Eric Roberts
CSCI 121

Handout #43
October 31, 2018

Assignment #6—Sorting and Records

Due: Wednesday, November 7, 11:59 P.M.

Problem 1 (Chapter 8, exercise 5)

If you manage to play all seven of your tiles in a single turn, you get a 50-point bonus for what Scrabble players call a ***bingo.*** To help you find bingos, it would be useful to have a function **`listAnagrams`** that takes a string of letters you might have in your Scrabble rack and returns all the legal words that can be formed by rearranging those letters in any order. Although the technique for generating all rearrangements of a string requires concepts beyond the scope of this book, you can achieve the same result by going through the English lexicon and printing every word that contains the same set of letters. The following IDLE log shows a few examples:

```
IDLE
>>> from anagrams import listAnagrams
>>> listAnagrams("aeinrst")
anestri
nastier
ratines
retains
retinas
retsina
stainer
stearin
>>> listAnagrams("adehrst")
dearths
hardest
hardset
hatreds
threads
trashed
>>> listAnagrams("aelqtuz")
quetzal
>>>
```

The starter file for this assignment contains the **`english.py`** file that defines a list of all English words. The hint for this problem given in the text is to look at exercise 3 from Chapter 7, which reads as follows:

> 3. Use the **`list`**, **`sort`**, and **`join`** methods to write a function **`sortLetters`** that rearranges the characters in a string so that they appear in lexicographic order. For example, calling **`sortLetters("cabbage")`** should return the string **`"aabbceg"`**.

Problem 2 (Chapter 8, exercise 7)

Write an implementation of the **`sort`** function that uses the insertion sort algorithm described in this exercise. Make sure that you include startup code that tests your implementation.

Problem 3 (Chapter 9, exercise 2)

The game of ***dominos*** is played using pieces that are usually black rectangles with some number of white dots on each side. For example, the domino

is called the 4-1 domino, with four dots on its left side and one on its right.

Define a simple `Domino` class that represents a traditional domino. Your class should export the following entries:

- A constructor that takes the number of dots on each side
- A `__str__` method that creates a string representation of the domino
- Two getter methods named `getLeftDots` and `getRightDots`

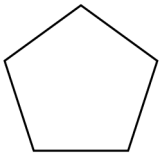
Test your implementation of the `Domino` class by writing a program that creates a full set of dominos from 0-0 to 6-6, where a full set of dominos contains every combination of pips, disallowing duplicates that result from flipping a domino over. Thus, a domino set has a 1-4 domino but not a separate 4-1 domino. Your test program should display the domino set using the following arrangement:

DrawDominoSet

```
0-0  0-1  0-2  0-3  0-4  0-5  0-6
1-1  1-2  1-3  1-4  1-5  1-6
2-2  2-3  2-4  2-5  2-6
3-3  3-4  3-5  3-6
4-4  4-5  4-6
5-5  5-6
6-6
```

Problem 4 (Chapter 9, exercise 10)

Create a new `GRegularPolygon` class that extends `GPolygon` so that it is easy to represent a ***regular polygon,*** which is a polygon whose sides all have the same length and whose angles are equal. The `GRegularPolygon` constructor should take two parameters: `nSides`, which indicates the number of sides, and `radius`, which indicates the distance from the reference point at the center to any of its vertices. The polygon should be oriented so that it is flat along the bottom. Calling `GRegularPolygon(5, 30)`, for example, should create a `GRegularPolygon` object that looks like this:



Similarly, calling `GRegularPolygon(200, 30)` should create a 200-sided polygon whose appearance—at least at the scale of the graphics window—is indistinguishable from that of a circle of radius 30: