Anagram Solution

Problem

Given two strings, check to see if they are anagrams. An anagram is when the two strings can be written using the exact same letters (so you can just rearrange the letters to get a different phrase or word).

For example:

    "public relations" is an anagram of "crap built on lies."

    "clint eastwood" is an anagram of "old west action"

Note: Ignore spaces and capitalization. So "d go" is an anagram of "God" and "dog" and "o d g".

Solution

There are two ways of thinking about this problem, if two strings have the same frequency of letters/element (meaning each letter shows up the same number of times in both strings) then they are anagrams of eachother. On a similar vien of logic, if two strings are equal to each other once they are sorted, then they are also anagrams of each other.

You would be able to implement this second solution pretty easily in Python:

In [1]:
```
def anagram(s1,s2):

    # Remove spaces and lowercase letters
    s1 = s1.replace(' ','').lower()
    s2 = s2.replace(' ','').lower()

    # Return boolean for sorted match.
    return sorted(s1) == sorted(s2)
```

In [2]:
```
anagram('dog','god')
```

Out[2]:
```
True
```

In [3]:
```
anagram('clint eastwood','old west action')
```

Out[3]:
```
True
```

In [4]:
```
anagram('aa','bb')
```

Out[4]:
```
False
```

Now the above sorting approach is simple, but is actually not optimal and in an interview setting you would probably be asked to implement a more manual solution involving just counting the number of

letters in each string to test your ability to understand hash tables. Let's build out a fuller solution using counting and Python dictionaries:

In [5]:
```
def anagram2(s1,s2):

    # Remove spaces and lowercase letters
    s1 = s1.replace(' ','').lower()
    s2 = s2.replace(' ','').lower()

    # Edge Case to check if same number of letters
    if len(s1) != len(s2):
        return False

    # Create counting dictionary (Note could use DefaultDict from Collections module)
    count = {}



    # Fill dictionary for first string (add counts)
    for letter in s1:
        if letter in count:
            count[letter] += 1
        else:
            count[letter] = 1

    # Fill dictionary for second string (subtract counts)
    for letter in s2:
        if letter in count:
            count[letter] -= 1
        else:
            count[letter] = 1

    # Check that all counts are 0
    for k in count:
        if count[k] != 0:
            return False

    # Otherwise they're anagrams
    return True
```

In [6]:
```
anagram2('dog','god')
```

Out[6]:
```
True
```

In [7]:
```
anagram2('clint eastwood','old west action')
```

Out[7]:
```
True
```

In [8]:
```
anagram2('dd','aa')
```

Out[8]:
```
False
```

A quick note on the second solution, the use of defaultdict form the collections module would clean up this code quite a bit, and the final for loop could be built into the second for loop, but in the above implementation every step is very clear.

Test Your Solution

Run the cell below to test your solution

In [10]:
```
"""
RUN THIS CELL TO TEST YOUR SOLUTION
"""
from nose.tools import assert_equal

class AnagramTest(object):

    def test(self,sol):
        assert_equal(sol('go go go','gggooo'),True)
        assert_equal(sol('abc','cba'),True)
        assert_equal(sol('hi man','hi     man'),True)
        assert_equal(sol('aabbcc','aabbc'),False)
        assert_equal(sol('123','1 2'),False)
        print("ALL TEST CASES PASSED")

# Run Tests
t = AnagramTest()
t.test(anagram)
```

```
ALL TEST CASES PASSED
```

In [11]:
```
t.test(anagram2)
```

```
ALL TEST CASES PASSED
```

Good Job!