Solutions to Practice Final Examination #1

Problem 1—Short answer (10 points)

1a) As written, the program leaves the array in the following state:

```
array
50 50 50 50 50
```

If you had wanted `mystery` to “rotate” the array elements, you would need to run the loop in the opposite order and start the forward indexing at 1, like this:

```python
def mystery(list):
    tmp = list[-1]
    for i in reversed(range(1, len(list))):
        list[i] = list[i - 1]
    list[0] = tmp
```

1b) Calling `enigma()` displays the value 64 on the console. The key to understanding this problem lies in figuring out which \( x \) and \( y \) values are used at each point. In the function returned by `puzzle`, the value of \( x \) comes from the closure and is therefore the value 37 passed to `puzzle`, and the value of \( y \) is the argument to the function \( f \) (which is equal to the value of `riddle` inside `puzzle`), which is 10.

Problem 2—Simple graphics (10 points)

```python
# Creates a GCompound object that represents a pie chart composed
# of the data in the array. The reference point of the GCompound
# is the center of the circle.
def createPieChart(r, data):
    gc = GCompound()
    total = sumArray(data)
    start = 0
    cx = 0
    for value in data:
        sweep = 360.0 * value / total
        arc = GArc(-r, -r, 2 * r, 2 * r, start, sweep)
        arc.setFilled(True)
        arc.setFillColor(WEDGE_COLORS[cx])
        gc.add(arc)
        start += sweep
        cx = (cx + 1) % len(WEDGE_COLORS)
    return gc

# Returns the sum of the array.
def sumArray(array):
    total = 0
    for value in array:
        total += value
    return total
```
Problem 3—Interactive graphics (15 points)

```python
# File: FifteenPuzzle.py

""
This program animates the Fifteen Puzzle.
""

from pgl import GWindow, GRect, GLabel, GCompound

# Constants
SQUARE_SIZE = 60
GWINDOW_WIDTH = 4 * SQUARE_SIZE
GWINDOW_HEIGHT = 4 * SQUARE_SIZE
SQUARE_FILL_COLOR = "LightGray"
PUZZLE_FONT = "18px 'SansSerif'"

# This program simulates the classic Fifteen Puzzle

def FifteenPuzzle():
    def clickAction(e):
        obj = gw.getElementAt(e.getX(), e.getY())
        if obj is not None:
            if tryToMove(gw, obj, SQUARE_SIZE, 0) or
                tryToMove(gw, obj, -SQUARE_SIZE, 0) or
                tryToMove(gw, obj, 0, SQUARE_SIZE) or
                tryToMove(gw, obj, 0, -SQUARE_SIZE):
                return

        gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    initFifteenPuzzle(gw)
    gw.addEventLister("click", clickAction)

    # Adds the numbered squares to the graphics window to create the
    # initial arrangement of the Fifteen Puzzle.

def initFifteenPuzzle(gw):
    x = 0
    y = 0
    for i in range(1, 16):
        gw.add(createNumberedSquare(i, SQUARE_SIZE), x, y)
        if i % 4 == 0:
            x = 0
        y += SQUARE_SIZE
    else:
        x += SQUARE_SIZE

    # Tries to move the object by the specified distance in the x
    # and y directions. If it succeeds, the method returns True.

def tryToMove(gw, obj, dx, dy):
    x = obj.getX() + dx
    y = obj.getY() + dy
    if x < 0 or x >= gw.getWidth() or y < 0 or y >= gw.getHeight():
        return False
    if gw.getElementAt(x, y) is not None:
        return False
    obj.move(dx, dy)
    return True
```
# Problem 4—Strings (10 points)

```python
# File: IsAnagram.py

""
This file defines the isAnagram function from the practice final. ""

from CountLetterFrequencies import createFrequencyTable
from CountLetterFrequencies import updateFrequencyTable

def isAnagram(s1, s2):
    """Returns True if s1 and s2 are anagrams as defined in the problem."""
    return createAnagramTable(s1) == createAnagramTable(s2)

def createAnagramTable(s):
    """Creates a frequency table for s counting only letters."""
    table = createFrequencyTable()
    updateFrequencyTable(table, selectOnlyLetters(s.upper()))
    return table

def selectOnlyLetters(s):
    """Returns a string containing only the letters in s."""
    result = ""
    for ch in s:
        if ch.isalpha():
            result += ch
    return result
```

# Problem 4—Strings (10 points)

```python
# Problem 4—Strings (10 points)

```
Problem 5—Arrays (10 points)

```python
# Returns an image that is twice the size of the original in each # dimension. Each pixel in the original is replicated so that # it appears as a square of four pixels in the new image.

def doubleImage(image):
    oldPixels = image.getPixelArray()
    height = len(oldPixels)
    width = len(oldPixels[0])
    newPixels = createPixelArray(2 * width, 2 * height)
    for i in range(height):
        for j in range(width):
            pixel = oldPixels[i][j]
            newPixels[2 * i][2 * j] = pixel
            newPixels[2 * i][2 * j + 1] = pixel
            newPixels[2 * i + 1][2 * j] = pixel
            newPixels[2 * i + 1][2 * j + 1] = pixel
    return GImage(newPixels)

# Creates a pixel array with dimensions width x height.

def createPixelArray(width, height):
    array = [ ]
    for i in range(height):
        array.append([ 0 ] * width)
    return array
```

Problem 6—Recursive functions (10 points)

```python
# File: SnowflakePerimeter.py

"""This program solves the problem from the final exam."""

def snowflakePerimeter(edge, order):
    """Returns the perimeter of the complete snowflake."""
    return 3 * fractalLineLength(edge, order)

def fractalLineLength(edge, order):
    """Returns the length of a fractal line whose order-0 length is edge."""
    if order == 0:
        return edge
    else:
        return 4 * fractalLineLength(edge / 3, order - 1)
```

It is interesting to test this code in conjunction with the Rational class presented in Chapter 9 because the results are then exact:

<table>
<thead>
<tr>
<th>SnowflakePerimeter</th>
</tr>
</thead>
<tbody>
<tr>
<td>snowflakePerimeter(Rational(1), 0) -&gt; 3</td>
</tr>
<tr>
<td>snowflakePerimeter(Rational(1), 1) -&gt; 4</td>
</tr>
<tr>
<td>snowflakePerimeter(Rational(1), 2) -&gt; 16/3</td>
</tr>
<tr>
<td>snowflakePerimeter(Rational(1), 3) -&gt; 64/9</td>
</tr>
<tr>
<td>snowflakePerimeter(Rational(1), 4) -&gt; 256/27</td>
</tr>
</tbody>
</table>

In the limit as the order goes to infinity, the perimeter approaches infinity, even though the area of the snowflake remains finite.
Problem 7—Defining classes (10 points)

```python
# File: AdvPassage.js

This module defines a passage in the Adventure game.

```class AdvPassage:
    def __init__(self, line):
        """Creates a new passage from the string line."""
        colon = line.find(':')
        if colon == -1:
            raise ValueError("Missing colon in " + line)
        slash = line.find('/', colon)
        if slash == -1:
            self._destination = line[colon + 1:].strip()
            self._key = None
        else:
            self._destination = line[colon + 1:slash].strip()
            self._key = line[slash + 1:].strip().upper()

    def getDirection(self):
        """Returns the direction for this passage."""
        return self._direction

    def getDestination(self):
        """Returns the destination room name for this passage."""
        return self._destination

    def getKey(self):
        """Returns the name of the object that unlocks the passage."""
        return self._key
```

Problem 8—Linked structures (10 points)

```python
def priorityEnqueue(self, v):
    """Adds the value v to the head of the queue."""
    self._head = Queue._Cell(v, self._head)
    if self._tail is None:
        self._tail = self._head
```

Problem 9—Python data structures (15 points)

```python
# Returns True if the player is one or two rooms away from the wumpus.

def playerSmellsAWumpus(cave):
    room = cave["playerLocation"]
    for i in range(3):
        oneRoomAway = cave["connections"][room][i]
        if oneRoomAway == cave["wumpusLocation"]:
            return True
    for j in range(3):
        twoRoomsAway = cave["connections"][oneRoomAway][j]
        if twoRoomsAway == cave["wumpusLocation"]:
            return True
    return False
```