String Permutation

Problem Statement

Given a string, write a function that uses recursion to output a list of all the possible permutations of that string.

For example, given s='abc' the function should return ['abc', 'acb', 'bac', 'bca', 'cab', 'cba']

Note: If a character is repeated, treat each occurence as distinct, for example an input of 'xxx' would return a list with 6 "versions" of 'xxx'

Fill Out Your Solution Below

Let's think about what the steps we need to take here are:

1. Iterate through the initial string – e.g., ‘abc’.

- For each character in the initial string, set aside that character and get a list of all permutations of the string that’s left. So, for example, if the current iteration is on 'b', we’d want to find all the permutations of the string 'ac'.
- Once you have the list from step 2, add each element from that list to the character from the initial string, and append the result to our list of final results. So if we’re on 'b' and we’ve gotten the list ['ac', 'ca'], we’d add 'b' to those, resulting in 'bac' and 'bca', each of which we’d add to our final results.
- Return the list of final results.

Let's go ahead and see this implemented:

In [1]:
```
def permute(s):
    out = []

    # Base Case
    if len(s) == 1:
        out = [s]

    else:
        # For every letter in string
        for i, let in enumerate(s):

            # For every permutation resulting from Step 2 and 3 described above
            for perm in permute(s[:i] + s[i+1:]):

                # Add it to output
                out += [let + perm]

    return out
```

In [2]:
```
permute('abc')
```

Out[2]:
```
['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
```

Test Your Solution

In [3]:
```python
"""
RUN THIS CELL TO TEST YOUR SOLUTION.
"""

from nose.tools import assert_equal

class TestPerm(object):

    def test(self,solution):

        assert_equal(sorted(solution('abc')),sorted(['abc', 'acb', 'bac', 'bca', 'cab', 'cba'
        assert_equal(sorted(solution('dog')),sorted(['dog', 'dgo', 'odg', 'ogd', 'gdo', 'god'

        print('All test cases passed.')



# Run Tests
t = TestPerm()
t.test(permute)
```

```
All test cases passed.
```

Conclusion

There were two main takeaways from tackling this problem:

- Every time we put a new letter in position i, we then had to find all the possible combinations at position i+1 – this was the recursive call that we made. How do we know when to save a string? When we are at a position i that is greater than the number of letters in the input string, then we know that we have found one valid permutation of the string and then we can add it to the list and return to changing letters at positions less than i. This was our base case – remember that we always must have a recursive case and a base case when using recursion!
- Another big part of this problem was figuring out which letters we can put in a given position. Using our sample string “abc”, lets say that we are going through all the permutations where the first letter is "c”. Then, it should be clear that the letter in the 2nd and 3rd position can only be either “a” or “b”, because “a” is already used. As part of our algorithm, we have to know which letters can be used in a given position – because we can’t reuse the letters that were used in the earlier positions.

Good Job!