Due date: **September 29th, 10:30am**

This lab is a sequence of exercises in Python. Each exercise requires you to write a program in a file called `exN.py`, where `N` is the number of the exercise.

General advice and rules:

- Efficiency is not a major worry (within reason); we're exploring what you can do, not how to get the very best solution.
- Don't worry about detecting input errors; assume the input is correct. That implies that both the value and the type of the data supplied are correct.
- **Make sure all programs run on Python 2.5.2 on the ECF workstations.**
- Make sure you exactly follow the input and output specifications for each question.
- When an exercise says to "read input from the user", this means that you should get input from standard input using the `raw_input()` function. Recall that this function can optionally take a prompt string as a parameter. Do not display such a prompt unless the exercise explicitly tells you to do so.
- For each exercise, all output should be to standard output. This means that you should use `print` statements to produce output.
- The specifications below are meant to be precise, unlike real-world requirements. However, there are bound to be places where clarity has not been achieved. If you find yourself confused, please ask.
- Some exercises look ahead to topics not covered at the time when the lab is handed out.
- These exercises are short, but you should still follow the usual style rules, which are just as relevant for all three programming languages used in this course as for any other languages you have used. For example, you should choose helpful names for variables, functions, classes, and methods. Comments are valuable in all languages; even a very short function or code segment may be worth commenting, if you had to think hard to write it.

Academic offenses:

The standard rule is in effect: don't share your work with anyone else. Complete this lab by yourself.

Submission:

Submit your work using the command `submitcsc326f`. Set the first argument – the "assignment#" – to 1.

Exercise 1 [5 marks]

Read lines of input from the user, without giving a prompt. When the input line is `quit`, stop accepting input. As output, print the input lines in reverse order, one on each output line. The line `quit` should not be included in the output.

Requirement: Do **not** use the Python list `reverse` method.

Your submitted file must be named `ex1.py`.

Example: When the input is:

```
hello world
csc
326
32.545
ostrich
quit
```

The output should be:

```
ostrich
32.545
326
csc
hello world
```

Exercise 2 [10 marks]

Read integers input by the user, one per line, without giving a prompt. When the input line is `quit`, stop accepting input. Print the second-smallest number that was input.

Definition: The Nth-smallest number in a list is the number in the list that is strictly larger than *exactly* N - 1 other numbers in the list. For example, in the list [2, 1, 2, 4], the fourth-smallest number is 4, there is no third-smallest or fifth-smallest number, and there are two second-smallest numbers, both 2.

Requirements:

- If there is no second-smallest number, print the empty list `[]`.
- If there is more than one second-smallest number, print a list of these numbers. So if there are, say, three second-smallest numbers, all having value 42, print `[42, 42, 42]`.
- You should be able to do this exercise without necessarily keeping the whole input in memory, but that is not required.

Your submitted file must be named `ex2.py`.

Example: When the input is:

```
4
1
0
10
quit
```

The output should be:

```
1
```

Exercise 3 [10 marks]

Read phone numbers input by the user, one per line, without giving a prompt. When the input line is `quit`, stop accepting input. As output, print the phone numbers with area code 416, each on a separate line.

The phone numbers are all in one of these two forms:

AAA-XXX-XXXX

or

XXX-XXXX

where AAA is the area code, and XXX-XXXX is the rest. If AAA is missing, it is assumed to be 416; in your program, define 416 as a named value rather than using it as a magic number (that is, create a variable whose value is "416" and use this variable rather than repeatedly using the literal "416"). When producing output, add 416 to phone numbers whose area code is missing.

Requirements:

- Use ordinary string operations (not regular expressions) to work with the phone numbers.
- You should have a function that returns true or false depending on whether the area code of a phone number matches.

Your submitted file must be named `ex3.py`.

Example: When the input is:

```
416-887-2345
905-887-2345
585-4596
quit
```

The output should be:

```
416-887-2345
416-585-4596
```

Exercise 4 [15 marks]

Write a `MyList` class implementing a linked list, in a file `MyList.py`. (That makes it a module that can be imported.) Implement list nodes as a class `__Node` within the `MyList` class. Your `MyList` class must provide these methods:

- `append()` – to add an item at the end of the `MyList`
- `traverse()` – to print all the items of the `MyList` with on a single line with a space between each item and a newline at the end
- `__init__(), __len__(), __getitem__()` – to perform their usual tasks

You may not add other public methods, even though you would if you were implementing a standard container.

In `ex4.py`, write a Python program that reads strings into a `MyList` and writes them out again.

(Hint: Remember to import the `MyList` module.)

Requirements:

- Read the input from the user without giving a prompt. Detect end-of-file to stop input, instead of waiting for "quit" (as you did in previous exercises). Use `EOFError` to detect end-of-file.
- On a Unix system (like ECF), control-D signals end of file if you're typing at the keyboard. If you are redirecting input from a file to standard input (this might be more convenient for you), no such tricks are needed.
- You may use standard Python lists and file input techniques in any way you like while reading data in `ex4.py`, but in `MyList.py` you must "build your own". For example to get from one list item to the next, you must follow the link explicitly.

Notice that once you've run `ex4.py`, you'll have a file `MyList.pyc`, the compiled form of `MyList.py`. The only files you need to submit are `ex4.py` and `MyList.py`.

Example: When the input is:

```
This is some sample input.
Isn't this lab a lot of fun? :)
```

The output should be:

```
This is some sample input.
Isn't this lab a lot of fun? :)
```

Exercise 5 [10 marks]

Use your `MyList` class from Exercise 4 to store ordered pairs (two-member tuples) containing names and occupations. The names and occupations must be read from a data file in the form

```
Alice
President
Bob
Vice-President
Eve
Accountant
```

Your program `ex5.py` must work like this:

- It reads the name-occupation ordered pairs from the file whose name is given as the first command-line argument. As they are read, the pairs are stored in a `MyList`.
- It prompts the user for input, with the prompt “next?”. If the user enters a name, it outputs the occupation part of the first ordered pair matching the name; if the user enters an occupation, it outputs the name part of the first ordered pair matching the occupation. If the user enters something that could be either a name or an occupation, it outputs whatever matches first. If no match is found, it outputs "No match found".
- When the user enters `quit`, the session ends.
- You’re not allowed to define more methods in the `MyList` class than are specified in Exercise 4.

Hint: You may need to remove trailing newlines from input.

Example: When the data file displayed above is named `sample.in` and the command

```
python2.5 ex5.py sample.in
```

is run, a sample interactive session is:

```
next?Eve
Accountant
next?Bob
Vice-President
next?Vice-President
Bob
next?Accountant
Eve
next?Ali
No match found
next?quit
```

Exercise 6 [5 marks]

Write a program `ex6.py` that reads the contents of a web page and outputs it. The URL of the website is given as a command-line argument. For example, to output the course website:

```
python2.5 ex6.py http://www.cs.toronto.edu/~ajuma/326f08
```

What you'll output is the HTML of the page content, not a browser-like rendering.

Hints:

- Look into the `urllib` module.
- The solution is very short. Not counting comments and import statements, there is a one-line solution (though putting it all in one line isn't very readable).

Example: When the command

```
python2.5 ex6.py  http://www.cs.toronto.edu/~ajuma/326f08/simple.html
```

is run, the output should be:

```
<html>

   A very simple page.

</html>
```