CSE 399-005 Python Programming, Spring 2006
Homework 1

March 17, 2006

Instructions

1. This assignment is due on Monday March 20th at 11:59 PM.
2. Homework should be submitted electronically using the `turnin` program. Instructions regarding `turnin` will be posted on the homework page: `www.seas.upenn.edu/~cse39905/homework.html`
3. The input/output format will be *strictly* enforced. We will provide detailed specifications along with a sample input/output case for each problem. Make sure your programs pass these samples before submission, otherwise you will get an automatic 0 for that problem. However, you don't need to worry about malformed input and we ensure that all input are valid.
4. For all problems in this homework, the input is the *standard input* and the output is the *standard output*. So you can test your program like this:

   ```
   cat test.in | python prog1.py > my_test.out
   diff -b test.out my_test.out
   ```

   where `test.in` and `test.out` are the sample testcase for this problem.
5. There will be 10 points for *Pythonic styles* in each assignment. (try to make your code look more pythonic and less C/Java-style.) Also, try your best to write simple, short, and beautiful code. If your code is better or prettier than ours, you get extra points.

Warm-up Problem

This problem will *not* be graded. The reference solution will be posted on web by Wednesday. You can now test the `turnin` program by submitting to `hw0`.

Hint: It's recommended to use standard I/O (`sys.stdin`) for all three problems, although it might be easier to use `raw_input` for Problem 1.

Problem 0 - GCD

Set up Python 2.4.2 for default use on `eniac-1`, or install it on your own computers. [1] Now you are to write a program computing the greatest common divisor (GCD) of two given integers, using the Euclidean Algorithm. You are encouraged to implement both the recursive and non-recursive versions.

The input consists of several lines, with each line containing two positive integers, a and b, separated by a white space ($a, b < 10000$). A pair of `0 0` terminates the input. The output should have one GCD per line for each pair in the input. The following are the sample input and sample output. Please download them and test with your program according to the instructions.

Sample Input gcd.in	Sample Output gcd.out
10 15 55 14 2 1024 0 0	5 1 2

Problems

Note: Your program for Problem 1 should be named `prog1.py` and so on.

Problem 1 - Permutations (30 points)

You are to write a program which takes in a pair of positive integers, n and m, ($0 < m \leq n < 20$), and outputs all the possible permutations of choosing m integers from the set $\{1, 2, \ldots, n\}$. The input is just one line with two integers separated by a white space. The output should print one permutation per line, and a final line containing the number of permutations printed. The permutations should be printed in ascending order lexicographically (e.g., `1 2` precedes `1 3`). In other words, the output is unique for a given input.

Sample Input perm.in	Sample Output perm.out
3 2	1 2 1 3 2 1 2 3 3 1 3 2 6

[1] Instructions can be found on the resources page:
`www.seas.upenn.edu/~cse39905/resources.html`

Problem 2 - QuickSelect (15 points)

In Monday's class we show an example of `quicksort` in Python, using *list comprehensions*. Now you are to adapt it to the problem of choosing the kth-smallest element in a list of length n. The idea is very similar: design a recursive function, and use the first element as the pivot to partition the list. Then decide whether to look into the left half or the right half. A full description of this algorithm can be found in [1, Sec. 9.2].

Note: in Wednesday's class, there is another list comprehension example which solves the selection problem directly (without recursion). Do **not** use it because it is much slower than `quickselect`. (why? how much slower?)

The input will contain several *scenarios*, as in Problem 0. Each scenario will be two lines: the first line has two numbers, n and k, ($0 < k \leq n < 100$), separated by one white space. The second line is a list of n **distinct** integers, separated by white spaces. A line of `0 0` terminates the input. Again, this line is treated as the end-of-input sign and there is *no* output corresponding to it (see sample output). The output will contain one line for each scenario, with a single number in each line denoting the number you found.

Sample Input select.in	Sample Output select.out
5 2 3 10 4 7 19 4 4 11 2 8 3 2 1 100 5 0 0	4 11 5

Problem 3 - Word Frequency (45 points)

You are to write a program that counts the frequencies of each word in a text, and output each word with its count and line numbers where it appears. We define a *word* as a contiguous sequence of non-white-space characters. (hint: `split()`) Note: different capitalizations of the same character sequence should be considered same word, e.g. `Python` and `python`, `I` and `i`. The input will be several lines with the empty line terminating the text. Only alphabet characters and white spaces will be present in the input. The output is formatted as follows: each line begins with a number indicating the frequency of the word, a white space, then the word itself, and a list of line numbers containing this word. You should output from the most frequent word to the least frequent. In case two words have the same frequency, the lexicographically smaller one comes first. All words are in lower case in the output.

Note:

1. you may **not** use the built-in `sort()` function for lists. You should implement yourself a sorting algorithm of your choice but we recommend the

`quicksort` program from the slides.

2. you may **not** use dictionaries since we haven't covered them yet.

Sample Input word.in
Python is a cool language but OCaml is even cooler since it is purely functional

Sample Output word.out
3 is 1 2 1 a 1 1 but 1 1 cool 1 1 cooler 2 1 even 2 1 functional 2 1 it 2 1 language 1 1 ocaml 1 1 purely 2 1 python 1 1 since 2

Problems: 30 + 15 + 45 = 90 points. Pythonic style: 10 points. Total: 100 points.

Debriefing

Please answer these questions in `debrief.txt` and submit it along with the programs. **Note:** You get 5 points off for not responding to this part.

1. How many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Are the lectures too fast, too slow, or just in the right pace?
4. Any other comments?

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.