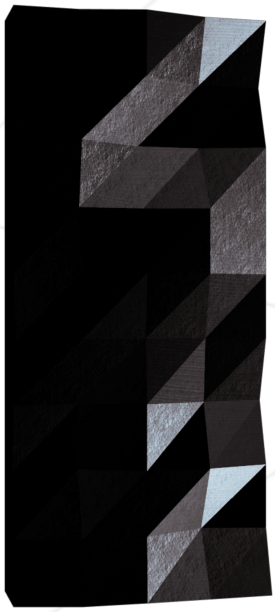

deCipher Me!!

Vigenere

Summary: I saw Michelangelo at work. He had passed his sixtieth year and although he was not very strong, yet in a quarter of an hour he caused more splinters to fall from a very hard block of marble than three young masons in three or four times as long... (Blaise de Vigenere)

Contents

Chapter I

Foreword

Vigenère cipher, type of substitution cipher invented by the 16th-century French cryptographer Blaise de Vigenère and used for data encryption in which the original plaintext structure is somewhat concealed in the ciphertext by using several different monoalphabetic substitution ciphers rather than just one; the code key specifies which particular substitution is to be employed for encrypting each plaintext symbol. Such resulting ciphers, known generically as polyalphabetics, have a long history of usage. The systems differ mainly in the way in which the key is used to choose among the collection of monoalphabetic substitution rules.

For many years this type of cipher was thought to be impregnable and was known as le chiffre indéchiffrable, literally “the unbreakable cipher.”

Chapter II

Introduction

This is your second Rush!! Here's a chance for you to show us your fast problem solving skills and to practice something you just learned. The goal of this project is to take a break from your usual projects to do a quick challenge that will make you really think fast since it's timed! There are no consequences from failing! This is supposed to be a quick and fun exercise for everyone to try. Have fun with it.

Chapter III

General instructions

- This project will be corrected by peers.
- Your project must be written in a language approved by the hack high school program.
- You will only have until 2 pm to finish and push your project.
- You only have to choose one language to do this challenge in, Ruby or Python.
- Ask your peers, mentor, slack or anywhere else if you need any help, and make sure to have fun

`Don't forget to test your code when you're done with it with multiple test cases`

`If Vigenere's cipher is too hard for you, try Caesar's cipher first!`

Chapter IV

Mandatory part: Ruby

Vigenere's Cipher is a tool for encrypting strings. We'll offset each character according to a key sequence. For example, if we encrypt "hackhighschoolisfun" with the key sequence [1, 2, 3], the result would be "icfljlhjvdjrpnlthxo":

Using the Ruby language, write a method that takes a string and a key-sequence, returning the encrypted word. Assume only lower-case letters are used.

IV.0.1 Sample Input/Output

```
Word:   b a n a n a s i n p a j a m a s
Keys:   1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
Cipher: c c q b p d t k q q c m b o d t
```

```
puts vigenere_cipher("toerrishuman", [1]) == "upfssjtivnbo"
puts vigenere_cipher("toerrishuman", [1, 2]) == "uqftsktjvobp"
puts vigenere_cipher("toarrispirate", [1, 2, 3, 4]) == "uqdvskvtjtdxf"
puts vigenere_cipher("zzz", [1, 2, 1]) === "aba"
```

IV.0.2 example

```
def alphabet()
    #code
end

def vigenere_cipher()
    #code
end
```

Note that offsets should wrap around the alphabet - offsetting 'z' by 1 should produce 'a'

Chapter V

Mandatory part: Python

Vigenere's Cipher is a tool for encrypting strings. We'll offset each character according to a key sequence. For example, if we encrypt "hackhighschoolisfun" with the key sequence [1, 2, 3], the result would be "icfljlhjvdjrpnlthxo":

Using the Python language, write a method that takes a string (ARGV[1]) and a key-sequence (ARGV[2]), returning the encrypted word. Assume only lower-case letters are used.

V.0.1 Sample Input/Output

```
Word:   b a n a n a s i n p a j a m a s
Keys:   1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
Cipher: c c q b p d t k q q c m b o d t
```

```
puts vigenere_cipher("toerrishuman", [1]) == "upfssjtivnbo"
puts vigenere_cipher("toerrishuman", [1, 2]) == "uqftsktjvobp"
puts vigenere_cipher("toarrispirate", [1, 2, 3, 4]) == "uqdvskvtjtdxf"
puts vigenere_cipher("zzz", [1, 2, 1]) === "aba"
```

V.0.2 example

```
def alphabet()
    #code

def vigenere_cipher()
    #code
```

Note that offsets should wrap around the alphabet - offsetting 'z' by 1 should produce 'a'

Chapter VI

Turn-in and peer-evaluation

Turn your work in using your `GiT` repository, as usual. Only work present on your repository will be graded in defense.

Good luck and remember to have fun!