Binary trees, general trees and tree traversal

If you have a hard time getting started or visualizing trees in program code (especially in a text based program) take some time to draw them and see how the connections need to be variables, referencing other nodes, and also watch the second half of today's video on trees and traversal to see the basics of it set up in code.

Implement a *node class for a binary tree*. It should have a variable for ***data*** and a variable for each of two references to other nodes, ***left*** and ***right***.

Implement a class called BinaryTree.
It should have a variable for the ***root*** of the tree.
Implement an operation, ***populate_tree***, that calls for user input for the data of each node.
It should call a recursive function, with the root as a parameter.
The recursive function allows the user to enter a data string for a node. If nothing (or zero, or some escape string) is entered it stops at that point, but if something is entered it makes a node with that data and recursively does the same for the nodes left and right nodes.
Try to make it clear what node you are entering the data for (*print "LEFT", "RIGHT" and indents*).
Implement operations, ***print_preorder***, ***print_inorder*** and ***print_postorder***.
These operations should call recursive versions with the root as a parameter.
The recursive operations should print the data from the parameter node and recursively do the same for their left and right nodes.
The order in which the left recursion, the right recursion and the print are called controls whether the tree's data will be printed preorder, inorder or postorder.

Implement a *node class for a general tree*. It should have a variable for ***data*** and a list of variables for references to ***children*** (other nodes).

Implement a class called GeneralTree.
It should have a variable for the root of the tree.
Implement an operation, ***populate_tree***, that calls for user input for the data of each node.
Can you make similar functionality as in BinaryTree? Since there is no maximum number of children per node (as in BinaryTree, where the max is 2) you will need the program to have a way of not adding more children, but still being able to control the number of children in each level of the tree.
Implement similar print functionality as in the BinaryTree, also with recursive programming.
Do all the print operations from BinaryTree make sense in a GeneralTree?

In all of these problems, use a pencil and piece of paper to keep track of what you're entering and how your tree structure should look, as it can be hard to visualize in a text based program.
Extra exercises on next page...

Extra exercises

- Make an operation on either tree (or both) that takes a value as a parameter, recursively traverses the tree and counts nodes with the former value as its data, returning the number of times that value is in the tree.
- Make an operation on either tree (or both) that takes two values as parameters, recursively traverses the tree and wherever it finds a node with the former value as its data, changes it to the latter value.
- Make an operation on either tree (or both) that takes a value as a parameter, recursively traverses the tree and when it finds a node with that value as its data, swaps its data with that of its parent (floats it one up).
- If you didn't do print_inorder on the general tree (because there isn't one left and one right node), implement it anyway. Inorder would print the first child, then the parent, then each of the rest of the children in order.

These operations will prepare you for more complex trees later on where you may have to float values up the tree or remove nodes from the middle of the tree, without changing the internal order of other nodes.