Problem Set #3

Due Tuesday, January 26, 2010, at the **beginning** of class. Assignments turned in more than 10 minutes after the beginning of class will be penalized.

Yes it is the dreaded all programming problem set. And some of these are not easy.

Tips:

- Be sure to run your program - even experienced programmers make minor errors
- Be sure your program runs on cases other than the example shown
- Build your program step by step, printing out lots of intermediate results (or parts of them - some of these problems work on very large files). At the end just remove the intermediate printing steps.
- Always keep in your head what type of object to which each variable refers. If you have trouble with this, then put the type into the variable name (e.g. myListOfStrings or myIntAsString)
- Use the python documentation (available in the Doc directory of wherever you have python installed) or use Google to look up how to do something. There are a lot of options and variants on methods that we didn't explicitly cover in class.
- Stretch yourself by solving a more complex version of one or two of the problems or write a program to solve some problem you've faced in your own research (not required).
- When you get to the end, pause and reflect on how you can now do fairly sophisticated things with large data sets.

1. (15 points) Write a program `read_matrix.py` that opens a file like matrix.txt, stores the entries in a 2-dimensional list and prints the matrix out on the screen. Don't just read the lines and spit them back - you must store the values in a 2-dimensional list first (as if you were going to use them for something). The format of the file is tab-delimited text, with one integer value in each data field and one row of the matrix on each line. Don't worry about getting the output to look pretty, just have it be readable. Make sure your program works on ANY file with the format of matrix.txt.

   ```
   >python read_matrix.py matrix.txt
   0     0     0     0     0
   0     5     2     0     2
   0     3    10     0     0
   0     5     5    15     0
   0     0     2    10    20
   ```

   How would you change your program to print each matrix value multiplied by a command-line specified integer? (you don't need to write the program, just indicate the changes)

2. (15 points) Write a program `print_n.py` that prints the first N nucleotides of the sequence in the file chr21.txt, where N is a command-line option. Also include a command-line option for whether to retain the original case (-o), convert all to lower-case (-l), or convert all to upper-case (-u). Be careful about new-lines in the input and make output with no new-lines! By the way, the file is the actual sequence of the entire human reference chromosome 21.

   ```
   >python print_n.py 1000 -o
   ctccaaagaaattgtagttttcttctggcttagaggtagatcatcttggtccaatcagactgaaatgccttgaggctagatttcagtctttgtggcagctggtgaatttctagtttgcct
   ```

   How would you change your program so that it writes output to a command-line specified file? (you don't need to write the program, just indicate the changes)

3. (25 points) Write a program `find_seq.py` that finds all the positions of exact matches on human chromosome 21 for a DNA sequence given as a command-line argument and prints each position and the total number of matches. Make sure that the search doesn't depend on the case of the query or the sequence in the file. TIPS: You don't need to use lists to solve the problem. Remember that string1.find(string2, start) returns the first position where string2 appears in string1 after start position in string1, and that if string2 isn't found it returns -1 (a common way of indicating not found or failed).

   ```
   >python find_seq.py GATTGATGATA
   1725839
   5312484
   7185252
   8417800
   8639981
   8946117
   11518008
   11582415
   11814084
   14410790
   16307228
   19025838
   22553983
   13 matches
   ```

4. (25 points) The file blastn_OUT.txt contains the text output from a blastn search (slightly edited for clarity). Look at the file and notice that it gives a series of alignments, each preceded by three lines that describe general values for the alignment (Score = etc.). Write a program `blastn_parse.py` that reads a blastn text output file and lists the alignment score and E-value (labeled Expect in the output) for each alignment, one per line.

   ```
   >python blastn_parse.py blastn_OUT.txt
   Score 1742 bits, E-value 0.0
   Score 48.1 bits, E-value 7e-004
   Score 44.1 bits, E-value 0.011
   Score 44.1 bits, E-value 0.011
   Score 42.1 bits, E-value 0.045
   Score 40.1 bits, E-value 0.18
   Score 40.1 bits, E-value 0.18
   Score 40.1 bits, E-value 0.18
   Score 40.1 bits, E-value 0.18
   Score 40.1 bits, E-value 0.18
   Score 38.2 bits, E-value 0.70
   Score 38.2 bits, E-value 0.70
   Score 38.2 bits, E-value 0.70
   Score 38.2 bits, E-value 0.70
   ```

```
Score 38.2 bits, E-value 0.70
Score 38.2 bits, E-value 0.70
Score 38.2 bits, E-value 0.70
Score 38.2 bits, E-value 0.70
Score 38.2 bits, E-value 0.70
Score 38.2 bits, E-value 0.70
Score 38.2 bits, E-value 0.70
Score 38.2 bits, E-value 0.70
```

5. (20 points) Write a program `compute_pval.py` that computes the P-value for a pair alignment, where the score, the mode mu, and the scaling factor lambda are given in that order as command-line arguments. The `math` module has a method `exp(float_val)` that returns e to the power of float_val. WARNING - the name "lambda" is reserved as a special key word by python, use a different variable name.

```
>python compute_pval.py 36.0 25.0 0.79
0.000168245871298
```

6. Challenge problem 1. Assuming that you have obtained the values of mu and lambda for the blast run in problem 4, write a program that makes the same output as for problem 4, but adds an entry for the pair-alignment p-value. Give mu and lambda as command-line arguments.

```
>python blastn_compute_parse.py blastn_OUT.txt 25.0 0.79
Score 42.1 bits, E-value 0.045, Pair P-value foo
etc.
```

7. Challenge problem 2. Write a program that fills an M x N 2-dimensional list with random integer values between -100 and 100 and writes them to a file. The values M and N should be command line arguments. TIP: the `random` module contains functions related to random numbers - use the python docs to figure out how to use it. Confirm that it produces a different matrix every time you run it.

```
>python rand_mat.py 3 3 result.txt
>cat result.txt
-3  12  81
-27  -5  77
19  -44  34
```