Some Python Exercises

In case if you feel rusty with your Python programming, or if you are new to the language, you might like to work on a few exercises. This is *not* a homework assignment: we won't collect them or grade them. Feel free to do as many or as few as you like.

Simple program with standard input and output

Write a stand-alone program (as opposed to having the interpreter executing it interactively in the python shell) that asks the user for some input (e.g., their name, an integer), does some calculation (e.g., find all prime numbers smaller than the input integer), and then outputs the answer, addressing the user by name.

File I/O and some string manipulation

Write a function or a stand-alone program that reads in a simple text file and then prints it out in a different format: for example, make sure that every line has exactly 70 characters.

Write a function or a stand-alone program that reads in a list of pairs of numbers from a text file, for example:
[(1,3), (2, 4), (3, 5)]

Take the sum of each pair of numbers, then compute the average of the sum. You do not have to write a list parser yourself; instead, use library functions (e.g., from the “abstract syntax tree” (ast) library).

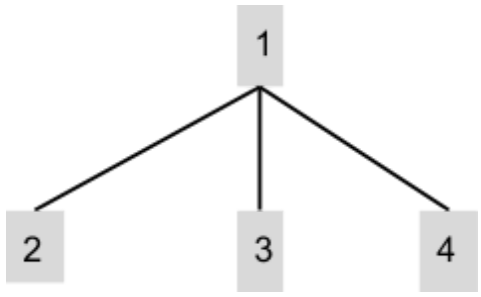
ASCII User Interface for a Puzzle Game

Write a program that prompts the user to play a puzzle game that we’ve discussed in class (lecture 2). Your program doesn't have to solve the puzzle itself, but it needs to keep track of valid moves that the user makes.

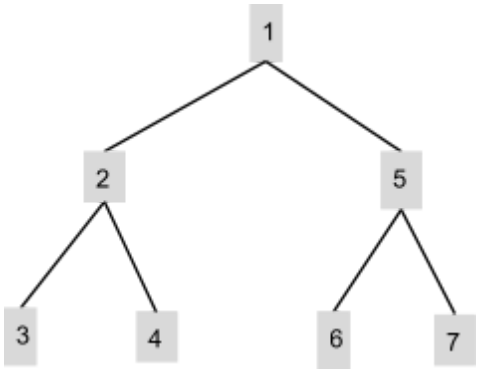
- At each turn, show the user a list of valid actions they can take
- Get their input and update the game
- If they solved the puzzle, tell them that they’ve succeeded, and print out the sequence of actions they took to solve the puzzle.

Tree Manipulation

It's convenient to represent trees with Python's built-in **list** data structure because nested lists are allowed. We represent the root of the tree with the first element of the list, and its children with the rest of the elements of the list. If a child is another tree, it, too, is represented by a list.



For example, [1, 2, 3, 4] represents a simple tree with a root (labeled 1) that has three children (2, 3, and 4).



In this next example, [1, [2, [3, 4]], [5, 6, 7]] represents a tree, rooted at 1, with two subtree children. The first subtree starts at a node labeled 2; it has a single child labeled 3, who, in turn, has a child labeled 4. The second subtree of node 1 starts with a node labeled 5 with children 6 and 7.

Write a program that gets a tree as an input (either by prompting the user or by reading it in from a file -- again, you may use an appropriate library), then perform a few tree-related tasks:

- Write a function that prints out the tree in depth first order; in breadth first order.
- Write a function that figures out how deep is the tree.
- Write a function that computes the average number of children that the interior nodes have.
- Write a function that takes a label as input and searches for it in the tree. Delete every node with that label (You may assume that the root has a unique label, and the user would never ask to delete it).