Solutions to Midterm 1 Exam

Question 1: 20 points.

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

$$f(x) = (x-5)(x-3) - \frac{24}{x(x+2)} \tag{1}$$

Figure 1 plots f(x) over the range $[-4, 8]$.

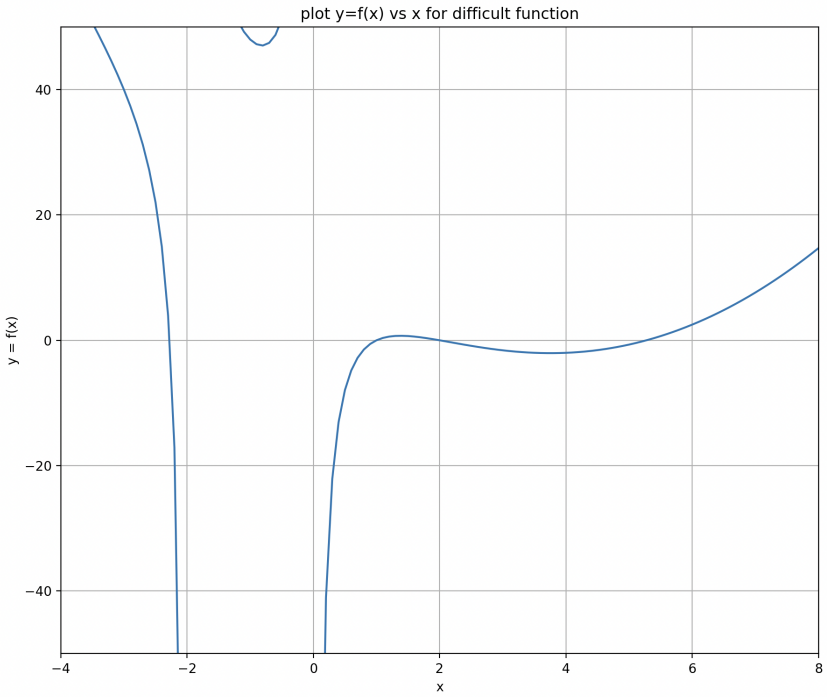


Figure 1. Plot y = f(x) vs x.

From the plot we can see that there are at least four real solutions to f(x) = 0. Also notice that when x = -2 and 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:

```
# ================================================================
# TestDifficultFunction03.py: Explore roots to difficult equation.
#
# Written by: Mark Austin                                March 2025
# ================================================================

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

# Difficult function ...

def difficultFunction(x):
    result = (x-3)*(x-5) - 24/(x*(x+2));
    return result

# main method ...

def main():
    print("--- Enter TestDifficultFunction03.main()     ... ");
    print("--- ======================================== ... ");

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

    # Define problem parameters.

    xcoord = np.linspace( -4, 8, 121)

    # 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,  8.0 )
    plt.ylim( -50.0, 50.0 )
```

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

    print("--- ======================================== ... ");
    print("--- Leave TestDifficultFunction03.main()     ... ");

# call the main method ...

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

The abbreviated textual output is:

```
==============================
          Coord          Value
            (x)            (y)
==============================
          -4.00      6.000e+01
          -3.90      5.817e+01
          -3.80      5.633e+01

... lines of output removed ...

          -2.10     -7.808e+01
          -2.00            inf
          -1.90      1.601e+02

... lines of output removed ...

          -0.20      8.331e+01
          -0.10      1.421e+02
           0.00           -inf
           0.10     -1.001e+02
           0.20     -4.111e+01

... lines of output removed ...

           1.90      1.711e-01
           2.00      0.000e+00
           2.10     -1.775e-01

... lines of output removed ...

           7.80      1.313e+01
           7.90      1.390e+01
           8.00      1.470e+01
==============================
```

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

Part [1a] (7 pts) Show that equation 1 has four real roots, two of them being integers, and the third and fourth roots given by:

$$[r_3, r_4] = \frac{3}{2} \pm \frac{\sqrt{57}}{2}. \tag{2}$$

Solution: Setting f(x) = 0 is equivalent to:

$$(x-5)(x-3)x(x+2) - 24 = 0 \tag{3}$$

Factoring equation 3 gives:

$$(x-1)(x-2)\left(x^2 - 3x - 12\right) = 0 \tag{4}$$

The integer roots are r_1 = 1 and r_2 = 2. Solving $x^2 - 3x - 12 = 0$ gives r_3 and r_4 as shown in equation 2.

Alternate 1: By inspection f(1) = f(2) = 0. Hence, equation 3 can be written

$$(x-1)(x-2)\left(ax^2 + bx + c\right) = 0 \tag{5}$$

Matching the constant, cubic, and fourth-order terms gives c = -12, b = -3 and a = 1. Solving $x^2 - 3x - 12 = 0$ gives r_3 and r_4 as shown in equation 2.

Alternate 2: Using the symbolic package in Python:

```
Abbreviated Python Code:
---------------------------------------

import numpy as np
from sympy import symbols, solve

# Compute roots ...

x = symbols('x')
equation = (x-3)*(x-5) - 24/(x*(x+2));
```

```
Output:
--------------------

Equation Roots:

1
2

3/2 - sqrt(57)/2
3/2 + sqrt(57)/2
```

```
# Compute and print roots ...

roots = solve(equation, x);

print("Equation Roots:");
for root in roots:
    print(root);
```

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

Part [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.

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. It is not used/necessary in this code.
- `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, 8, 121), 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] (2 pts) The fragment of textual output:

```
-0.10      1.421e+02
 0.00           -inf
 0.10     -1.001e+02
```

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?

Solution: When x = 0, f(x) encounters a divide by zero in the expression 24/x. Python records the divide-by-zero as a `-Inf` run-time error. Support for handling of run-time errors, such as divide by zero and NaNs, is defined by the IEEE 754 Standard.

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

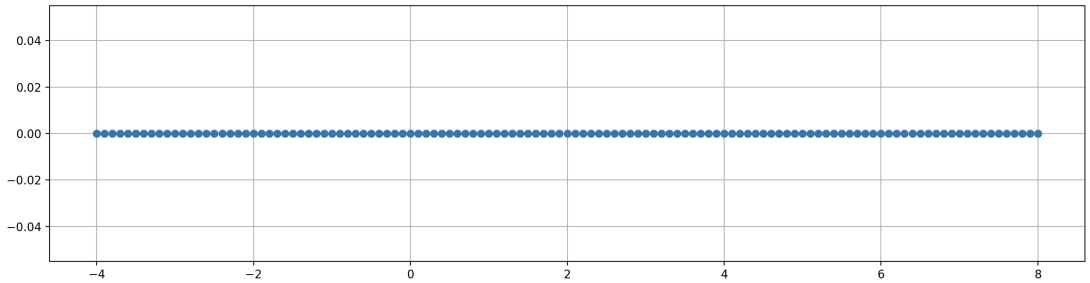
```
xcoord = np.linspace( -4, 8, 121)
```

Solution: Any hand-drawn figure that shows a sequence of 120 intervals of equal length between -4 and 8 (there are 121 nodes) is acceptable. Alternatively, the short Python script:

```
import numpy as np
import matplotlib.pyplot as plt;
    xcoord = np.linspace( -4, 8, 121);
    ycoord = np.zeros(121);

    plt.plot(xcoord, ycoord, 'o' );
    plt.grid(); plt.show();
```

generates:



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

```
print("        {:7.2f}   {:12.3e}".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 $\{: 12.3e\}$. $\{: 7.2f\}$ tells Python to print `x` using up to 7 characters, and two decimal places of accuracy to the right of the decimal point. Similarly, $\{: 12.3e\}$ specifies a floating-point value printed 12 characters wide in exponential format and three decimal places of accuracy to the right-hand side of the decimal point.

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

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

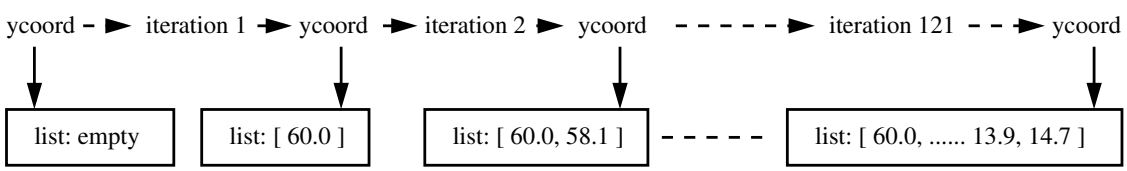
Draw and label the layout of memory for ycoord as it works its way through this looping structure. To keep things manageable, just show an abbreviation of what happens, focusing on the beginning and end of the list assembly. All reasonable answers will be accepted.

Solution: This process can be summarized in a table:

```
Loop   x value   y value                          ycoord list []
-------------------------------------------------------------------
   0       ---       ---                        empty list --> [ ]
   1      -4.0      60.0                                  [ 60.0 ]
   2      -3.9      58.1                              [ 60.0, 58.1 ]
   3      -3.8      58.1                         [ 60.0, 58.1, 56.3 ]

   ... lines of output removed ...

 120       7.9      13.9           [ 60.0, 58.1, 56.3, .... 13.9 ]
 121       8.0      14.7     [ 60.0, 58.1, 56.3, .... 13.9, 14.7 ]
-------------------------------------------------------------------
```



Question 2: 20 points.

Problem Statement. This question covers use of Python to systematically assemble and draw the polygon shown in Figure 2, and then compute/print the polygon area.

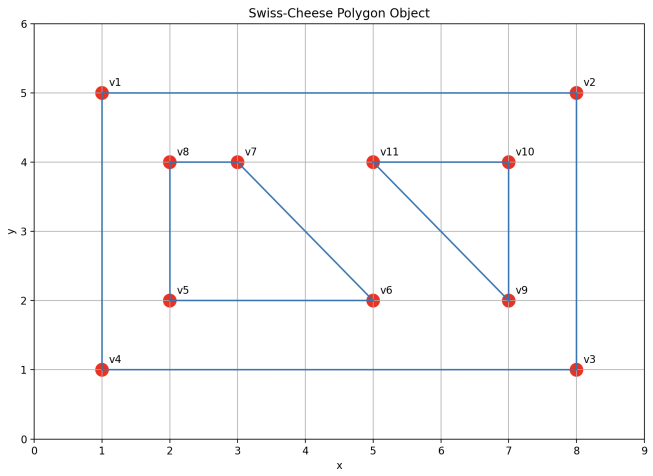


Figure 2. Rectangular polygon with two holes (i.e., Swiss-cheese polygon object).

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.

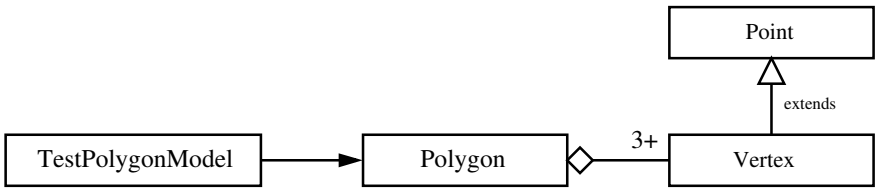


Figure 3. Test program script and classes in a polygon-with-holes system.

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 verticies of the exterior boundary (i.e., *v1*, *v2*, *v3* and *v4*) are organized clockwise, and the verticies of the holes (i.e., *v5*, *v6* $\cdots$ *v11*) 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                          March 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 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)

  # 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: 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                                    March 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 represention of simple polygon ...
# ----------------------------------------

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

```
        # String represention 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}\" ... \n
                               format( i+1, xc, yc, label));

        # String represention 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: TestPolygonModel01.py

```
# ============================================================
# TestPolygonModel01.py: Exercise PolygonModel01 class ...
# ============================================================

from Vertex01 import Vertex
from PolygonModel01 import Polygon

import matplotlib.pyplot as plt

# main method ...

def main():
    print("--- Enter TestPolygonModel01.main()     ... ");
    print("--- ================================== ... ");

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

    v01 = Vertex ( 1.0, 5.0 ); v01.setLabel("v1");
    v02 = Vertex ( 8.0, 5.0 ); v02.setLabel("v2");
    v03 = Vertex ( 8.0, 1.0 ); v03.setLabel("v3");
    v04 = Vertex ( 1.0, 1.0 ); v04.setLabel("v4");

    exteriorLoop = [ v01, v02, v03, v04 ];
```

```
    print("--- Part 2: Lists of vertices for holes 1 and 2 ... ");

    v05 = Vertex ( 2.0, 2.0 ); v05.setLabel("v5");
    v06 = Vertex ( 5.0, 2.0 ); v06.setLabel("v6");
    v07 = Vertex ( 3.0, 4.0 ); v07.setLabel("v7");
    v08 = Vertex ( 2.0, 4.0 ); v08.setLabel("v8");
    v09 = Vertex ( 7.0, 2.0 ); v09.setLabel("v9");
    v10 = Vertex ( 7.0, 4.0 ); v10.setLabel("v10");
    v11 = Vertex ( 5.0, 4.0 ); v11.setLabel("v11");

    hole01 = [ v05, v06, v07, v08 ];
    hole02 = [ v09, v10, v11 ];

    print("--- Part 3: Assemble and print swiss-cheese polygon ... ");

    polygon01 = Polygon( exteriorLoop );
    polygon01.setName("Swiss-Cheese Polygon");
    polygon01.addHole( "hole01", hole01 );
    polygon01.addHole( "hole02", hole02 );

    print( polygon01 )

    print("--- Part 4: Draw polygon exterior + holes ... \n");

    # Define Matplotlib figure and axis

    fig, ax = plt.subplots()

    polygon01.draw(ax)

    plt.title('Swiss-Cheese Polygon Object')
    plt.ylabel('y')
    plt.xlabel('x')
    plt.ylim( 0,  6 )
    plt.xlim( 0,  9 )
    plt.grid(True)
    plt.show()

    print("--- ====================================== ... ");
    print("--- Finished TestPolygonModel01.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 holes 1 and 2 ...
--- Part 3: Assemble and print swiss-cheese polygon ...

--- PolygonModel: Swiss-Cheese Polygon ...
```

```
--- ------------------------------------------------
--- Polygon Boundary
---   Vertex  1: (x,y) = (  1.00,   5.00): label = "v1" ...
---   Vertex  2: (x,y) = (  8.00,   5.00): label = "v2" ...
---   Vertex  3: (x,y) = (  8.00,   1.00): label = "v3" ...
---   Vertex  4: (x,y) = (  1.00,   1.00): label = "v4" ...
--- Polygon Hole: hole01 ...
---   Vertex  1: (x,y) = (  2.00,   2.00): label = "v5" ...
---   Vertex  2: (x,y) = (  5.00,   2.00): label = "v6" ...
---   Vertex  3: (x,y) = (  3.00,   4.00): label = "v7" ...
---   Vertex  4: (x,y) = (  2.00,   4.00): label = "v8" ...
--- Polygon Hole: hole02 ...
---   Vertex  1: (x,y) = (  7.00,   2.00): label = "v9" ...
---   Vertex  2: (x,y) = (  7.00,   4.00): label = "v10" ...
---   Vertex  3: (x,y) = (  5.00,   4.00): label = "v11" ...

--- Polygon Area = 22.000000 ...
--- ------------------------------------------------
--- Part 4: Draw polygon exterior + holes ...
```

Questions: Let's start with Vertex01.py.

Part [2a] (2 pts) 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 `TestPolygonModel01` script and the `Polygon` object model.

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

Solution:

```
--- getX (self)
--- setX (self, xCoord )
--- getY (self)
--- setY (self, yCoord )
--- Point.__str__(self)
--- Point.__init__(self, xCoord = 0, yCoord = 0)
```

Part [2d] (2 pts) Briefly explain what the line is doing?:

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

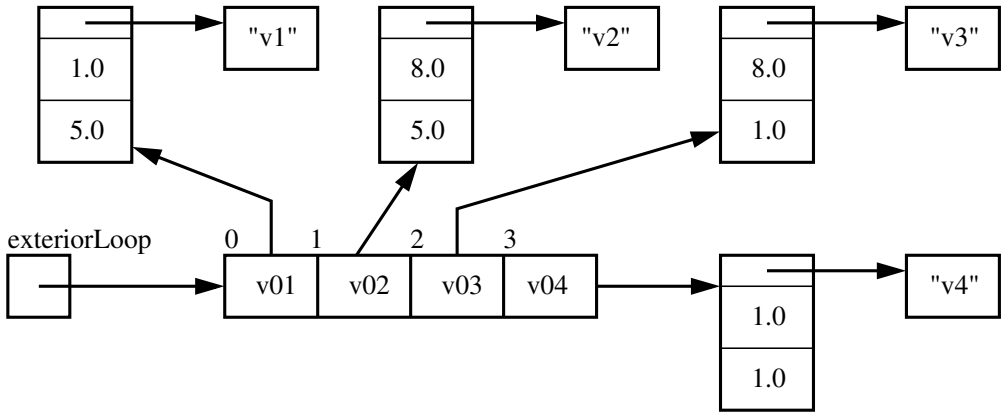
Solution: This line adds a new element to the vertexinfo list, a string representation for Point, from which Vertex is derived.

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

```
v01 = Vertex ( 1.0, 5.0 ); v01.setLabel("v1");
v02 = Vertex ( 8.0, 5.0 ); v02.setLabel("v2");
v03 = Vertex ( 8.0, 1.0 ); v03.setLabel("v3");
v04 = Vertex ( 1.0, 1.0 ); v04.setLabel("v4");

exteriorLoop = [ v01, v02, v03, v04 ];
```

Solution:

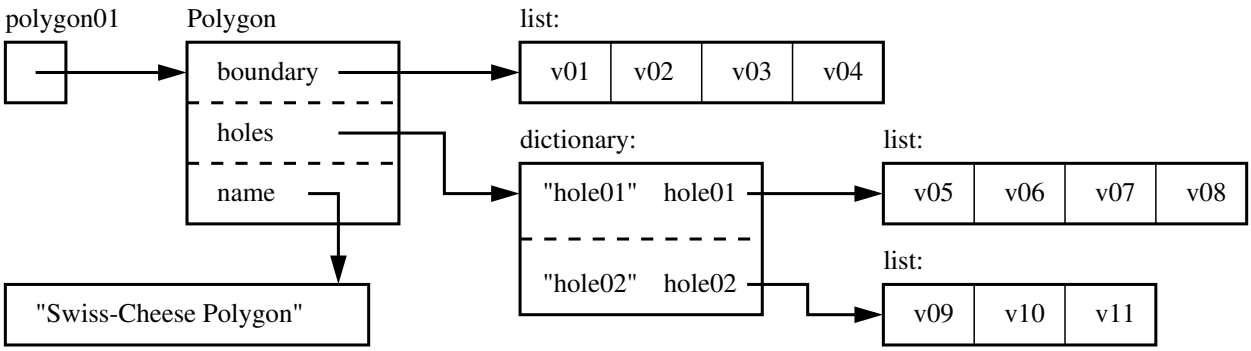


Part [2f] (4 pts) Draw and label a diagram of the layout of memory created by the block of code:

```
polygon01 = Polygon( exteriorLoop );
polygon01.setName("Swiss-Cheese Polygon");
polygon01.addHole( "hole01", hole01 );
polygon01.addHole( "hole02", hole02 );
```

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.

Solution: Here is an abbreviated layout



Finally, let's consider the area computation for the swiss-cheese polygon (see Figure 2).

Part [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 polygon area by first computing the area of the outer boundary polygon, and then adjusting the area for each of the holes. 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()*.

Solution:

Key task	self.boundary	hole	self.holes.get(hole)	self.getLoopArea()	area
area of exterior boundary	[v1, v2, v3, v4]	---	---	+28	28
subtract hole01		hole01	[v5, v6, v7, v8]	-4	24
subtract hole02		hole02	[v9, v10, v11]	-2	22
				Final Area = 22	