Lab 1 1 Due at the end of lab

CMPSC 370
Artificial Intelligence
Spring 2006
Bob Roos
Home page: http://cs.allegheny.edu/~rroos/cs370s2006/

Lab 1
Whatever you get done due at the end of lab

Purpose: Gain proficiency in Python

Summary: You will try to implement “familiar” data structures as one or more Python classes. Work in pairs. I want you to hand in printouts of whatever code you have written, even if it’s not working, and on Wednesday we’ll go over the results. Grading will be “informal” (e.g., mostly based on attempting the implementation).

Details:

Stacks
The van Rossum Python tutorial discusses using lists as stacks (see chapter 5). Create a “stack” class in Python. (Chapter 9 talks about creating classes; see also program “frame.py”.) It should have methods named “push”, “pop”, “empty”, and “top.” The first two should be obvious: “push” pushes an object onto the stack and doesn’t return anything; “pop” removes and returns the object on the top (it should return the special object “None” if you attempt to pop from an empty stack). The “empty” method should return true (1) if the stack is empty, 0 otherwise, and the “top” method should return the top element without popping.

Write a main program that creates several stacks and puts them through their paces. Learn about the “raw_input” method and write some code to ask the user to enter stack manipulation commands.

Optional: Trees
If stacks were too easy, try trees. Create one or more Python classes that can be used to represent a general tree, i.e., a data structure consisting of nodes, each of which has zero or more children and zero or one parents. For instance, an “exploded parts” diagram of a complex mechanism might have the form of a general tree:

```
  computer
  |memory
  |  fast memory
  |  |RAM
  |  |cache
  |archival memory
  |  |CD ROM
  |  |diskette
  |  |flash drive
  |  |processor(s)
  |  |CPU
```
Here, “computer” is the root node, with three children named “memory,” “processors,” and “peripherals.” “Memory” has two children named “fast memory” and “archival memory,” etc.

What would be good methods to have in a “node” class? In a “tree” class? Feel free to search the Web for Python implementations of trees; let me know if you find some good (but not overly complicated) ones.