --> {:10.6f} ...".format(expression, value));

value = math.factorial(4)/(4+4) + 4;
expression = "4!/(4+4) + 4";
print("---     7: {:25s} --> {:10.6f} ...".format(expression, value));

value = (4+4)*(4/4);
expression = "(4+4)*(4/4)";
print("---     8: {:25s} --> {:10.6f} ...".format(expression, value));

value = 4 + 4 + 4/4;
expression = "4 + 4 + 4/4";
print("---     9: {:25s} --> {:10.6f} ...".format(expression, value));

value = (44 - 4)/4;
expression = "(44 - 4)/4";
print("---    10: {:25s} --> {:10.6f} ...".format(expression, value));

value = 4/.4 + 4/4;
expression = "4/.4 + 4/4";
print("---    11: {:25s} --> {:10.6f} ...".format(expression, value));

value = math.factorial(4) - 4 - 4 - 4;
expression = "4! - 4 - 4 - 4";
print("---    12: {:25s} --> {:10.6f} ...".format(expression, value));

value = (math.factorial(4)*math.sqrt(4) + 4)/4;
expression = "(4!*sqrt(4) + 4)/4";
print("---    13: {:25s} --> {:10.6f} ...".format(expression, value));
```

```
    value = (4/.4  + math.sqrt(4) + math.sqrt(4));
    expression = "4/.4 + sqrt(4) + sqrt(4)";
    print("---    14: {:25s} --> {:10.6f} ...".format(expression, value));

    value = 4*4 - 4/4;
    expression = "4*4 - 4/4";
    print("---    15: {:25s} --> {:10.6f} ...".format(expression, value));

    value = 4 + 4 + 4 + 4;
    expression = "4 + 4 + 4 + 4";
    print("---    16: {:25s} --> {:10.6f} ...".format(expression, value));

    value = 4*4 + 4/4;
    expression = "4*4 + 4/4";
    print("---    17: {:25s} --> {:10.6f} ...".format(expression, value));

    value = 44*.4 + .4;
    expression = "44*.4 + .4";
    print("---    18: {:25s} --> {:10.6f} ...".format(expression, value));

    value = math.factorial(4) - 4 - 4/4;
    expression = "4! - 4 - 4/4";
    print("---    19: {:25s} --> {:10.6f} ...".format(expression, value));

    # Four-fours expression for 73 ...

    value = math.sqrt( math.sqrt( math.sqrt( math.pow( 4.0, math.factorial(4)) ))) + (4.0/(4.0/9.0));
    expression = "sqrt(sqrt(sqrt(4^4!)) ) + 4/(4.0/9.0)";

    print("---");
    print("---    73: {:25s} --> {:10.6f} ...".format(expression, value));

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

# call the main method ...

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

Program Output:

```
---
--- Value  Expression                    Evaluation ...
--- ============================================== ...
---     0: 4/4 - 4/4                 -->   0.000000 ...
---     1: 44/44                     -->   1.000000 ...
---     2: 4/4 + 4/4                 -->   2.000000 ...
---     3: sqrt(4*4) - 4/4           -->   3.000000 ...
---     4: 4! - 4*4 - 4              -->   4.000000 ...
---     5: sqrt(4*4) + 4/4           -->   5.000000 ...
```

```
---     6: (4+4)/4 + 4                -->   6.000000 ...
---     7: 4!/(4+4) + 4               -->   7.000000 ...
---     8: (4+4)*(4/4)                -->   8.000000 ...
---     9: 4 + 4 + 4/4                -->   9.000000 ...
---    10: (44 - 4)/4                 -->  10.000000 ...
---    11: 4/.4 + 4/4                 -->  11.000000 ...
---    12: 4! - 4 - 4 - 4             -->  12.000000 ...
---    13: (4!*sqrt(4) + 4)/4         -->  13.000000 ...
---    14: 4/.4 + sqrt(4) + sqrt(4)   -->  14.000000 ...
---    15: 4*4 - 4/4                  -->  15.000000 ...
---    16: 4 + 4 + 4 + 4              -->  16.000000 ...
---    17: 4*4 + 4/4                  -->  17.000000 ...
---    18: 44*.4 + .4                 -->  18.000000 ...
---    19: 4! - 4 - 4/4               -->  19.000000 ...
---
---    73: sqrt(sqrt(sqrt(4^4!)) ) + 4/(4.0/9.0) -->  73.000000 ...
---
--- ============================================== ...
```

Question 2: 10 points. Write a Python program that solves for all positive integer pairs, i.e., $x, y \geq 0$,

$$x + xy + y = 2025. \tag{2}$$

You can solve this problem by simply looping over all potentially good (x,y) values and evaluating equation 2. However, if you factor the left-hand side of equation 2, and then look at the prime factors of the resulting equation, it should be evident there are only a small number of (x,y) pairs that can work. Print your results in a tidy table.

Hint: 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.

Theoretical Considerations: Equation 2 can be factorized by adding one to both sides, i.e.,

$$x + xy + y + 1 = (x + 1)(y + 1) = 2025 + 1 = 2026. \tag{3}$$

The prime factors of 2026 are 2 and 1013. Hence the only solutions to equation 2 are (0, 2025), (1, 1012), (1012, 1) and (2025, 0).

Python Source Code:

```
# ====================================================================
# TestIntegerSolutions2025b.py: Compute positive integer solutions to:
#
#      x + xy + y = 2025.
#
# where x and y are integers greater than or equal to zero.
#
# Note: adding 1 to both sides of the equation gives:
#
#       x + xy + y + 1 = (x+1)(y+1) = 2025 + 1 = 2026.
#
# The prime factorization of 2026 is 1*2*1013. Hence, we only expect
# four solutions.
#
# Written by: Mark Austin                                September, 2025
# ====================================================================

import math
from prettytable import PrettyTable

# main method ...

def main():
```

```
    print("--- Enter TestIntegerSolutions2025.main()    ... ");
    print("--- ======================================== ... ");

    n = 2025

    print("--- Part 1: Lists of solutions ... ")

    xc   = [];
    yc   = [];
    soln = [];

    print("--- Part 2: Nested loops, naive test for equality ...")
    print("")

    i = 1
    for x in range(0, n+1):
        for y in range(0, n+1):
            error = x + x*y + y - 2025;
            if error == 0:
               print("--- soln {:2d}: x = {:6d}, y = {:6d}: solution !! ...".format(i,x,y) );

               xc.append( x );
               yc.append( y );
               soln.append( (1+x)*(1+y) );

               i = i + 1

    print("")
    print("--- Part 3: Print table ...")
    print("")

    table01 = PrettyTable();
    table01.add_column("x", xc )
    table01.add_column("y", yc )
    table01.add_column("(x+1)(y+1)", soln )

    print(table01)

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

# call the main method ...

main()
```

Program Output: The textual output is:

```
--- Part 1: Lists of solutions ...
--- Part 2: Nested loops, naive test for equality ...

--- soln  1: x =      0, y =   2025: solution !! ...
--- soln  2: x =      1, y =   1012: solution !! ...
--- soln  3: x =   1012, y =      1: solution !! ...
```

```
--- soln  4: x =   2025, y =      0: solution !! ...

--- Part 3: Print table ...

+------+------+------------+
|  x   |  y   | (x+1)(y+1) |
+------+------+------------+
|  0   | 2025 |    2026    |
|  1   | 1012 |    2026    |
| 1012 |  1   |    2026    |
| 2025 |  0   |    2026    |
+------+------+------------+
```

Question 3: 10 points. Suppose that during squally conditions, regular one second wind gusts produce a forward thrust on a yacht sail corresponding to

$$F(t) = \begin{cases} 10 + 25 \cdot t - 15 \cdot t^3 & 0.0 \leq t \leq 0.4, \\ (900 - 150t)\,/100 & 0.4 < t \leq 1.0 \end{cases} \tag{4}$$

F(t) has units kN.

Write a Python program that computes and prints $F(t)$ for $0 \leq t \leq 4$ seconds, and then creates a line plot of the wind force vs time.

The textual output should look something like:

```
           Time         Thrust
          (sec)           (kN)
==============================
           0.00          10.00
           0.25          16.02
           0.50           8.25
           0.75           7.88
           1.00          10.00

... lines of output removed ...

           3.00          10.00
           3.25          16.02
           3.50           8.25
           3.75           7.88
           4.00          10.00
==============================
```

Hint: Equation 4 is periodic. For values of time, t, greater than one second, you can compute the fractional part of the time with: t - math.floor(t).

Python Source Code:

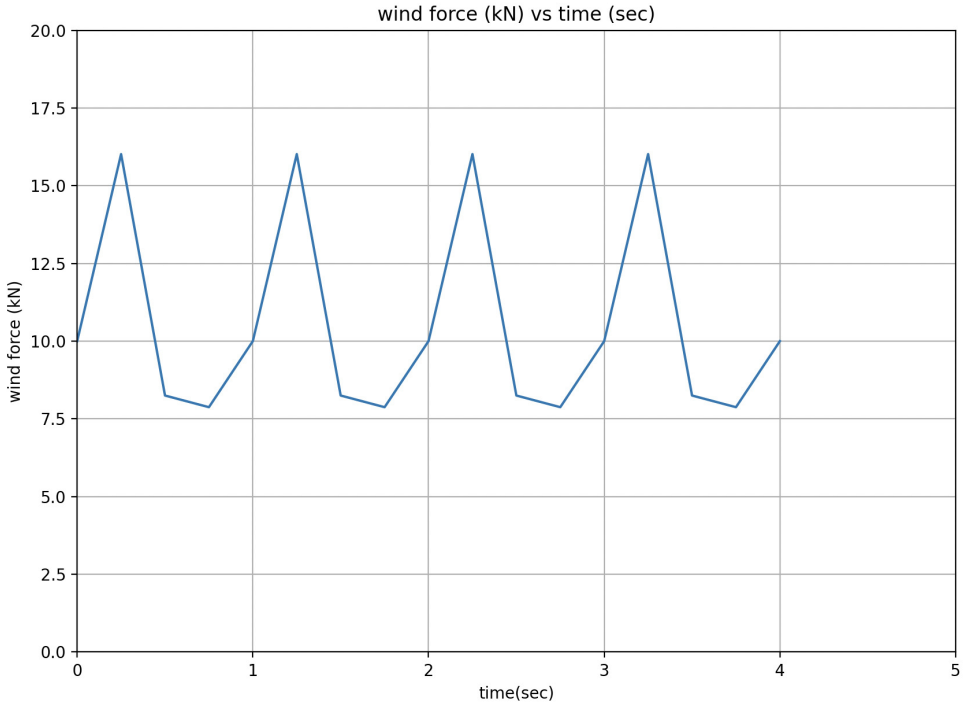
```
# ============================================================
# TestWindForceYachtSail02.py: Compute wind force on yacht sail.
#
# Written by: Mark Austin                          September 2025
# ============================================================

import math
import matplotlib.pyplot as plt

# Compute wind force on yacht sail.
```

```
def windforce(time):
    rt = time - math.floor(time)

    if rt <= 0.4:
       thrust = 10 + 25*rt - 15*rt**3;
    else:
       thrust = (900 - 150*(rt))/100;

    return thrust

# main method ...

def main():
    print("--- Enter TestWindForceYachtSail02.main()    ... ");
    print("--- ======================================== ... ");
    print("");

    print("           Time         Thrust");
    print("          (sec)           (kN)");
    print("==============================");

    # Define problem parameters.

    dt = 0.25;

    # Create empty lists for (time, force) pairs ...

    time  = [];
    force = [];

    # Main loop for windforce computation ...

    i = 0
    while(i < 17):
      dtime  = i*dt;             # <-- compute time.
      dforce = windforce(dtime)  # <-- wind force on yacht sail.

      time.append(dtime);
      force.append(dforce);

      print("  {:13.2f}  {:13.2f}".format(dtime, dforce) );
      i = i + 1;

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

    # Plot windforce vs time ...

    plt.plot(time,force)
    plt.title('wind force (kN) vs time (sec)')
    plt.ylabel('wind force (kN)')
    plt.xlabel('time(sec)')
    plt.ylim( 0, 20)
    plt.xlim( 0, 5)
```

```
    plt.grid(True)
    plt.show()

    print("--- ======================================== ... ");
    print("--- Leave TestWindForceYachtSail02.main()    ... ");

# call the main method ...

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

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

The graphical output is:



Question 4: 10 points. Figure 1 shows a two-dimensional grid of masses. If the total number of point masses is denoted by N, then the total mass of the grid, M, is given by

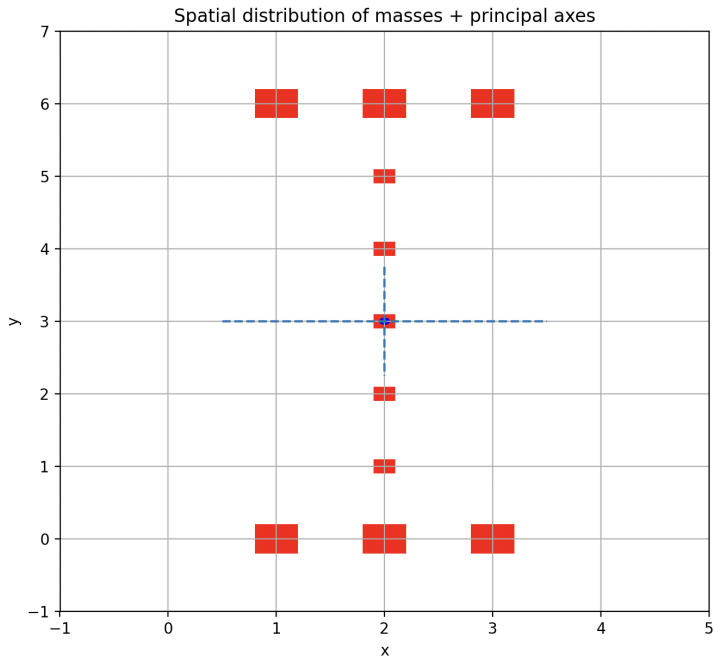


Figure 1: I-shaped grid of masses + principal axes.

$$M = \sum_{i=1}^{N} m_i \tag{5}$$

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

$$M\bar{x} = \sum_{i=1}^{N} x_i \cdot m_i \quad \text{and} \quad M\bar{y} = \sum_{i=1}^{N} y_i \cdot m_i \tag{6}$$

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

$$I_{xx} = \sum_{i=1}^{N} y_i^2 \cdot m_i \quad \text{and} \quad I_{yy} = \sum_{i=1}^{N} x_i^2 \cdot m_i \tag{7}$$

respectively. Similarly the cross moment of inertia is given by

$$I_{xy} = \sum_{i=1}^{N} x_i \cdot y_i \cdot m_i \tag{8}$$

With solutions to equations 6 - 8 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

$$\tan(2\theta) = \left[\frac{2I_{xy}}{I_{xx} - I_{yy}} \right] \tag{9}$$

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

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

    coord = np.array( [ ( 1.0, 0.0 ), ( 2.0, 0.0 ), ( 3.0, 0.0 ),
                        ( 2.0, 1.0 ), ( 2.0, 2.0 ), ( 2.0, 3.0 ),
                        ( 2.0, 4.0 ), ( 2.0, 5.0 ), ( 2.0, 6.0 ),
                        ( 1.0, 6.0 ), ( 3.0, 6.0 ) ] );
```

Write a Python program to evaluate equations 5 – 9, and create a plot in Python similar to Figure 1. 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                              August 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( [ 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0 ])
coord = np.array( [ ( 1.0, 0.0 ), ( 2.0, 0.0 ), ( 3.0, 0.0 ),
                    ( 2.0, 1.0 ), ( 2.0, 2.0 ), ( 2.0, 3.0 ),
                    ( 2.0, 4.0 ), ( 2.0, 5.0 ), ( 2.0, 6.0 ),
                    ( 1.0, 6.0 ), ( 3.0, 6.0 ) ] );

# 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, 7)
    plt.xlim( -1, 5)
    plt.grid(True)
    plt.show()

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

# call the main method ...

main()
```

Program Output: The abbreviated textual output is:

```
--- Ixx (about axis)   = 271.00  ...
--- Iyy (about axis)   = 76.00  ...
--- Ixy (about origin) = 102.00  ...
--- Total mass = 17.00  ...
--- Centroid (x,y) = (2.00, 3.00)  ...
--- Ixx (about centroid) = 118.00  ...
--- Iyy (about centroid) = 8.00  ...
--- Ixy (about centroid) = 0.00  ...
--- Mohr's circle: angle = 0.000 radians ...
```

Graphical output: see Figure 1.

Question 5: 10 points. Figure 2 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.

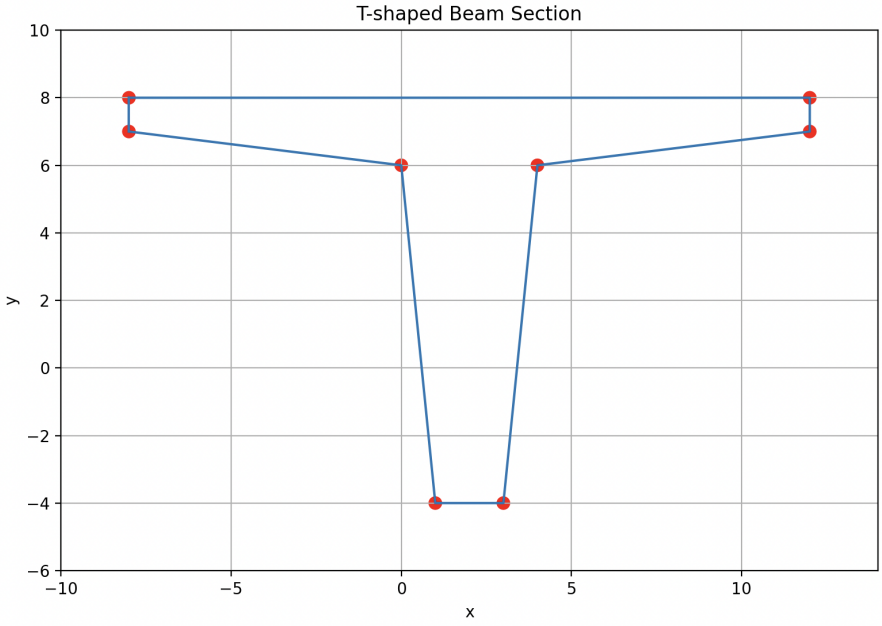


Figure 2: T-shaped beam cross section.

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 2 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 2.
4. Write functions perimeter() and area() to compute the perimeter and area of the T-beam, respectively.

Hints. For Parts 1 and 2, use the `max()` and `min()` methods in Python. One way of creating Figure 2 is to draw the vertices as circle objects (i.e., from matplotlib.patches import Circle) and the edges as objects of type Line2D (i.e., from matplotlib.lines import Line2D). To compute the perimeter and area of the T-beam, use the fact that the vertices have been specified in a clockwise manner. You should be able to systematically walk around the perimeter of the T-beam and compute the required values of interest.

Python Source Code:

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

```
--- 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 ...
---
```

Graphical output: see Figure 2.

Question 6: 10 points. Write a Python program that will compute and print a list of (x, y) pairs for:

$$y(x) = \left[\frac{(x^3 - 25x)}{(x-4)(x+5)\sin(x)} \right] \tag{10}$$

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

```
# ============================================================
# TestDifficultFunction05.py: Compute difficult function and see
# how Python handles divide by zero and NaN at runtime.
#
# Written by: Mark Austin                              September 2025
# ============================================================

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

# Implement test function, ignore difficulties ...

def testFunction(x):
    result = (x*x*x - 25*x)/(x-4)/(x+5)/math.sin(x)
    return result

# main method ...

def main():
    print("--- Enter TestDifficultFunction05.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 TestDifficultFunction05.main()     ... ");

# call the main method ...

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

Program Output: The abbreviated textual output is:

```
==============================
          Coord          Value
            (x)            (y)
==============================
         -10.00        -19.695
          -9.75        -32.734
          -9.50       -135.776

... lines of output removed ...

          -5.75        -12.473
          -5.50         -8.616
          -5.25         -6.773
          -5.00            nan
          -4.75         -5.297
          -4.50         -5.145
          -4.25         -5.324

... lines of output removed ...

          -0.75          1.332
          -0.50          1.275
```

```
          -0.25          1.248
           0.00            nan
           0.25          1.280
           0.50          1.341
           0.75          1.439

... lines of output removed ...

           3.25        -70.089
           3.50        -29.933
           3.75        -32.805
           4.00            inf
           4.25         14.246
           4.50          4.603
           4.75          1.584
           5.00         -0.000
           5.25         -1.222

... lines of output removed ...

           9.50       -103.428
           9.75        -25.208
          10.00        -15.318
==============================
```

The graphical output is: