ITM251 Project:

 The project consists of two parts. The first part allows you to practice thinking about problems in a recursive fashion (applying the concepts of functions and recursions), taking advantage of the idea that one can reduce the problem to a simpler version of the same problem. In this part, you will write a recursive function that takes a string as an input and figures out all the possible reorderings of the characters in the string.

The second part will give you experience in thinking about problems in terms of classes, each instance of which contains specific attributes as well as methods for manipulating them. In this part, you will use object-oriented programming to write a Caesar/shift cipher.

**Part1: Permutation of a string**

 A *permutation* is simply a name of the process of finding all possible reorderings of a string. For example: the permutations of the string ‘abc’ are:

* ‘abc’
* ‘acb’
* ‘bac’
* ‘bca’
* ‘cab’
* ‘cba’.

Note that a sequence is a permutation of itself (the trivial permutation). Therefore, you’ll need to write a **recursive** function named ***findpermuations*** that takes a string as a parameter and returns a **list** of all its permutations.

A couple of notes on the requirements:

1. **You MUST use Recursion to solve the problem**
2. You CANNOT use global variables.

Hint: you need to use nested loops to code the solution. The order of the returned permutations does not matter. Please also avoid returning duplicates in your final list.

For this part, you need to submit

1. A word document named (**group#part1pseudocode**): the document includes the Pseudocode that you used to plan to recursively solve this problem.
2. A Python file named (**group#part1answer.py**) that has the code to solve the problem

Part 2: Cipher Text Encryption

**Caesar Cipher**

The idea of the Caesar Cipher is to pick an integer and shift every letter of your message by that integer. In other words, suppose the shift is *k*th . Then, all instances of the *i* letter of the alphabet that appear in the plaintext should become the (*i + k*) th letter of the alphabet in the ciphertext. You will need to be careful with the case in which *i* + *k* > 26 (the length of the alphabet).

Treat uppercase and lowercase letters individually, so that uppercase letters are always mapped to an uppercase letter, and lowercase letters are always mapped to a lowercase letter. If an uppercase letter maps to “A”, then the same lowercase letter should map to “a”. Punctuation and spaces should be retained and not changed. For example, a plaintext message with a comma should have a corresponding ciphertext with a comma in the same position.

We will have a Message class with two subclasses EncryptMessage and DecryptMessage.

Message contains methods that could be used to apply a cipher to a string, either to encrypt or to decrypt a message (since for Caesar codes this is the same action). EncryptMessage has methods to encode a string using a specified shift value; our class will always create an encoded version of the message, and will have methods for changing the encoding. DecryptMessage contains a method used to decode a string.

When you have completed your implementation, you can either create a DecryptMessage instance using an encrypted string that someone provides you and try to decrypt it; or you can encrypt your own EncryptMessage instance, then create a DecryptMessage instance from the encrypted message within the EncryptMessage instance, and try to decrypt it and see if it matches the original plaintext message.

**Your job will be to fill methods for all three of these classes according to the specifications given in the docstrings of part2project.py.**

Please remember that you never want to directly access attributes outside a class that’s why you have getter and setter methods. Don’t overthink this; a getter method should just return an attribute and a set method should just set an attribute equal to the argument passed in. Although they seem simple, we need these methods in order to make sure that we are not manipulating attributes we shouldn’t be. Directly using class attributes outside of the class, itself instead of using getters and setters will result in a point deduction – and more importantly can cause you headaches as you design and implement object class hierarchies.

For this part, you need to submit

1. A word document named (**group#part2pseudocode**): the document includes the Pseudocode that you used to plan to solve this problem.
2. A Python file named (**group#part2answer.py**) that has the code to solve the problem