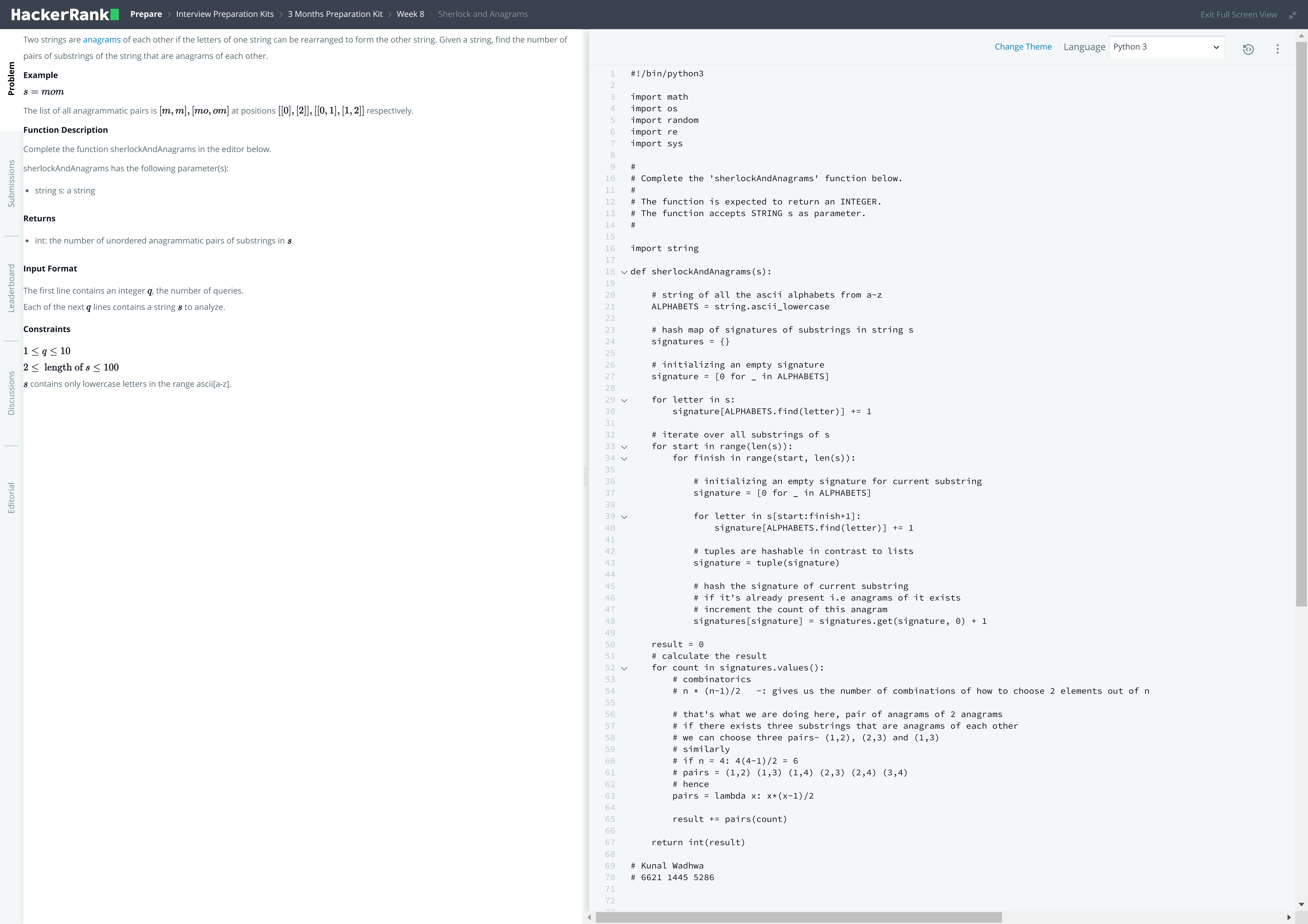

Two strings are anagrams of each other if the letters of one string can be rearranged to form the other string. Given a string, find the number of pairs of substrings of the string that are anagrams of each other.

Example

$s = mom$

The list of all anagrammatic pairs is $[m, m], [mo, om]$ at positions $[[0], [2]], [[0, 1], [1, 2]]$ respectively.

Function Description

Complete the function sherlockAndAnagrams in the editor below.

sherlockAndAnagrams has the following parameter(s):

- string s: a string

Returns

- int: the number of unordered anagrammatic pairs of substrings in s

Input Format

The first line contains an integer q, the number of queries.

Each of the next q lines contains a string s to analyze.

Constraints

$1 \leq q \leq 10$

$2 \leq$ length of $s \leq 100$

s contains only lowercase letters in the range ascii[a-z].

```python
#!/bin/python3

import math
import os
import random
import re
import sys

#
# Complete the 'sherlockAndAnagrams' function below.
#
# The function is expected to return an INTEGER.
# The function accepts STRING s as parameter.
#

import string

def sherlockAndAnagrams(s):

    # string of all the ascii alphabets from a-z
    ALPHABETS = string.ascii_lowercase

    # hash map of signatures of substrings in string s
    signatures = {}

    # initializing an empty signature
    signature = [0 for _ in ALPHABETS]

    for letter in s:
        signature[ALPHABETS.find(letter)] += 1

    # iterate over all substrings of s
    for start in range(len(s)):
        for finish in range(start, len(s)):

            # initializing an empty signature for current substring
            signature = [0 for _ in ALPHABETS]

            for letter in s[start:finish+1]:
                signature[ALPHABETS.find(letter)] += 1

            # tuples are hashable in contrast to lists
            signature = tuple(signature)

            # hash the signature of current substring
            # if it's already present i.e anagrams of it exists
            # increment the count of this anagram
            signatures[signature] = signatures.get(signature, 0) + 1

    result = 0
    # calculate the result
    for count in signatures.values():
        # combinatorics
        # n * (n-1)/2   -: gives us the number of combinations of how to choose 2 elements out of n

        # that's what we are doing here, pair of anagrams of 2 anagrams
        # if there exists three substrings that are anagrams of each other
        # we can choose three pairs- (1,2), (2,3) and (1,3)
        # similarly
        # if n = 4: 4(4-1)/2 = 6
        # pairs = (1,2) (1,3) (1,4) (2,3) (2,4) (3,4)
        # hence
        pairs = lambda x: x*(x-1)/2

        result += pairs(count)

    return int(result)

# Kunal Wadhwa
# 6621 1445 5286
```