CSCI-SHU 210 Data Structures

Assignment 1 Python Review

Problem 1 Anagram

An anagram is a word or a phrase made by transposing the letters of another word or phrase; for example, "parliament" is an anagram of "partial men," and "software" is an anagram of "swear oft." Implement function `anagram(string1, string2)` that figures out whether one string is an anagram of another string. The program should ignore white space and punctuation.

Return True if param1 is an anagram of param2.
Return False if param1 is not an anagram of param2.

Example 1:

```
Input: anagram("software", "swear oft")
Return: True
```

Example 2:

```
Input: anagram("parliament", "partial men")
Return: True
```

Problem 2 Number of times to divide

Implement function `number_of_times_divideby2(n)` that takes a positive integer greater than 2 as input and return the number of times one must repeatedly divide this number by 2 before getting a value less than 2.

Example 1:

```
Input: number_of_times_divideby2(999)
Return: 9
```

Example 2:

```
Input: number_of_times_divideby2(123456789)
Return: 26
```

Problem 3: Vigenere Cipher

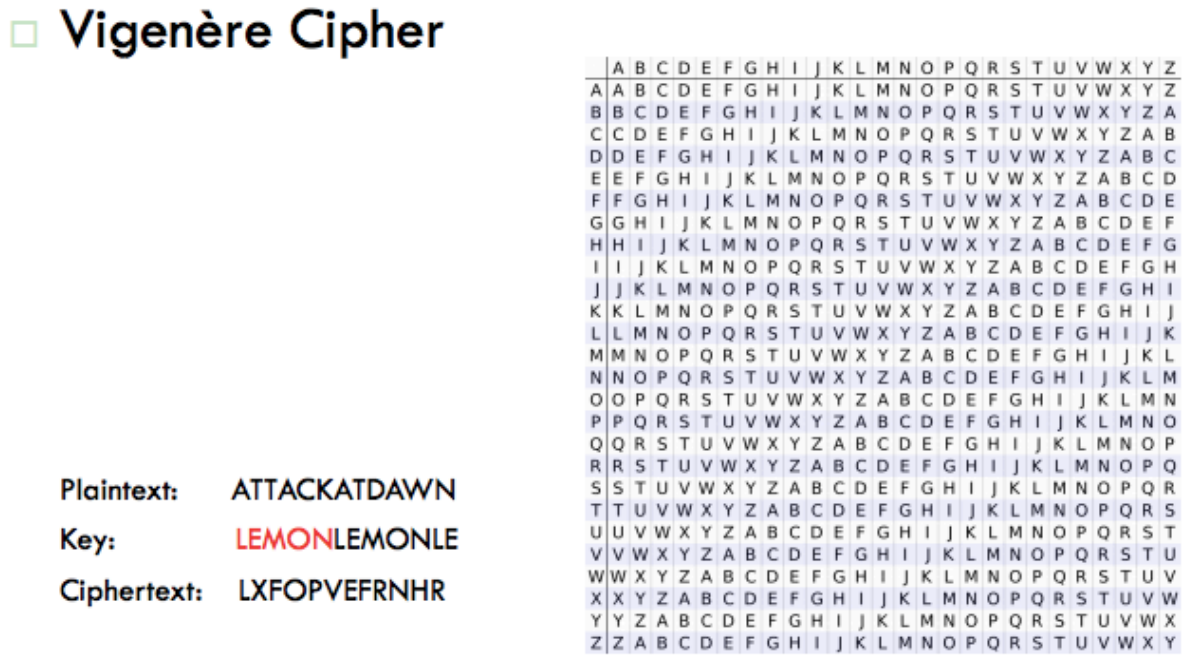
Implement functions,
`vigenere_encrypt(plain, key)`
`vigenere_decrypt(cipher, key)`

`vigenere_encrypt(plain, key)` takes two parameters, the plain text string and the key string. It should return the corresponding **cipher** text.

`vigenere_decrypt(cipher, key)` takes two parameters, the cipher text string and the keystring. It should return the corresponding **plain** text.

Important:

- If the key string is shorter than the plain/cipher text, you should pad the key string to proper length.
- Vigenere Cipher algorithm is demonstrated in the following image.
- You can assume all strings are in upper case.



Example 1:

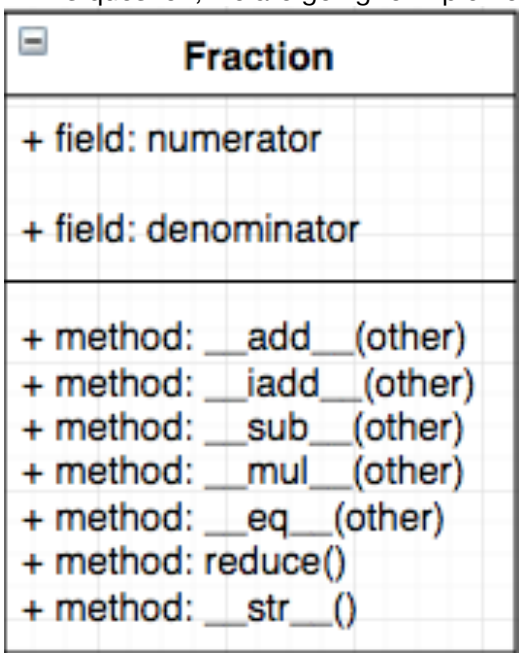
Input: vigenere_encrypt("ATTACKATDAWN", "LEMON")
Return: "LXFOPVEFRNHR"

Example 2:

Input: vigenere_decrypt("LXFOPVEFRNHR", "LEMON")
Return: "ATTACKATDAWN"

Problem 4: Object oriented programming

In this question, we are going to implement the Fraction class:



Important:

- The starting point for this problem is provided in the assignment.
- Support the following operations:
    - Fraction + Fraction
    - Fraction += Fraction
    - Fraction – Fraction
    - Fraction * Fraction
    - Fraction == Fraction
    - print(Fraction)
- Your Fraction does not need to be the most simplified representation, unless `reduce()` function is called.
- You can define new functions and call your new functions. Just make sure the original provided test code runs without problem.
- Check your implementation's correctness with the given test code.