A L E X M A L A V E

Assignment: Homework #4

Problem 1:
caesar.py

My solution in caesar.*py* contains four functions, **caesarEncrypt**, **caesarDecrypt**, **caesarBreak**, and **main**. **caesarEncrypt** uses a Caesar Cipher to encrypt a given string. The variable **numShift** determines the step by which the substitution happens. To start off, the variable **alphabet** contains all of the letters of the alphabet, which acts as a sort of template. The function then uses a for loop which goes through all of the characters in the string provided by the user, and checks to see if the character in the string is present within the alphabet defined. If so, the variable **newLetter** refers to the corresponding letter in **alphabet**. **numShift** is then added to **newLetter** (to shift the index by the step provided), and the sum is bounded by the number 27 using the modulus operator (so it stays between the range of 0 and 27). The function then returns **newText**, a variable that acts as a sort of string accumulator, and in turn, it's the encrypted text.

caesarDecrypt follows the same pattern as **caesarEncrypt**, except, instead of adding **numShift**, it subtracts it. It follows that for the function to properly decrypt, the **numShift** used for the decryption should be the same as the **numShift** used for the encryption.

The function **caesarBreak** calls **caesarDecrypt** in a for loop in order to go through each possible **numShift** (every number between 1 & 27 because of the way we defined **alphabet**) in an attempt to decrypt a string given using a Caesar Cipher. I use the range [1,28) instead of [0,27) because I use str(v) as the guess number. I therefore call **caesarDecrypt** with *v – 1* rather than just *v*.

main takes one argument, which it uses as the step, or shift, for the Caesar Cipher to solve parts a and b of this problem. It then calls **caesarBreak** to solve part c. The code should be run as follows: *python3 caesar.py <number to use as the shift>*

Problem 2:
threerailencrypt.py & threeraildecrypt.py

My solution to problem 2 consists of two files, *threerailencrypt.py* and threeraildecrypt.py. *threerailencrypt.py* contains two functions: **encrypt** and **main**. **encrypt** splits the string given as its parameter, **plainText**, into two parts, after going through it once, following the rule stated in problem 2 by using the modulus operator to determine remainders (every third character gets put into **rail1**, **rail2**, and **rail3**, respectively). I essentially use a similar method to a two-rail encryption by using the variable **temporary**. After I split it once, I split **temporary**, this time using i%2 (i%2 == 0 and i%2 == 1) instead of i%3 (i%3 == 1 and i%3 == 2), as they are the same.

In *threerailencrypt.py*, main takes two files as arguments. It takes an input file, which it reads, and returns an output file, which it writes to using the **encrypt** function, and saves; thereby creating an output file with encrypted text. The function encrypts text line by line by using truncation in the shape of a for loop that scans for the special character "*n*." The code should be run as: *python3 threerailencrypt.py <pre-made text file with lines of text> <output file name>*

On the other hand, *threeraildecrypt.py* also contains two functions: **decrypt** and **main**. **decrypt**, like **encrypt** in *threerailencrypt.py*, splits the given **cipherText**. The way it does this is by determining the length of the first rail by using the number of characters in **cipherText**. A similar method is done to determine the size of length of **rail2**, and the rest of the characters are put into **rail3**. The function then sorts the rails, by placing the first character of each respective rail behind the other, sequentially, all the while cleaning up the rails (eliminating characters that have been placed in the accumulator string we're filling up). The function then returns this string accumulator, **plainText**.

The **main** function in *threeraildecrypt.py* uses **decrypt** in a similar way that *threerailencrypt.py* uses **encrypt**. It takes an input file, which it reads, and returns an output file, which it writes to using the **decrypt** function, and saves; thereby creating an output file with decrypted text. The function decrypts text line by line by using truncation in the shape of a for loop that scans for the special character "\n." The code should be run as follows: *python3 threeraildecrypt.py <pre-made text file with encrypted lines of text> <output file name>*