How can I make the grader accept that my solution is correct

By bribre, February 2016

This paper is inspired by some real life cases I met on the forum, but here, I've choosen the following exercise as example:

Quiz 4, Part 4 (Anagrams Test)

Write a function named **test_for_anagrams** that receives two strings as parameters, both of which consist of alphabetic characters and returns True if the two strings are anagrams, False otherwise. Two strings are anagrams if one string can be constructed by rearranging the characters in the other string using all the characters in the original string exactly once. For example, the strings "Orchestra" and "Carthorse" are anagrams because each one can be constructed by rearranging the characters in the other one using all the characters in one of them exactly once. Note that capitalization does not matter here i.e. a lower case character can be considered the same as an upper case character

and this is my solution that I wrote:

```
def test_for_anagrams(string1, string2):
    """
     A function that receives two strings as parameters and returns
     True if the two strings are anagrams, False otherwise.
     Capitalization does not matter.
    """
    str1 = string1.lower()
    str2 = string2.lower()

    # some code that
    # does the logic

     if str1 and str2 are anagram ----> return True
                          if not  ----> return False
```

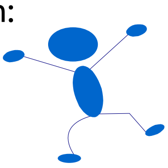
Now I'm ready to test and will call my function on the given example inputs:

```
string_1 = "Orchestra"
string_2 = "Carthorse"
print(test_for_anagrams(string_1, string_2))
```

I run the code and the printed result is True

But one test case might not be enough. I search the Internet to find some more anagrams and run some more tests on them:

```
parliament - partialmen          ---> True
Rearrangement - greenermantra    ---> True
quizsolution - suntooilquiz      ---> True
```

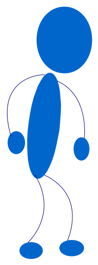


Hurray, my solution is phantastic, it runs like a charm and I will submit it....

But The grader gives me:

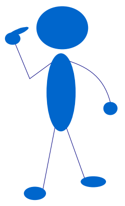
Your solution is not correct: Your funtion was tested with different
test cases. For example for the input.....

Ok – I have to change my code and include this test case too...
I resubmit – and get again another Your solution is not correct...
This time on another test case.



I do the same procedure once again...
...and again
...and again I get another Your solution is not correct...

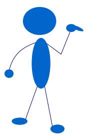
This can't be the right way!



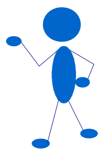
If the grader finds errors in my code by using test cases – I can do the same!
And what did I learn in this course? - For repetitive tasks – use loops, write a program or a function!
I will write my own grader.

For every test case, I will create a list [string1, string2, expected_result]
Then, I create a list of all test cases and put the single tests inside this list.

Let's start!



How can I find good test cases?



I re-read the instructions:

...receives two strings as parameters, both of which consist of alphabetic characters...
Do I have to test this? No, not at this point of the course.
We can assume that the inputs are conform to what the instructions tell us.
What we don't need to check:

- both arguments are of type string
- both arguments are not empty strings
- both arguments don't contain other than alphabetic characters

In real life, we can't assume that inputs are what they are expected to be and we would have to check the inputs whether they are legal or not!

What does the grader expect?

When can I consider my function being correct?

The grader expects that my function returns the correct result for all legal inputs.

But what are legal inputs?
All none-empty strings that contain only alphabetic characters are legal inputs.

I continue analysing the instructions:
Two strings are anagrams if one string can be constructed by rearranging the characters in the other string using all the characters in the original string exactly once. Note that capitalization does not matter

In other words:
– the two strings must contain the same characters
– the two strings must contain the same number of each character
– the second string must be rearranged
– capitalization doesn't matter

And the best of all: the two strings don't need to be “real” words in any language

Now I am ready to start constructing my test cases. Luckily, I don't have to find existing words!

First the test cases that should be recognized as being an anagram:
I choose “anagram” as my base word, but it could be anything else

1) rearrange the word ---> “gamanra”
    testcase1 = [“anagram”, “gamanra”, True]

2) rearrange and change capitalization ---> “GraMana”
    I need two tests to verify that capitalization is ignored for both input strings
    testcase2a= [“anagram”, “GraMana”, True]
    testcase2b= [“AnAgrAm”, “gramana”, True]

3) swap only 2 characters
    a) at the start “naagram”
        testcase3a= [“anagram”,“naagram”, True]
    b) in the middle “anargam”
        testcase3b= [“anagram”, “anargam”, True]
    c) at the end “anagrma”
        testcase3c= [“anagram”, “anagrma”, True]

4) test the shortest possible anagram (containing only two characters)
    testcase4 = [“an”, “na”, True]

At this point, I have no more idea of other combinations and I put this seven cases I found in a list:

```
positive_testcases = [[“anagram”, “gamanra”, True], [“anagram”, “GraMana”, True],
[“AnAgrAm”, “gramana”, True],  [“anagram”,“naagram”, True], [“anagram”, “anargam”,
True], [“anagram”, “anagrma”, True], [“an”, “na”, True]]
```

I can start now to look for testcases that should be recognized NOT being an anagram.

The function should return False if one or more of the mentioned characteristics is not given (see above the analysis of the instructions)

For simplicity, I keep my base word “anagram”
1) a completely different word
    a) of same length
        testcase1a = [“anagram”, “wxdcfvb”, False]
    b) of different length
        testcase1b = [“anagram”, “python”, False]

2) the two words are identic
    I need two additional tests to verify that capitalization is not interpreted as rearranging:
    testcase2a= [“anagram”, “anagram”, False]
    testcase2b= [“aNaGram”, “anagram”, False]
    testcase2c= [“anagram”, “anAgraAM”, False]

3) same characters but not the same number of each character
    a) at the start
        testcase3a= [“anagram”,“nnagram”, False]
    b) in the middle
        testcase3b= [“anaagram”, “anaggram”, False]
    c) at the end
        testcase3c= [“anagrama”, “anagramm”, False]

4) test the shortest not possible anagram (containing only one characters)
    testcase4a = [“a”, “a”, False]

As I don't need to test for legal inputs, I can stop here and create the negative list out of these 9 test cases:

```
negative_testcases = [[“anagram”, “wxdcfvb”, False], [“anagram”, “python”, False],
[“anagram”, “anagram”, False],  [“aNaGram”, “anagram”, False], [“anagram”, “anAgraAM”,
False], [“anagram”,“nnagram”, False], [“anaagram”, “anaggram”, False], [“anagrama”,
“anagramm”, False], [“a”, “a”, False]]
```

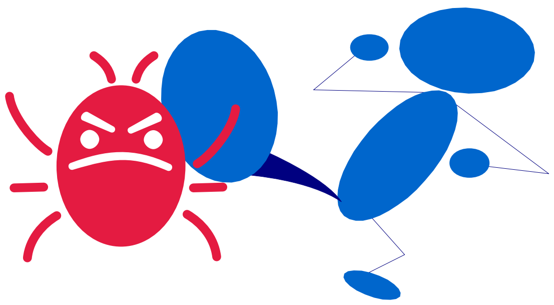
Finally, I feel ready to write my "grader" program (I could also run the test cases one by one...., but what if I find more and more test cases....)

```
testcases = positive_testcases[:]
testcases.extend(negative_testcases)
number_of_tests = len(testcases)
test_results = []
# run my function with all test cases
for n in range(number_of_tests):
    result = test_for_anagrams(testcases[n][0], testcases[n][1])
    # only append failed tests to the test results
    if not result == testcases[n][2]:
          error_message = "input "+testcases[n][0]+" "+ testcases[n][1] +
                          " failed: returned "+ str(result)+" instead of  " +
                          str(testcases[n][2])
          test_results.append(error_message)
if test_results:
    for error_message in test_results:
          print(error_message)
else:
    print("all test passed")
```

When I run this program and get any error message printed, I know that I have to revise my code. The error message itself gives me a hint where I have to look, on which kind of difference between the input strings my function fails.

When I run this program and get "all tests passed" printed, can I assume that my function is correct?
No, I can't be sure that my function is correct. I might have forgotten to add a certain test case and my function can still contain errors.
But: I gained a rather high grade of confidence about the correctness of my code.

It might also be that the grader is less strict...
(For example it might consider identic strings as being an anagram because shuffling doesn't guarantee at 100% that the all the shuffled characters doesn't end in their initial position! In this case, we would have an issue on interpreting the instructions)



Happy testing!

This is my “grader” program converted into a function that I can use to check all functions that take 2 arguments and return something:

```
def my_grader_for_2arg_functions(testcases, function):
    “”””
    A function that tests a function taking two arguments. The inputs to this
    function are: A list of testcases where each test case is a list [arg1, arg2,
    expected_result] and as second input the name of the function to test. The
    results for the failing tests are printed if any, otherwise it prints “all test
    pasesd”
    “”””
    number_of_tests = len(testcases)
    test_results = []
    # run my function on all test cases
    for n in range(number_of_tests):
        result = function(testcases[n][0], testcases[n][1])
        # only append failed tests
        if not result == testcases[n][2]:
            error_message = “input “+testcases[n][0]+” “+ testcases[n][1] +
                            “ failed: returned “+ str(result)+“ instead of  “ +
                            str(testcases[n][2])
            test_results.append(error_message)
    if test_results:
        for error_message in test_results:
            print(error_message)
        return
    print(“all test passed”)
```

A last remark:
I wrote this paper in a normal text editor.
All the code examples are not formatted as code – If you want to use any parts by copy/pasting, you must first format it correctly ;)