

ENCE 201 Midterm 1, Open Notes and Open Book

Name : _____

E-mail (print neatly!): _____

Exam Format and Grading. This take home midterm exam is open notes and open book. You need to comply with the **university regulations for academic integrity**.

A few questions ask for diagrams showing the layout of memory generated by Python computations – feel free to use Google for inspiration/ideas. Partial credit will be given for partially correct answers, so please show all your working.

Please see the **class web page for instructions on how to submit your exam paper**.

Question	Points	Score
1	20	
2	20	
Total	40	

Question 1: 20 points. This question covers use of Python to model and visualize the difficult equation:

$$f(x) = \frac{1}{x^5} - \frac{2}{x^4} + \frac{4}{x^3} - \frac{8}{x^2} + \frac{16}{x} - 32 \quad (1)$$

Figure 1 plots $f(x)$ over the range $[-4, 4]$.

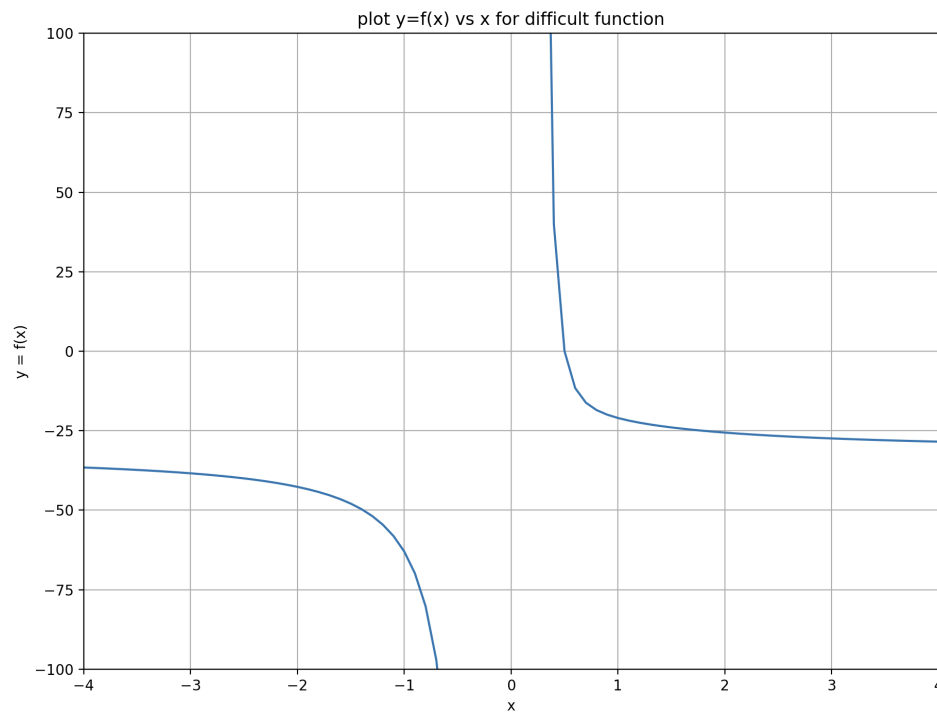


Figure 1. Plot $y = f(x)$ vs x .

From the plot we can see that there is one solution to $f(x) = 0$ within the interval $[0, 1]$. Also notice that when $x = 0$, evaluation of $f(x)$ will result in positive/negative infinity, a run-time error.

The Python code to model equation 1 and produce Figure 1 is as follows:

```
# =====
# TestDifficultFunction04.py: Explore roots to difficult equation.
#
# Written by: Mark Austin                                October 2025
# =====

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

```

# Difficult function ...

def difficultFunction(x):
    result = 1/x**5 - 2/x**4 + 4/x**3 - 8/x**2 + 16.0/x - 32.0;
    return result

# main method ...

def main():
    print("=====");
    print("          Coord          Value");
    print("          (x)             (y)");
    print("=====");

    # Define problem parameters.

    xcoord = np.linspace( -4, 4, 81)

    # Create list for y coordinates ...

    ycoord = [];

    # Traverse xcoord array and compute y values ...

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

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

    # Plot y=f(x) vs x for test function ...

    plt.plot(xcoord,ycoord)
    plt.title('plot y=f(x) vs x for difficult function')
    plt.ylabel('y = f(x)')
    plt.xlabel('x')
    plt.xlim( -4.0, 4.0 )
    plt.ylim( -100.0, 100.0 )
    plt.grid(True)
    plt.show()

# call the main method ...

if __name__ == "__main__":
    main()

```

The abbreviated textual output is:

```

=====
          Coord          Value

```

```

          (x)          (y)
=====
      -4.00      -3.657e+01
      -3.90      -3.671e+01
      -3.80      -3.685e+01

... lines of code removed ...

      -0.10      -1.250e+05
       0.00              nan
       0.10       8.333e+04
       0.20       2.223e+03
       0.30       2.452e+02
       0.40       4.003e+01
       0.50       0.000e+00
       0.60      -1.161e+01

... lines of code removed ...

       3.80      -2.828e+01
       3.90      -2.836e+01
       4.00      -2.844e+01
=====

```

Let's start with a theoretical evaluation of the roots to equation 1.

[1a] (7 pts) Use mathematics to show that equation 1 has **only one real root**.

Show all of your working.

Now look the Python code over carefully and answer the following questions.

[1b] (2 pts) Explain the purpose of the statements:

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

and how they are used in the program.

[1c] (2 pts) The fragment of textual output:

```
-0.10      -1.250e+05
0.00              nan
0.10      8.333e+04
```

indicates an error in the evaluation for $f(x)$ when $x = 0$. What is the nature of this error, and what is the standard that prevents Python from crashing?

[1d] (3 pts) Draw and label a figure for the coordinate values generated by:

```
xcoord = np.linspace( -4, 4, 81)
```

[1e] (2 pts) Briefly explain how the formatting specification for x and $result$

```
print("          {:7.2f}    {:12.3e}".format( x, result) );
```

can be re-written to take advantage of Python f-strings.

[1f] (4 pts) Now consider the block of code:

```
ycoord = [];  
for x in xcoord:  
    result = difficultFunction(x)  
    ycoord.append(result)
```

Draw and label the layout of memory for `ycoord` as it works its way through this looping structure. To keep things manageable, just show an abbreviation of what happens, focusing on the beginning and end of the list assembly. All reasonable answers will be accepted.

Question 2 (20 points). This question covers use of Python to systematically assemble and draw the box girder bridge cross section shown in Figure 2, and then compute and print the polygon area.

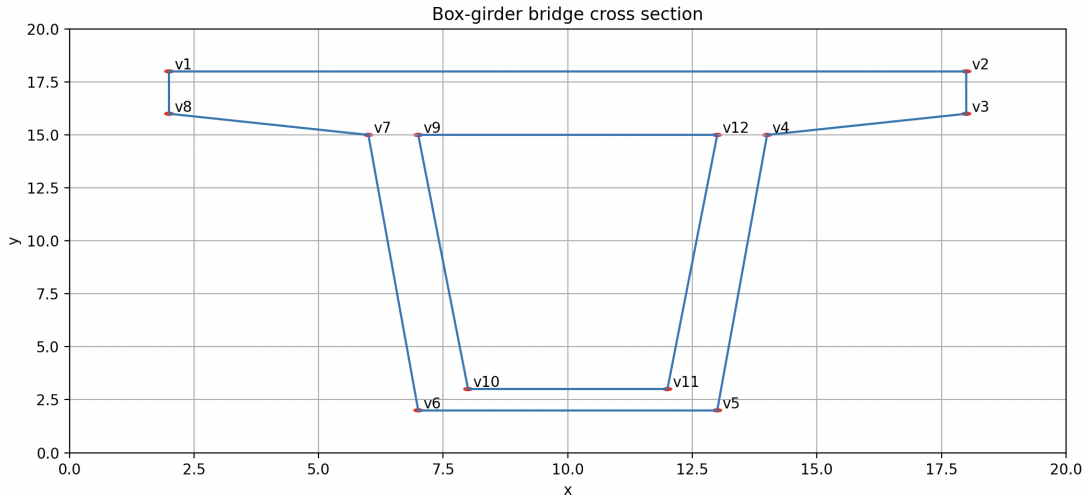


Figure 2. Cross section of box-girder bridge.

In geometry, a polygon with holes is an area-connected planar polygon with one exterior boundary and one or more interior boundaries (holes). An appropriate arrangement of Python scripts and classes for the test program and polygon modeling is shown in Figure 3.

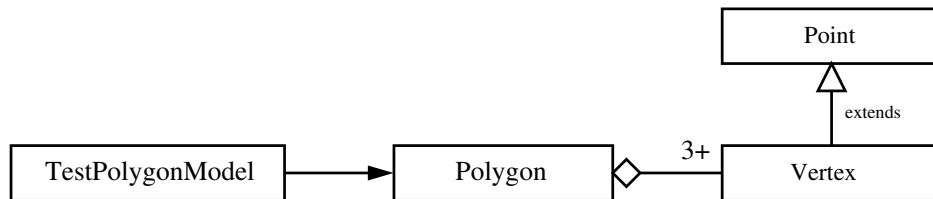


Figure 3. Test program script and classes in a polygon-with-holes system.

Our specification for polygons with holes will be assembled from lists of vertex objects, one list for the exterior boundary, and individual lists for the boundary of each hole. Each vertex will be modeled as a point (data attributes: x,y) with a label (data attribute: name). The individual lists for the boundary of each hole will be stored in a dictionary.

Finally, notice that vertices of the exterior boundary (i.e., $v1, v2, v3 \dots v8$) are organized clockwise, and the vertices of the holes (i.e., $v9, v10 \dots v12$) are organized anti-clockwise. This modeling convention simplifies the design of algorithms to compute engineering properties, such as the polygon area.

Python Code: The program source code is comprised of four Python files. Look it over carefully and answer the questions that follow:

Object Source Code: Point01.py

```
# =====  
# Point01.py: Bare-bones implementation of a Point class ...  
#  
# Written by: Mark Austin                                October 2025  
# =====  
  
class Point:  
  
    def __init__(self, xCoord=0, yCoord=0):  
        self.xCoord = xCoord  
        self.yCoord = yCoord  
  
    # Get/set X coordinate  
  
    def getX(self):  
        return self.xCoord  
  
    def setX(self, xCoord):  
        self.xCoord = xCoord  
  
    # Get/set Y coordinate  
  
    def getY(self):  
        return self.yCoord  
  
    def setY(self, yCoord):  
        self.yCoord = yCoord  
  
    # Return string representation of object ...  
  
    def __str__(self):  
        return "( %6.2f, %6.2f )" % ( self.xCoord, self.yCoord )
```

Object Source Code: Vertex01.py

```
# =====  
# Vertex01.py: Bare-bones vertex ...  
# =====  
  
from Point01 import Point  
  
class Vertex(Point):  
    label = ""  
  
    # Constructor method ...  
  
    def __init__(self, x, y):  
        Point.__init__(self, x, y)  
  
    # Set/get label ...
```

```

def setLabel(self, label ):
    self.label = label

def getLabel(self):
    return self.label

# Assemble string representation of Vertex ...

def __str__(self):
    vertexinfo = [];
    vertexinfo.append("\n");
    vertexinfo.append("--- Vertex: {:s} ... \n".format( self.getLabel()));
    vertexinfo.append("----- \n");
    vertexinfo.append("--- Coordinate: (x,y) = {:s} ... \n".format( Point.__self__()));
    vertexinfo.append("----- ");
    return "".join(vertexinfo);

```

Object Source Code: PolygonModel01.py

```

# =====
# PolygonModel01.py: Bare-bones implementation of a polygon containing
# holes. Rules of implementation:
#
# -- The polygon exterior will be modeled by a sequence of nodes
#    organized into a clockwise orientation.
# -- Holes will be modeled as loop having nodes organized into an
#    anti-clockwise orientation.
#
# Written by: Mark Austin                                October 2025
# =====

from Vertex01 import Vertex

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

class Polygon:
    area      = 0
    perimeter = 0
    name      = ""
    boundary  = []      # <-- coords for polygon exterior ...
    holes     = {}      # <-- dictionary of holes (key = name, value = list of coords) ...

    # Constructor method ...

    def __init__(self, vertexlist):
        self.boundary = vertexlist;          # <--- Assign vertex list to coords ...

    # Set/get name ...

    def setName(self, name):
        self.name = name

```

```

def getName(self):
    return self.name

# Add hole to polygon model ...

def addHole(self, name, hole ):
    self.holes[ name ] = hole;

# -----
# Compute area of a loop ...
# -----

def getLoopArea( self, coord ):
    norows = len(coord);

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

    dx = coord[0].getX() - coord[norows-1].getX();
    dy = coord[0].getY() + coord[norows-1].getY();
    darea = darea + dx*dy/2.0;

    return darea;

# -----
# Polygon Area ...
# -----

def getPolygonArea(self):

    # Compute area of outer boundary polygon ...

    area = self.getLoopArea( self.boundary );

    # Compute area of holes inside polygon ...

    holeKeyList = list( self.holes.keys() )
    for hole in holeKeyList:
        area = area + self.getLoopArea( self.holes.get(hole) );

    return area;

# -----
# Draw loop of edges ...
# -----

def drawLoop(self, ax, coords ):

    # Draw polygon edges ...

    for i in range( len(coords)-1):

```

```

        xcoords = [ coords[i].getX(), coords[i+1].getX() ];
        ycoords = [ coords[i].getY(), coords[i+1].getY() ];
        ax.add_line( Line2D(xcoords, ycoords) )

lastnode = len( coords ) - 1
xcoords = [ coords[0].getX(), coords[ lastnode ].getX() ];
ycoords = [ coords[0].getY(), coords[ lastnode ].getY() ];
ax.add_line( Line2D(xcoords, ycoords) )

# Draw polygon vertices as small circles ...

width = 0.1;
for i in range(len( coords )):
    xcoord = coords[i].getX();
    ycoord = coords[i].getY();
    ax.add_patch( Circle( (xcoord, ycoord), width, facecolor='red' ) )

# Draw node labels ...

dx = 0.1; dy = 0.1
for i in range(len( coords )):
    xcoord = coords[i].getX();
    ycoord = coords[i].getY();
    label = coords[i].getLabel();
    ax.text( xcoord + dx, ycoord + dy, label )

# -----
# Draw polygon model ...
# -----

def draw(self, ax):

    # Draw exterior of polygon ...

    self.drawLoop( ax, self.boundary );

    # Draw holes inside polygon ...

    holeKeyList = list( self.holes.keys() )
    for hole in holeKeyList:
        self.drawLoop( ax, self.holes.get(hole) );

# -----
# String representation of simple polygon ...
# -----

def __str__(self):
    polygoninfo = [];
    polygoninfo.append("\n");
    polygoninfo.append("--- PolygonModel: {:s} ... \n".format( self.name ));
    polygoninfo.append("--- ----- \n");

    # String representation of polygon boundary ...

    polygoninfo.append("--- Polygon Boundary \n");

```

```

for i in range(len( self.boundary )):
    xc    = self.boundary[i].getX();
    yc    = self.boundary[i].getY();
    label = self.boundary[i].getLabel();
    polygoninfo.append("--- Vertex {:2d}: (x,y) = ({:6.2f}, {:6.2f}): label = \"{:s}\" .
                        format( i+1, xc, yc, label));

# String representation of holes ...

holeKeyList = list( self.holes.keys() )
for hole in holeKeyList:
    polygoninfo.append("--- Polygon Hole: {:s} ... \n".format( hole ));
    loop    = self.holes.get(hole);
    for i in range(len( loop )):
        xc    = loop[i].getX();
        yc    = loop[i].getY();
        label = loop[i].getLabel();
        polygoninfo.append("--- Vertex {:2d}: (x,y) = ({:6.2f}, {:6.2f}): label = \"{:s}\"
                            format( i+1, xc, yc, label));

polygoninfo.append("\n");
polygoninfo.append("--- Polygon Area = {:f} ... \n".format( self.getPolygonArea() ));
polygoninfo.append("--- ----- ");
return "".join(polygoninfo);

```

Test Program Source Code: TestBoxGirderPolygonModel01.py

```

# =====
# TestBoxGirderPolygonModel01.py: Analysis of a box-girder bridge cross section.
# =====

from Vertex01 import Vertex
from PolygonModel01 import Polygon

import matplotlib.pyplot as plt

# main method ...

def main():
    print("--- Enter TestBoxGirderPolygonModel01.main() ... ");
    print("--- ===== ... ");

    print("--- Part 1: List of vertices for polygon exterior ... ");

    v01 = Vertex ( 2.0, 18.0 ); v01.setLabel("v1");
    v02 = Vertex ( 18.0, 18.0 ); v02.setLabel("v2");
    v03 = Vertex ( 18.0, 16.0 ); v03.setLabel("v3");
    v04 = Vertex ( 14.0, 15.0 ); v04.setLabel("v4");
    v05 = Vertex ( 13.0, 2.0 ); v05.setLabel("v5");
    v06 = Vertex ( 7.0, 2.0 ); v06.setLabel("v6");
    v07 = Vertex ( 6.0, 15.0 ); v07.setLabel("v7");
    v08 = Vertex ( 2.0, 16.0 ); v08.setLabel("v8");

```

```

exteriorLoop = [ v01, v02, v03, v04, v05, v06, v07, v08 ];

print("--- Part 2: Lists of vertices for interior hole ... ");

v09 = Vertex ( 7.0, 15.0 ); v09.setLabel("v9");
v10 = Vertex ( 8.0, 3.0 ); v10.setLabel("v10");
v11 = Vertex ( 12.0, 3.0 ); v11.setLabel("v11");
v12 = Vertex ( 13.0, 15.0 ); v12.setLabel("v12");

hole01 = [ v09, v10, v11, v12 ];

print("--- Part 3: Assemble and print box-girder polygon ... ");

polygon01 = Polygon( exteriorLoop );
polygon01.setName("Box-girder cross section");
polygon01.addHole( "hole01", hole01 );

print( polygon01 )

print("--- Part 4: Draw box-girder cross section ... \n");

# Define Matplotlib figure and axis

fig, ax = plt.subplots()

polygon01.draw(ax)

plt.title('Box-girder bridge cross section')
plt.ylabel('y')
plt.xlabel('x')
plt.ylim( 0, 20 )
plt.xlim( 0, 20 )
plt.grid(True)
plt.show()

print("--- ===== ... ");
print("--- Finished TestBoxGirderPolygonModel01.main() ... ");

# call the main method ...

main()

```

Program Output: The abbreviated program output is:

```

--- Part 1: List of vertices for polygon exterior ...
--- Part 2: Lists of vertices for interior hole ...
--- Part 3: Assemble and print box-girder polygon ...

--- PolygonModel: Box-girder cross section ...
-----
--- Polygon Boundary
--- Vertex 1: (x,y) = ( 2.00, 18.00): label = "v1" ...
--- Vertex 2: (x,y) = ( 18.00, 18.00): label = "v2" ...

```

```
--- Vertex 3: (x,y) = ( 18.00, 16.00): label = "v3" ...
--- Vertex 4: (x,y) = ( 14.00, 15.00): label = "v4" ...
--- Vertex 5: (x,y) = ( 13.00,  2.00): label = "v5" ...
--- Vertex 6: (x,y) = (  7.00,  2.00): label = "v6" ...
--- Vertex 7: (x,y) = (  6.00, 15.00): label = "v7" ...
--- Vertex 8: (x,y) = (  2.00, 16.00): label = "v8" ...
--- Polygon Hole: hole01 ...
--- Vertex 1: (x,y) = (  7.00, 15.00): label = "v9" ...
--- Vertex 2: (x,y) = (  8.00,  3.00): label = "v10" ...
--- Vertex 3: (x,y) = ( 12.00,  3.00): label = "v11" ...
--- Vertex 4: (x,y) = ( 13.00, 15.00): label = "v12" ...

--- Polygon Area = 75.000000 ...
-----
--- Part 4: Draw box-girder cross section ...
```

Questions: Let's start with Vertex01.py.

[2a] (2 pts) What does the line of code:

```
from Point01 import Point
```

do, and why is it needed in this program?

[2b] (2 pts) Briefly explain how the class hierarchy relationship between Point and Vertex is established (see right-hand side of Figure 3).

[2c] (2 pts) List the methods available to the class `Vertex` via inheritance from class `Point`.

[2d] (2 pts) Briefly explain what the line:

```
vertexinfo.append("--- Coordinate: (x,y) = {:s} ... \n".format( Point.__str__( ) ));
```

is doing?

Let's move onto `TestPolygonModel01.y` and `PolygonModel01.py`:

[2e] (4 pts) Draw and label a diagram showing the layout of memory generated by the block of code:

```
v01 = Vertex ( 2.0, 18.0 ); v01.setLabel("v1");  
v02 = Vertex ( 18.0, 18.0 ); v02.setLabel("v2");  
  
... lines of source code removed ...  
  
v08 = Vertex ( 2.0, 16.0 ); v08.setLabel("v8");  
  
exteriorLoop = [ v01, v02, v03, v04, v05, v06, v07, v08 ];
```

All reasonable answers will be accepted.

[2f] (4 pts) Draw and label a diagram of the layout of memory created by the block of code:

```
polygon01 = Polygon( exteriorLoop );  
polygon01.setName("Box-girder cross section");  
polygon01.addHole( "hole01", hole01 );
```

Clearly indicate how the boundary list (i.e., `boundary = []`) and holes dictionary (i.e., `holes = {}`) are used within the Polygon object.

Hint: For ideas, Google: python dictionary layout.

Finally, let's consider the area computation for the box-girder cross section polygon (see Figure 2).

[2g] (4 pts) The method `getPolygonArea()`:

```
def getPolygonArea(self):  
    # Compute area of outer boundary polygon ...  
    area = self.getLoopArea( self.boundary );  
    # Compute area of holes inside polygon ...  
    holeKeyList = list( self.holes.keys() )  
    for hole in holeKeyList:  
        area = area + self.getLoopArea( self.holes.get(hole) );  
    return area;
```

systematically computes the cross section area by first computing the area of the outer boundary, and then adjusting the area for the inner hole.

Create a table that shows the key stages of the area computation. A reasonable table would include values for *area*, *self.boundary*, *hole*, *self.holes.get(hole)* and results from *self.getLoopArea()*.