Introduction to Data Science -Fall 2022

Assignment#1

Date: 04 Sep, 2022

Due date: Wed, 7th Sep 2022 **Total Marks:60**

Instructions:

- All questions must be in single notebook OR .py file.
- You must follow the file naming conventions, submission file should be named as RollNo.ipynb/.py (i.e. i20-xxxx.ipynb where xxxx is your Roll Number)
- Make use of headings for each question in notebook.
- Any question with Error is not acceptable.
- Late submissions are not allowed and will be marked zero.

Question 1 [10 Marks]

A string is considered "even" if *every* letter in the string appears an even number of times; the string is "odd" if *every* letter in the string appears an odd number of times.

The Problem:

Given a string, determine whether the string is even, odd, or neither.

The Input:

The input consists of a single line, starting in column 1, not exceeding column 70, and containing only the lowercase letters (at least one letter).

The Output:

The output consists of a single integer: print 0 (zero) if the string is even, 1 (one) if the string is odd, or 2 if the string is not even and is not odd (i.e., it is neither).

Sample Input	Sample Output
coachessoaehwwwwww	**0**
coachesarefun	**2**
coachesc	**1**

Question 2 [10 Marks]

Most credit card numbers are encoded with a "Check Digit". A check digit is a digit added to a number (either at the end or the beginning) that validates the authenticity of the number. A simple algorithm is applied to the other digits of the number which yields the check digit. By running the algorithm, and comparing the check digit you get from the algorithm with the check digit encoded with the credit card number, you can verify that they make a valid combination.

The LUHN Formula (Mod 10) for validating credit card numbers has the following steps:

Step 1: Double the value of alternate digits of the credit card number beginning with the second digit from the right (the rightmost digit is the check digit.)

Step 2: Add the individual digits comprising the products obtained in Step 1 to each of the unaffected digits in the original number.

Step 3: The total obtained in Step 2 must be a number ending in zero (30, 40, 50, etc.) for the account number to be validated. The total mod 10 = 0.

For example, to validate the credit card account number 49927398716:

Step 1:

```
4  9  9  2  7  3  9  8  7  1  6
   x2    x2    x2    x2    x2
---------------------------------
  18     4     6    16     2
```

Step 2: 4 +(1+8)+ 9 + (4) + 7 + (6) + 9 +(1+6) + 7 + (2) + 6

Step 3: Sum = 70 : Card number is valid because the 70/10 yields no remainder.

Using the LUHN Formula (Mod 10) for validating credit card numbers, write a program to determine if credit card numbers are valid or invalid.

The input will contain a credit card number. The credit card number will have less than 20 digits. The output is: If the credit card number is valid, output the word "VALID". If the credit card number is not valid, output the word "INVALID" followed by a space and the correct check digit, which is the right-most digit, that would make the credit card number valid.

Sample Input	Sample Output
49927398716	VALID
513467882134	INVALID 2
432876126	VALID

Question 3 [10 Marks]

A "prime" number is an integer greater than 1 with only two divisors: 1 and itself; examples include 5, 11 and 23. Given a positive integer, you are to print a message indicating whether the number is a prime or how close it is to a prime.

The Input:

The first input line contains a positive integer, n ($n \leq 100$), indicating the number of values to check. The values are on the following n input lines, one per line. Each value will be an integer between 2 and 10,000 (inclusive).

The Output:

At the beginning of each test case, output "Input value: v" where v is the input value. Then, on the next output line, print one of the following two messages:

- If the number is a prime, print "Would you believe it; it is a prime!"

- If the number is not a prime, print "Missed it by that much (d)!" where d shows how close the number is to a prime number (note that the closest prime number may be smaller or larger than the given number).

Leave a blank line after the output for each test case. Follow the format illustrated in Sample Output.

Sample Input:	**Sample Output:**
4 23 25 22 10000	Input value: 23 Would you believe it; it is a prime! Input value: 25 Missed it by that much (2)! Input value: 22 Missed it by that much (1)! Input value: 10000 Missed it by that much (7)!

Question 4 [20 Marks]

This question is about automatic spelling correction. Studies have shown that the majority of typing errors are caused by (let's assume the dictionary contains the word “these”):

1. Omitting one letter, e.g., the input word is “thse”.

2. Adding an extra letter, e.g., the input word is “thesce”.

3. Mistyping one letter, e.g., the input word is “thise”.

4. Transposing two adjacent letters, e.g., the input word is “tehse”.

With the aid of a dictionary, word processing systems can often detect and automatically correct these kinds of spelling errors.

The Problem:

You are to write a program that recognizes the four kinds of errors described above.

The Input:

The first input line contains an integer *d (1 ≤ d ≤ 100)* indicating the number of words in the dictionary. Each of the following *d* input lines contains a dictionary word. Assume these words start in column 1, contain at least 1 and at most 15 lowercase letters, and contain no other characters. Following the dictionary words (i.e., the next input line), there is an integer *n (n ≥ 1)* indicating the number of words to be spell checked. Each of the following n input lines contains a word. These words also start in column 1, contain at least 1 and at most 15 lowercase letters, and contain no other characters.

The Output:

Print each input word to be spell checked. Then, if the word is in the dictionary, print CORRECT. If the word is not in the dictionary, then find each word in the dictionary (in the order provided in the dictionary) for which the given input word might be a misspelling, and print the appropriate message from the following list:

ONE LETTER OMITTED FROM *word*

ONE LETTER ADDED TO *word*

ONE LETTER DIFFERENT FROM *word*

TWO LETTERS TRANSPOSED IN *word*

where *word* is a dictionary word. If the input word is not CORRECT and none of the above messages apply, then print UNKNOWN.

Note that two or more of the above messages might be applicable to an input word, and that one message might apply for more than one dictionary word. Note, however, that for a given input word and given dictionary word, at most one of the above messages apply. For each input word, you are to process the dictionary words in the order provided in the input and print all messages that are valid.

Leave a blank line after the output for each input word. Follow the format illustrated in Sample Output.

7	ali
ali	CORRECT
thru	
tu	thu
funfunfun	ONE LETTER OMITTED FROM thru
the	ONE LETTER ADDED TO tu
tuh	ONE LETTER DIFFERENT FROM the
th	TWO LETTERS TRANSPOSED IN tuh
3	ONE LETTER ADDED TO th
ali	
thu	orooji
orooji	UNKNOWN

Question#5 [10 Marks]

In this question you will revise your OOP concepts. Design an Object Oriented Solution for given problem.

Design FAST Isb parking system using object-oriented principles

The solution must have following functionality:

- The solution should tell us how many spots are remaining
- The solution should tell us how many total spots are in the parking lot
- The solution should tell us when the parking lot is full
- The solution should tell us when the parking lot is empty
- The solution should tell us when certain spots are full e.g. when all motorcycle spots are taken
- Tell us how many spots buses are taking up

Assumptions:

- The parking lot can hold motorcycles, cars and buses
- The parking lot has motorcycle spots, car spots and large spots
- A motorcycle can park in any spot
- A car can park in a single compact spot, or a regular spot
- A van can park, but it will take up 3 regular spots
