Extra Credit #4

Due: May 30, 2014 **Points:** 20

A *permutation* of a list is a rearrangement of its elements. For example, given the list `[1, 2, 3]`, all permutations of the list are:

```
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 2, 1]
[3, 1, 2]
```

(note the original list is considered a perturbation of itself).

You are to write a recursive function `perm(L)` that will print all permutations of the list `L`.

Input. This is a function, so it must take a list as its only parameter. You do not need to write a wrapping program to call it, nor do you need to check that the parameter is a list; you may assume that. But it could be a list with 0 elements, so be sure you handle that case!

Output. Print permutations of the input list. For example,

```
perm([1, 'b', -1])
```

is to produce

```
[1, 'b', -1]
[1, -1, 'b']
['b', 1, -1]
['b', -1, 1]
[-1, 'b', 1]
[-1, 1, 'b']
```

The output lines can be in any order, but all must be present, and lines cannot be repeated.

Submit. Name your file "perm.py" and submit it to the Extra Credit #4 area for this class on SmartSite.

Hint: For this problem, try the following. You have a list of n elements. Print the list (this is the first permutation). Swap the zeroth and the first element, and print all the permutations of this list. Then swap the zeroth and the first element to get the original list back. Repeat this for the zeroth and second elements, the zeroth and third element, and so forth, until you reach the end of the list.

It is fair to have `perm(L)` call a "helper function", for example one called `permhelp(curr, L)`, that is recursive — in other words, the helper function is recursive and `perm(L)` simply calls the helper function. In this example, `perm(L)` would call `permhelp(0, L)`, which will do the recursion.