Part 2 (15 points)

The questions in this part should be answered in a separate booklet; do **not** *answer them on this page. The questions are independent, and can be answered in any order. You can use the line numbers in the left-hand margin to identify areas of code in your answers.*

Question 2.1 (4 points)

The following function contains several errors (its docstring says what it *should* be doing). Identify the errors, and explain how you would fix the function.

```
1  def write_line_lengths(filename):
2      """Read in each line of the named file; for each line x the file,
3      print a line to filename.counts containing the number of characters
4      (not counting the newline at the end).
5      """
6      infile = open(filename, 'w')
7      outfile = open(filename + '.counts', 'r')
8      for line in infile:
9          outfile.write(len(line) + '\n')
```

Expected response: About 5 lines. simply

Question 2.3 (3 points)

The following function contains some subtle errors that make it run incorrectly sometimes. Identify the problems, and explain briefly how you would fix the function.

```
1  def remove_negatives(seq):
2      """Remove all negative values from seq."""
3      for x in seq: ← seq changes, if type(x) = int
4          if x < 0:
5              seq.remove(x)
```

Expected response: a short paragraph.

Question 2.2 (3 points)

The following recursive function is supposed to take a list as its argument, and return a list with the same elements but in reversed order. However, it does not work; in fact, it crashes with a runtime error.

```
1  def reverse(seq):
2      """Return s list with the elements of seq in reversed order."""
3      new_first = seq[len(x)-1]
4      new_last = seq[0]
5      new_middle = reverse(seq[1:-1])
6      return new_first + new_middle + new_last
```

Find the problem(s) in this code, and explain how you would fix them.

Expected response: 1 paragraph.

Question 2.4 (5 points)

Each of the following two functions reads in a file, and prints out only its odd-numbered lines (i.e., line 1, line 3, ..., **counting up from 0**).

```
1  def print_odd_lines_1(filename):
2      """Print odd-numbered lines of the given file."""
3      line_list = []
4      with open(filename) as file:
5          line_list = file.readlines()
6      odd_lines = [ line_list[i] for i in range(1, len(line_list), 2) ]
7      for line in odd_lines:
8          print(line)
9
10 def print_odd_lines_2(filename):
11     """Print odd-numbered lines of the given file."""
12     odd = False
13     with open(filename) as file:
14         for line in file:
15             if odd:
16                 print(line)
17             odd = not odd
```

Compare these two functions. Which do you think is the better approach? Justify your answer. *Expected response: 1 paragraph.*

Part 3 (15 points)

The questions in this part should be answered in a separate booklet; do **not** *answer them on this page. You can use the line numbers in the left-hand margin to identify areas of code in your answers.*

Questions 3.1, 3.2, and 3.3 all refer to the following program:

```
1  class TreeNode:
2      def __init__(self,key,value,left,right):
3          self.key = key
4          self.value = value
5          self.left = left
6          self.right = right
7
8      def contains(self ,k):
9          """Return True if the tree rooted at this node contains a node with key k.
10         Otherwise, return False.
11         """
12         if k == self.key:
13             return True
14         else if k < self.key and self.left != None:
15             return self.left.contains(k)
16         else:
17             self.right.contains(k)
18
19 node_0 = TreeNode('A', 'Alpha'), None, None)
20 node_1 = TreeNode('B', 'Bravo')
21 node_2 = TreeNode('C', 'Charlie')
22 node_3 = TreeNode('D', 'Delta')
23 node_4 = TreeNode('E', 'Echo')
24 node_5 = TreeNode('F', 'Foxtrot')
25 node_6 = TreeNode('G', 'Golf'), None, None)
26
27 node_0.left = node_1
28 node_3.left = node_6
29 node_3.right = node_0
30 node_4.right = node_3
31 node_4.left = node_5
32 node_5.right = node_2
```

Question 3.1 (6 points)

Draw a diagram to represent the various `Node` and `string` objects created in this program, and the relationships between them. *Your diagram should follow the same format as you saw in the lectures: draw a box for each object. Your* `Node` *boxes should include the names of each of the data attributes. Add an arrow from each attribute name to the box holding the value of that attribute.*

Expected response: half a page.

Question 3.2 (3 points)

The program above constructs a tree of `TreeNode` objects, with its root at `node_4`.

Is this tree a binary search tree? Why or why not? (And if not, then what must be changed to make this tree into a binary search tree?)

Expected response: 3 lines.

Question 3.3 (6 points)

The `contains` method in the `Node` class *nearly* works, but...

Find and describe **three bugs** in ~~`find`~~ that cause the algorithm to fail (these might represent syntax errors, or cause runtime errors). For each of the bugs you find, give a concrete and compact example of a tree of `Node` objects and a value of `k` that would cause the corresponding runtime error to occur.

Expected answer: Three very short paragraphs. Use trees of the smallest depth possible, and draw diagrams of them, in the same style as in Question 3.1, instead of writing code.