4.5 Reading Input from a File

Motivation

Up until now, we've read in all of our input from the keyboard. But imagine that you wanted to read in 1000 food items and put them all in a list! It would be rather tedious to physically type in all of that information. Secondly, quite a bit of information already exists electronically in various types of files. In this section, we'll learn how to read in information from a standard text file. (Note: Many common files you may use, such as Word documents or Excel spreadsheets are NOT text documents. They have their own complicated file formats. Text files are ones you can create in simple text editors, by simply typing regular keys without any special features, such as bolding text, different fonts, etc.)

Opening a File

In order to open a file in Python, all you have to use is the open function. The following opens the file numbers.txt to read from:

```
myFile = open("numbers.txt", "r")
```

It's important to note that this ONLY works if the file from which you are reading is in the same directory as your python program. If it is not, then you have to provide the entire path to the file inside of the double quotes as well.

Once the file is open, Python provides for us two ways in which we can read in the contents of the file. The first method, which will probably be used a majority of the time, is to read in the file, one line at a time. The method that reads in a single line is the readline method.

Let's say that the file numbers.txt has a single integer, *n*, on the first line, indicating the number of values appear in the file, subsequently. Then, following this line, we'll have *n* lines, with one number each. For now, let's assume our goal is simply to add all of the numbers in the file (except for the first one, which is indicating the number of numbes.)

The readline function returns a string storing the contents of the whole line. Thus, in order to truly read in the value on the first line of the file, we have to call two functions as follows:

```
numValues = int(myFile.readline())
```

Python knows WHERE to read from because we called the readline method on the file object, myFile, that we had previously initialized. This instructs Python to read an entire line from the designated file. Since this is the very first readline after the file was opened, it will read in the

first line of the file. We then take that string and convert it to an integer using the int function. Then, this gets stored in the variable numValues.

A sample input file may contain the following contents:

```
5
27
16
22
25
14
```

Now that we know how to read in one integer on a line by itself, we can repeat this task appropriately to finish our program, so that it sums up all the numbers in the file and prints this number to the screen:

```
def main():

    myFile = open("numbers.txt", "r")
    numValues = int(myFile.readline())

    sum = 0
    for i in range(numValues):
        num = int(myFile.readline())
        sum = sum + num

    print("The total was ",sum,".",sep="")
    myFile.close()

main()
```

When running this program, using the sample file given above as numbers.txt, we get the following output:

```
The total was 104.
```

Example Reading from a File with Multiple Items on One Line

In the previous example, we were only able to read in one item per line, since we were forced to read in the whole line all at once. However, in many text files, more than one item is contained on a single line. Luckily, there's a method for strings that allows us to easily read in more than one piece of information on a line, called the split method. The split method will "split" a string into separate items (each of these items must be separated by spaces or tabs), and then return all of the items in a list.

Before we try a full program that utilizes this capability, let's look a a couple lines typed in the IDLE interpreter so we can understand the split method:

```
>>> sentence = "sally goes to the store."
>>> words = sentence.split()
>>> words
['sally', 'goes', 'to', 'the', 'store.']
>>> for i in range(len(words)):
     print("word",i+1,"is",words[i])

word 1 is sally
word 2 is goes
word 3 is to
word 4 is the
word 5 is store.
>>>
```

Thus, if we read a line from a file that has more than one item, we can simply split the string and store the contents into a list. From there, we can access each component one by one. One detail we need to worry about is that we need to know the type of each item, so that we can convert it from a string (which is how it's stored in the list) to whichever type is necessary.

Let's look at a program that reads in a file that has more than one piece of information on a line:

Consider the problem of calculating grades for a class. Typically, a teacher will have several assignments and each assignment will be worth a percentage of the course grade. For example, if the homework is worth 20%, two exams are worth 25% and the final exam is worth 30% and a student scored 90, 80, 100, and 85 on these assignments respectively, we can calculate her final grade as follows:

$$.20 \times 90\% + .25 \times 80\% + .25 \times 100\% + .30 \times 85\% = 88.5\%$$

Essentially, we multiply each grade (out of 100) by its relative weight and add the results together. The relative weights must add up to 1 (100% of the course grade).

In this problem, we'll assume a class has four grades we need to average. Our input file will tell us what percentage each grade is worth, the number of students in the class, and each student's grades. In particular, the input file format will be as follows:

The first line of the input file has a single positive integer, *n*, on it, representing the number of students in the class. The second line of the input file will contain 4 positive real numbers representing the proportion of the class grade of the four assignments. Each of these will be separated by spaces and each is guaranteed to add up to 1. The following *n* lines will each contain four positive integers in between 0 and 100, inclusive, representing the grades on the four assignments, respectively for that student.

Consider an input file with the following contents:

```
5
.6 .1 .1 .2
100 50 70 80
85 90 90 90
78 100 100 100
83 0 100 95
99 92 45 88
```

This file stores information about five students. The second student got an 85% on the first assignment, worth 60% of the course grade and 90% on the rest of her assignments. The third student, got a 78% on the assignment worth 60% of the class and got 100% on the rest of his assignments, and so on.

Our task will be to write a program that prints out the course average for each student. The output should include one line per student with the following format:

```
Student #k: A
```

where k is the number of the student, starting with 1, and A is their average in the class.

Let's break our task down into smaller pieces:

1) Read in the number of students.
2) Read in the weights of each assignment and store into a list (of floats).
3) Go through each student
    a) Read in all of the grades and store into a list (of ints).
    b) Create a variable to store their grade and set it to zero.
    c) Go through each grade:
        i) Multiply this grade by the corresponding weight and add to the grade.
    d) Print out this student's average.

For the purposes of his program, we will read the input from the file grades.txt. Now that we've planned our program, let's take a look at it.

```
def main():

    myFile = open("grades.txt", "r")

    # Read in the number of students and weights.
    numStudents = int(myFile.readline())
    weights = myFile.readline().split()

    # Go through each student.
    for i in range(numStudents):

        # Store their grades.
        allgrades = myFile.readline().split()

        # Add up each grade's contribution.
        grade = 0
        for j in range(len(weights)):
            grade = grade + float(weights[j])*int(allgrades[j])

        print("Student #",i+1,": ","%.2f"%grade, sep="")

    myFile.close()

main()
```

Note that since the lists store strings, we had to convert each list item to a float or int, respectively to properly carry out the desired calculation.

4.6 ***Writing to a File***

Motivation

While it's often adequate to view information scrolling in the IDLE window, sometimes it would be nice to save that information in a file. Thus, in some cases it becomes desirable to store information that a program processes directly into a file. One simple example of the use of an output file would be to store high scores for a video game, for example.

Opening and Closing a File

This works very, very similar to opening and closing a file for reading. The only difference is in the opening step, where you must indicate that the file be opened for writing mode by making "w" the second parameter to the open function:

```
outFile = open("results.txt", "w")
```

When a file is opened for writing, if it previously existed, its contents are emptied out and anything the program writes to the file will be placed at the beginning of the file. In essence, opening a file for writing erases its previous contents.

We close an output file in the identical manner to which we close an input file:

```
outFile.close()
```

Writing to a File

Python keeps writing to a file simple by providing a single method that writes a string to a file:

```
file.write(<string>)
```

The trick in using this method is to only provide a single string to print to the file each time. Also realize that while newlines are automatically inserted between prints to the IDLE screen, they are NOT inserted into a file between each write method. Thus, the programmer has to manually indicate each newline that is printed in the file. The standard way to accomplish writing various items into a file is to use string concatenation (+) frequently. This is similar to how we can get varied prompts for the input method.

Grades Example Extended

We will now extend the grades example so that it produces output to the file "results.txt". Very few changes have been added between the old version and this one:

```
def main():

    myFile = open("grades.txt", "r")
    outFile = open("results.txt", "w")

    # Read in the number of students and weights.
    numStudents = int(myFile.readline())
    weights = myFile.readline().split()

    # Go through each student.
    for i in range(numStudents):

        # Store their grades.
        allgrades = myFile.readline().split()

        # Add up each grade's contribution.
        grade = 0
        for j in range(len(weights)):
            grade = grade + float(weights[j])*int(allgrades[j])

        # Output the result.
        outFile.write("Student #"+(str(i+1))+": "+"%.2f"%grade+"\n")

    myFile.close()
    outFile.close()

main()
```

In fact, except for opening and closing the file, the key change was creating a single string as input for the write method. The most difficult portion of this was remembering to convert the integer (i+1) to a string using the str function. Everything else is identical to the previous version of the program. When this program is executed, assuming that grades.txt is in the same directory as the python program itself, results.txt will appear in the same directory with the appropriate contents. The corresponding output for the input file provided previously is:

```
Student #1: 88.00
Student #2: 87.00
Student #3: 86.80
Student #4: 78.80
Student #5: 90.70
```

4.7 Problems

1) Write a program that prompts the user to enter two strings and prints out the match score to indicate how similar the words are. If the two strings are different lengths, then you should print a score of 0. Otherwise, if the two strings are the same length, the score printed should be the number of characters in corresponding locations that are the same. For example, the match score between "home" and "host" should be 2 and the match score between "paper" and "caper" should be 4.

2) Write a program that asks the user to enter 10 words and prints out the word that comes first alphabetically.

3) Write a program that simulates using a Magic Eight Ball. Your program should have a list of pre-programmed responses to questions stored in a list of strings. Ask the user to enter a question and the present the user with a randomly chosen response from the list you have created.

4) Write a program that reads in a text file of test scores and prints out a histogram of the scores. The file format is as follows: the first line of the file contains a single positive integer, *n*, representing the number of test scores. The following *n* lines will contain one test score each. Each of these test scores will be a non-negative integer less than or equal to 100. The histogram should have one row on it for each distinct test score in the file, followed by one star for each test score of that value. For example, if the tests scores in the file were 55, 80, 80, 95, 95, 95 and 98, the output to the screen should look like:

```
55*
80**
95***
98*
```

5) Write a simulation where the user collects packs of baseball cards. Each pack contains a set of 10 cards, where each card is numbered from 1 to 100. In the simulation, have the user continue to buy packs of cards until she has collected all 100 distinct cards. Each set must contain distinct cards, but two different sets may contain the same card.

6) Generate a set containing each positive integer less than 1000 divisible by 15 and a second set containing each positive integer less than 1000 divisible by 21. Create a set of integers that is divisible by either value, both values and exactly one value. Print out the contents of each of these resultant sets.

7) Write a program that allows the user to add telephone book entries, delete people from a telephone book, allows the user to change the number of an entry, and allows the user to look up a person's phone number. Put in the appropriate error checking.

8) Write a program that asks the user to enter a list of censored words, along with their approved replacements. (For example, "jerk" might be replaced with "mean person".) Read in a sentence from the user and write a modified sentence to the screen where each censored word in the sentence was replaced with the approved replacement. In the given example, the sentence "he is such a jerk" would be converted to "he is such a mean person".

9) Rewrite program #8 so that it reads in the list of censored and replacement words from the file "censordictionary.txt", reads the sentence to convert from the input file "message.txt" and outputs the converted message to the file "safemessage.txt". Create appropriate file formats for all three files.

10) The following problem is taken from a programming contest. The input for the problem is to be read from the file "idnum.in" and the output is to be printed to the screen. The exact file format is given as well as some sample input and the corresponding output for those cases.

The Problem

There are many new summer camps starting up at UCF. As new departments try to start up their summer camps, word got around that there was a computer science summer camp at UCF that was already established. One of the tools that these other summer camps need is a tool to create identification numbers for all the campers. These summer camps have kindly asked Arup to create a computer program to automate the process of allocating identification numbers. Naturally, Arup has decided that this would be an excellent exercise for his BHCSIers. For each summer camp in question, identification numbers will be given in the order that students sign up for the camp. Each camp will have a minimum number for which to start their student identification numbers. Each subsequent number will be generated by adding 11 to the previously assigned number so that each number is sufficiently spaced from the others. After assigning all the identification numbers, your program will need to print an alphabetized list of each student paired with his/her identification number.

The Input

The first line of the input file will consist of a single integer *n* representing how summer camps for which you are assigning identification numbers. For each summer camp, the first line of input will contain one integer $k\ (0 < k \leq 200)$, representing the number of students in that summer camp. The second line of input for each summer camp will contain a single positive integer, *minID*, which represents the minimum identification number for all the students in the camp. (This is the number that will be given to the first student to sign up for the camp.) The following *k* lines will each contain a single name consisting of only 1 to 19 upper case letters. These names are the names of all the students in the class, in the order in which they signed up for the camp. The names within a single summer camp are guaranteed to be unique.

The Output

For every summer camp, the first line will be of the following format:

Summer camp #m:

where m is the number of the summer camp starting with 1.

The following *k* lines should list each student in the summer camp in alphabetical order and his/her identification number, separated by a space.

Put a blank line of output between the output for each summer camp.

Sample Input

```
2
8
2000
SARAH
LISA
ARUP
DAN
JOHN
ALEX
CONNER
BRIAN
10
100001
JACK
ISABELLA
HAROLD
GARY
FRAN
EMILY
DANIELLE
CAROL
BOB
ADAM
```

Sample Output

```
Summer camp #1:
ALEX 2055
ARUP 2022
BRIAN 2077
CONNER 2066
DAN 2033
JOHN 2044
LISA 2011
SARAH 2000

Summer camp #2:
ADAM 100100
BOB 100089
CAROL 100078
DANIELLE 100067
EMILY 100056
FRAN 100045
GARY 100034
HAROLD 100023
ISABELLA 100012
JACK 100001
```