Solutions to Midterm 1 Exam

Question 1: 20 points.

Problem Statement. This question covers use of Python to model and visualize the difficult equation:

$$f(x) = \left[x - \frac{1}{x}\right]^3 + \left[x - \frac{1}{x}\right] - 30. \tag{1}$$

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

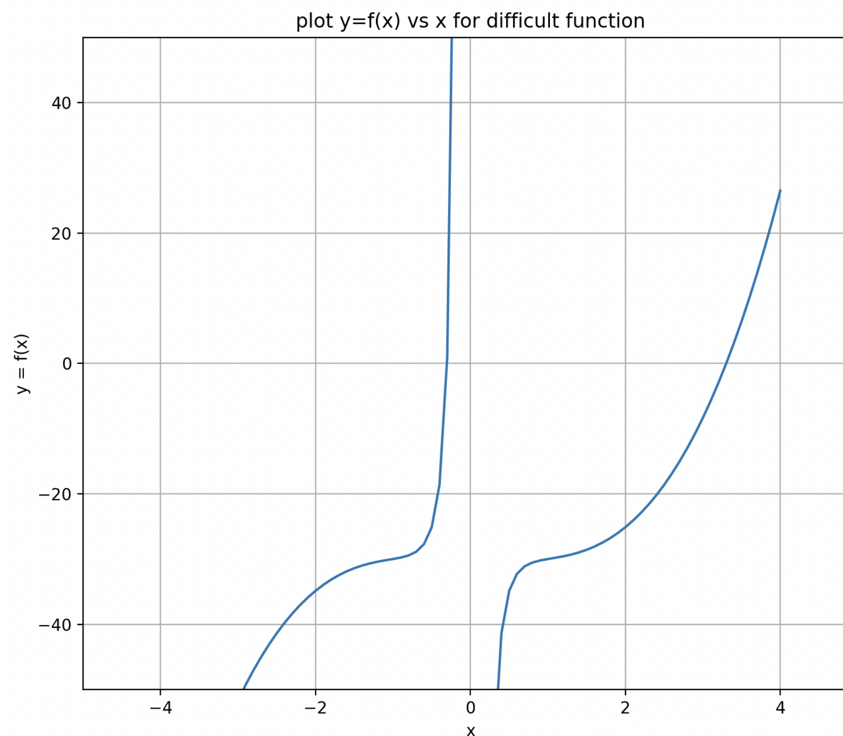

Figure 1. Plot y = f(x) vs x.

From the plot we can see that within the range $[-4, 4]$ there are two real roots, the first within the interval $[-1, 0]$, and a second within the interval $[3, 4]$. Also notice that when x = 0, evaluation of $f(0)$ will result in a run-time error.

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

```
# =================================================================
# TestDifficultFunction01.py: Explore roots to cubic equation.
#
# Written by: Mark Austin                           October 2024
# =================================================================

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

# difficult cubic function ...

def cubicFunction(x):
    result = (x - 1/x)**3 + (x - 1/x) - 30.0
    return result

# main method ...

def main():
    print("--- Enter TestDifficultFunction01.main()     ... ");
    print("--- ======================================== ... ");
    print("");

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

    # Part 1: Define problem parameters.

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

    # Part 2: Create list for y coordinates ...

    ycoord = [];

    # Part 3: Traverse xcoord and compute y values ...

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

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

    # Part 4: 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(  -5,  5)
```

```
    plt.ylim( -50, 50)
    plt.grid(True)
    plt.show()

    print("--- ======================================== ... ");
    print("--- Leave TestDifficultFunction01.main()     ... ");

# call the main method ...

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

The abbreviated textual output is:

```
==============================
          Coord          Value
              x       y = f(x)
==============================
          -4.00        -86.484
          -3.90        -82.015
          -3.80        -77.780

... lines of output removed ...

          -0.50        -25.125
          -0.40        -18.639
          -0.30          0.943
          -0.20         85.392
          -0.10        950.199
           0.00           -inf
           0.10      -1010.199
           0.20       -145.392
           0.30        -60.943

... lines of output removed ...

           3.10         -5.797
           3.20         -3.038
           3.30         -0.085
           3.40          3.067
           3.50          6.423
           3.60          9.990
           3.70         13.774
           3.80         17.780
           3.90         22.015
           4.00         26.484
==============================
```

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

Part [1a] By using the substitution u = (x - 1/x), show that an **equivalent representation** for the roots to equation is:

$$[u - 3]\left[u^2 + 3u + 10\right] = 0. \tag{2}$$

Hence, show that the two roots in Figure 1 are:

$$[r_1, r_2] = \frac{3}{2} \pm \frac{\sqrt{13}}{2} \tag{3}$$

and that there are no other real roots. Show all of your working.

Solution: The easiest way of showing equivalency is to multiply out the terms in equation 2, i.e.,

$$[u - 3]\left[u^2 + 3u + 10\right] \longrightarrow u^3 + u - 30. \tag{4}$$

Alternatively, you can simply observe that u = 3 is a root (i.e., 3^3 + 3 = 30), and hence can be factored out of equation . Polynomial division gives:

$$u^3 + u - 30 \longrightarrow [u - 3]\left[au^2 + bu + c\right] \tag{5}$$

where coefficients a, b, and c are unknown constants. Matching the cubic, quadratic, and constant terms gives a = 1, b = 3, and c = 10.

Both approaches lead to the conclusion that there are two cases to consider: (1) u = 3, and (2) $u^2 + 3u + 10$ = 0. The first case says:

$$x - \frac{1}{x} = 3, \tag{6}$$

which is equivalent to $x^2 - 3x - 1 = 0$, and the two roots shown in equation 3.

In the second case, the descriminant of $u^2 + 3u + 10$ is less than zero, meaning that the roots to equation 4 will be complex. It follows that solutions to the roots in x will also be complex.

Part [1b] 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.

Solution: In Python, the keyword import provides access to library modules and their functions and constants therein. Here we have three statements:

- `import math` accesses the math library, and it's constants (e.g., math.pi) and methods (e.g., math.cos(x)) therein. However, in this code, it is not used/necessary.
- `import numpy as np` accesses the numpy library and gives is a shortened name `np`. Numpy provides support for the creation of 0-d, 1-d, 2-d, and 3-d arrays, along with computational support for array operations. Use of the shortened name means that instead of creating an array object with something like numpy.array(2,2), we can simply write np.array(2,2).

  This program uses numpy in the line xcoord = np.linspace(-4, 4, 81), to create an array of evenly-spaced values.
- `import matplotlib.pyplot as plt` accesses the `pyplot` module of the `matplotlib` library and gives it the shortened name `plt`. This module provides support for the creation of plots/figures, and in this program, is used to plot y = f(x) (see Figure 1), label the axes, and provide a title.

Part [1c] The fragment of textual output:

```
    -0.10        950.199
     0.00           -inf
     0.10      -1010.199
```

indicates an error in the evaluation for f(x) when x = 0. What is the nature of this error, and why doesn't Python crash?

Solution: When x = 0, f(x) encounters a divide by zero in the expression 1/x. Python records the divide-by-zero as an `Inf` run-time error.

Part [1d] What is the purpose of the statement:

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

Solution: This statement creates an array of 80 intervals of equal length between -4 and 4 (there are 81 nodes).

The `xcoord` array values are passed to the cubic function to get y = f(x), then to create a plot and print the x and f(x) values.

Part [1e] Briefly explain how the formatting specification for x and $result$ works in:

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

Solution: The statement prints two floating point-values, `x` and `result`, with a one-to-one match of `x` with the formatting specification $\{: 7.2f\}$, and `result` with $\{: 12.3f\}$. $\{: 7.2f\}$ tells Python to print `x` using up to 7 characters, and two decimal places of accuracy to the right of the decimal point. Similarly, $\{: 7.2f\}$ specifies a floating-point value printed 12 characters wide, and three decimal places of accuracy to the right-hand side of the decimal point.

Part [1f] Briefly explain the purpose of the statements:

```
ycoord = [];

for x in xcoord:
    result = cubicFunction(x)
    ycoord.append(result)
```

Solution: The statement `ycoord = []` creates an empty list. The for loop indicates that for each `x` value in `xcoord`, evaluate `cubicFunction(x)` and and then add the result to the `ycoord = []` list. At the conclusion of the for loop, both the `xcoord` array and `ycoord` list will contain 81 float-point values.

Question 2 (20 points)

Problem Statement. A polygon is said to simple if it has one exterior boundary and no internal holes. Figure 2 shows, for example, a simple polygon that has six sides and an irregular shape.

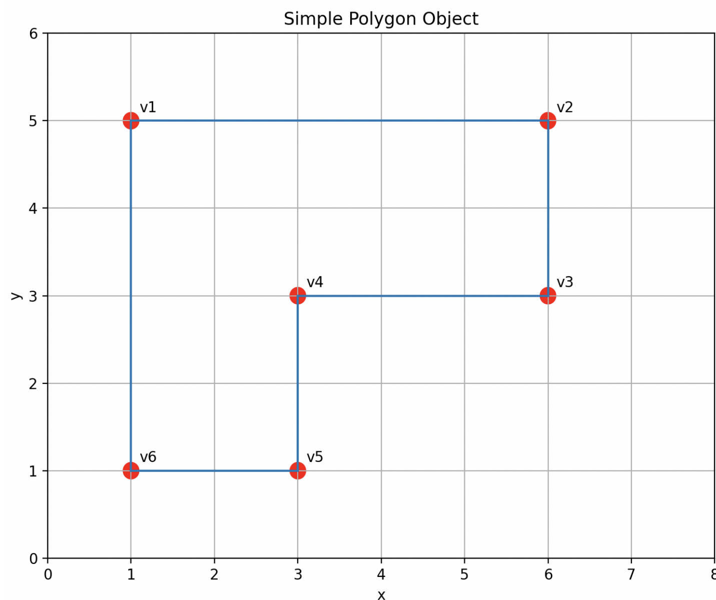

Figure 2. An irregular six-sided simple polygon.

This question covers use of Python to systematically assemble and draw the polygon shown in Figure 2, and then compute the polygon perimeter.

Instead of using a two-dimensional numpy array to handle coordinates $(x_1, y_1) \cdots (x_n, y_n)$, our specification for simple polygons will be assembled from a list of vertex objects. Each vertex will be modeled as a point (data attributes: x,y) with a label (data attribute: name). An appropriate arrangement of TestSimplePolygon, SimplePolygon, Vertex, and Point classes is as follows:

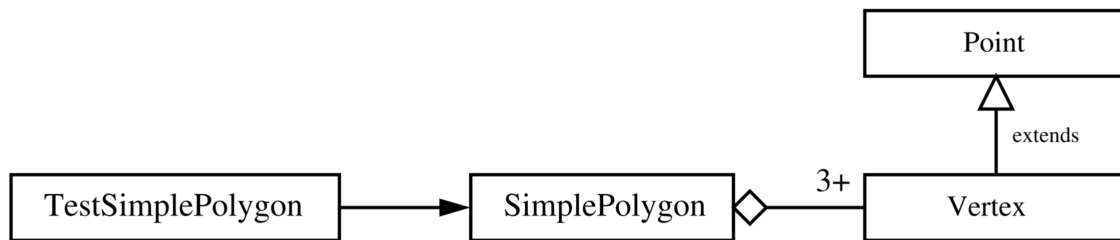

Figure 3. Test program script and classes in a simple polygon system.

The right-hand side of Figure 3 says: a valid simple polygon has three or more vertices.

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, 2024
# ==========================================================

import math

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

    # Get current position

    def get_position(self):
        return self.__xCoord, self.__yCoord

    # Compute distance between two points ...

    def distance(self, second):
        x_d = self.xCoord - second.xCoord
        y_d = self.yCoord - second.yCoord
        return (x_d**2 + y_d**2)**0.5

    # Return string represention 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)
     self.label = ""

   # Set/get label ...

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

   def getLabel(self):
     return self.label

   # Assemble string represention 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: SimplePolygon01.py

```
# ===================================================================
# SimplePolygon01.py: Bare-bones implementation of a simple polygon.
#
# Written by: Mark Austin                                October 2024
# ===================================================================

import math

from Vertex01 import Vertex

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

class SimplePolygon:
```

```
area       = 0
perimeter  = 0
name       = ""
coords = []

# Constructor method ...

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

# Set/get name ...

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

def getName(self):
  return self.name

# Compute polygon perimeter ...

def getPerimeter(self):
   lastnode = len( self.coords) - 1

   dperimeter = 0.0;
   for i in range( len(self.coords)-1):
       dx = self.coords[i].getX() - self.coords[i+1].getX();
       dy = self.coords[i].getY() - self.coords[i+1].getY();
       dperimeter = dperimeter + math.sqrt( dx*dx + dy*dy );

   lastnode = len( self.coords) - 1
   dx = self.coords[ lastnode ].getX() - self.coords[0].getX();
   dy = self.coords[ lastnode ].getY() - self.coords[0].getY();
   dperimeter = dperimeter + math.sqrt( dx*dx + dy*dy );

   return dperimeter;

# Draw simple polygon ...

def draw(self, ax):

    # Draw polygon edges ...

    for i in range( len(self.coords)-1):
        xcoords = [ self.coords[i].getX(), self.coords[i+1].getX() ];
        ycoords = [ self.coords[i].getY(), self.coords[i+1].getY() ];
        ax.add_line( Line2D(xcoords, ycoords) )

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

    # Draw polygon vertices as small circles ...
```

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

    # String represention of simple polygon ...

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

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

        polygoninfo.append("--- Perimeter    = {:6.2f} ... \n".format( self.getPerimeter()));
        polygoninfo.append("--- ------------------------------------------------- ");
        return "".join(polygoninfo);
```

Test Program Source Code: TestSimplePolygon01.py

```
# =========================================================
# TestSimplePolygon01.py: Exercise SimplePolygon class ...
# =========================================================

from Vertex01 import Vertex
from SimplePolygon01 import SimplePolygon

import matplotlib.pyplot as plt

# main method ...

def main():
    print("--- Enter TestSimplePolygon01.main()    ... ");
    print("--- ================================== ... ");

    print("--- Part 1: Create list of vertices ... ");

    v01 = Vertex ( 1.0, 5.0 );
    v01.setLabel("v1");
```

```
    v02 = Vertex ( 6.0, 5.0 );
    v02.setLabel("v2");

    v03 = Vertex ( 6.0, 3.0 );
    v03.setLabel("v3");

    v04 = Vertex ( 3.0, 3.0 );
    v04.setLabel("v4");

    v05 = Vertex ( 3.0, 1.0 );
    v05.setLabel("v5");

    v06 = Vertex ( 1.0, 1.0 );
    v06.setLabel("v6");

    print("--- Part 2: Assemble and print simple polygon object ... ");

    polygon01 = SimplePolygon( [ v01, v02, v03, v04, v05, v06 ] )
    polygon01.setName("L-shaped Polygon")

    print( polygon01 )

    print("--- Part 3: Draw simple polygon ... \n");

    # Define Matplotlib figure and axis

    fig, ax = plt.subplots()

    polygon01.draw(ax)

    plt.title('Simple Polygon Object')
    plt.ylabel('y')
    plt.xlabel('x')
    plt.ylim( 0, 6 )
    plt.xlim( 0, 8 )
    plt.grid(True)
    plt.show()

    print("--- ======================================= ... ");
    print("--- Finished TestSimplePolygon01.main()     ... ");

# call the main method ...

main()
```

Program Output: The abbreviated program output is:

```
--- Part 1: Create list of vertices ...
--- Part 2: Assemble and print simple polygon object ...

--- SimplePolygon: L-shaped Polygon ...
--- -----------------------------------------------
```

```
---   Vertex  1: (x,y) = (  1.00,   5.00) ...
---   Vertex  2: (x,y) = (  6.00,   5.00) ...
---   Vertex  3: (x,y) = (  6.00,   3.00) ...
---   Vertex  4: (x,y) = (  3.00,   3.00) ...
---   Vertex  5: (x,y) = (  3.00,   1.00) ...
---   Vertex  6: (x,y) = (  1.00,   1.00) ...
--- Perimeter    =  18.00 ...
--- -------------------------------------------------

--- Part 3: Draw simple polygon ...
```

Questions: Let's start with Vertex01.py.

Part [2a] What does the line of code:

```
from Point01 import Point
```

do, and why is it needed in this program?

Solution: This line imports the class `Point` from the file `Point01.py`. With this class definition in place, other classes (e.g., Vertex) can customize its implementation through class extension (i.e., build a class hierarchy).

This program extends `Point` to create `Vertex`, which in turn, is used by both the `TestSimplePolygon` script and the `SimplePolygon` object model.

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

Solution: With the single statement: `class Vertex(Point):`

Part [2c] List the methods available to the class Vertex via inheritance from class Point.

Solution:

```
--- getX (self)
--- setX (self, xCoord )
--- getY (self)
--- setY (self, yCoord )
--- get_position (self )
--- distance (self, second )
```

```
--- Point.__str__(self)
--- Point.__init__(self, xCoord = 0, yCoord = 0)
```

Part [2d] What does a constructor method do?

Solution: Contructor methods (i.e., `__init__(self, ...)` in Python) create new objects and initialize its attributes. We say that the new object is an instance of the class.

Part [2e] What is the purpose of the line:

```
Point.__init__(self, x, y)
```

within the constructor method for Vertex?

Solution: As part of the process of creating a new object of type Vertex, this line creates an object of type `Point` and initializes its attributes with the coordinate values for `x` and `y`.

Let's move onto TestSimplePolygon01.py:

Part [2f] What does the line:

```
polygon01 = SimplePolygon( [ v01, v02, v03, v04, v05, v06 ] )
```

do?

Solution: This line creates an instance of the class `SimplePolygon`, and passes to the contructor method, details of the perimeter as a list of vertices `v01` through `v06`.

Finally, let's consider SimplePolygon01.py:

Part [2g] The method `getPerimeter()`:

```
def getPerimeter(self):
   lastnode = len( self.coords) - 1

   dperimeter = 0.0;
   for i in range( len(self.coords)-1):
       dx = self.coords[i].getX() - self.coords[i+1].getX();
```

```
        dy = self.coords[i].getY() - self.coords[i+1].getY();
        dperimeter = dperimeter + math.sqrt( dx*dx + dy*dy );

    lastnode = len( self.coords) - 1
    dx = self.coords[ lastnode ].getX() - self.coords[0].getX();
    dy = self.coords[ lastnode ].getY() - self.coords[0].getY();
    dperimeter = dperimeter + math.sqrt( dx*dx + dy*dy );

    return dperimeter;
```

systematically walks along the list of vertices and computes the polygon perimeter. Create a table that shows the iteration value i, and values of dx, dy and dperimeter for each iteration of the loop computation required to compute the perimeter of the test polygon shown in Figure 2.

Solution:

```
Iteration i         dx            dy                             dperimeter
========================================================================
0               5  (6-1)       0  (5-5)                      dperimeter = 5
------------------------------------------------------------------------
1               0  (6-6)      -2  (3-5)              dperimeter = 5 + 2 = 7
------------------------------------------------------------------------
2              -3  (3-6)       0  (3-3)             dperimeter = 7 + 3 = 10
------------------------------------------------------------------------
3               0  (3-3)      -2  (1-3)            dperimeter = 10 + 2 = 12
------------------------------------------------------------------------
4              -2  (1-3)       0  (1-1)            dperimeter = 12 + 2 = 14
------------------------------------------------------------------------
5 (close loop)  0  (1-1)       4  (5-1)            dperimeter = 14 + 4 = 18
========================================================================
```

Part [2h] Write a new version – let's call it version 2 – of the method `getPerimeter()` that takes advantage of the `distance()` method in Point. You should find that it is quite short.

Solution: See part [2g] for the source code to Version 1. Here's Version 2:

```
  def getPerimeter(self):

     dperimeter = 0.0;
     for i in range( len(self.coords)-1):
         dperimeter += self.coords[i].distance( self.coords[i+1] );

     lastnode = len( self.coords) - 1
     dperimeter += self.coords[ lastnode ].distance( self.coords[0] );

     return dperimeter;
```