# Задание №1

## Условие:

Реализовать одно алфавитный шифр Цезаря для шифрования и дешифрование строки любой длины и заданным ключем (сдвигом алфавита), используется кириллический алфавит, знаки препинания, цифры, верхний и нижний регистр.

## Исходный код программы:

*"""  
Реализовать одно алфавитный шифр Цезаря для шифрования и дешифрование   
строки любой длины и заданным ключем(сдвигом алфавита),   
используется кириллический алфавит, знаки препинания,   
цифры, верхний и нижний регистр.  
"""*GLOBAL\_STR = **"АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцшщъыьэюя0123456789 .,;...?:!()-|\"«»\'"  
  
class** CesarClass(object):  
   
 **def** \_\_init\_\_(self, key, string):  
  
   
   
 out = **""  
 for** char **in** string:  
 out += self.encrypter(char, key)  
 self.encrypted = out   
  
 s = **""  
 for** char **in** out:  
 s += self.decrypter(char, key)  
 self.decrypted = s  
   
 **def** encrypter(self, char, k):  
 *"""  
 Метод для шифрования данных по шифру Цезаря  
 """* s = GLOBAL\_STR  
 index = s.find(char)  
  
 *#Если длина ключа больше самой строки* **if** k > len(s):  
 k = k-len(s)  
  
 **if** index+k >= len(s):  
 **return** s[index+k-len(s)]  
 **else**:  
 **return** s[index+k]  
  
 **def** decrypter(self, char, k):  
 *"""  
 Метод для расшифровки данных по шифру Цезаря  
 """* s = GLOBAL\_STR  
 index = s.find(char)  
  
 *#Если длина ключа больше самой строки* **if** k > len(s):  
 k = k-len(s)  
  
 **if** index-k >= len(s):  
 **return** s[index-k+len(s)]  
 **else**:  
 **return** s[index-k]  
  
**def** main():  
 **try**:  
 k = int(input(**"Введите сдвиг K -> "**))  
 s = str(input(**"Введите строку -> "**))  
 **except**:  
 print(**"Что-то пошло не так при вводе данных"**)  
 **return** obj = CesarClass(k, s)  
 print(**"Зашифрованный ключ: "** + obj.encrypted)  
 print(**"Расшифрованный ключ: "** + obj.decrypted)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## Скриншоты программы:

СКРИНШОТЫ

# Задание №2

## Условие:

Реализовать шифр с использованием кодового слова, используется латинский алфавит с верхним регистром.

## Исходный код программы:

*"""  
Реализовать шифр с использованием кодового слова,  
используется латинский алфавит с верхним регистром.  
"""  
  
# pip3 install collections  
# Нужон для подсчёта кол-ва символов в строке***import** collections  
  
GLOBAL\_STR = **"ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
  
class** UtilClass(object):  
  
 @staticmethod   
 **def** check\_dubl(check\_str):  
 *"""  
 Метод для провери на то, чтоб не было дубликатов символов в строке  
 """* d = collections.defaultdict(int)  
 **for** c **in** check\_str:  
 d[c] += 1  
   
 **for** e **in** d:  
 **if** d[e] != 1:  
 **return False  
   
 return True** @staticmethod  
 **def** check\_values(check\_str):  
   
 *"""  
 Метод для возврата значения того,   
 чтоб символы кодового слова содержались в исходном алфавите   
 """* bool\_flag = **True  
 for** char **in** check\_str:  
 **if** char **not in** GLOBAL\_STR:  
 bool\_flag = **False  
 return** bool\_flag  
  
**class** WordClass(object):  
   
 **def** \_\_init\_\_(self, keyword, s):  
 *"""  
 Конструктор с вводом данных  
 """* self.keyword = keyword  
 self.s = s  
 self.result = **""** self.buf\_str = **""** *#Создаем новый сгенерированный алфавит в self.buf\_str* self.generator()  
  
 *#Шифруем слово* self.encrypter()  
   
 **def** generator(self):  
   
 s = GLOBAL\_STR  
 print(**"\nИсходный алфавит:\n"**+s)  
 keyword = self.keyword  
 **for** char **in** keyword:  
 s = s.replace(char,**""**)  
 self.buf\_str = keyword + s  
 print(**"\nСгенерировали новый алфавит замены:\n"**+self.buf\_str)  
  
 **def** encrypter(self):  
   
 *"""  
 Метод для шифрования данных c использованием кодового слова  
   
 """  
 #Исходная строка* input\_str = self.s  
  
 *#Алфафит замены (старый)* old\_str = GLOBAL\_STR  
  
 *#Алфавит замены (новый)* buf\_str = self.buf\_str  
  
 out = **""  
 for** char **in** input\_str:  
 out += buf\_str[old\_str.find(char)]  
 print(**"\nРезультат:\n"**+out)  
  
  
**def** main():  
 **try**:  
 keyword = str(input(**"Введите кодовое слово -> "**))  
 s = str(input(**"Введите строку -> "**))  
   
 **except**:  
 print(**"Что-то пошло не так при вводе данных"**)  
 **return  
  
 if** UtilClass.check\_values(keyword) == **False**:  
 print(**"Нет символов кодового слова в исходном алфавите!"**)  
 **return  
   
 if** UtilClass.check\_dubl(keyword) == **False**:  
 print(**"Есть дубликат символов в введенной строке!\nТакие слова как WOOD, BOOK и т.д. нельзя использовать"**)  
 **return** WordClass(keyword, s)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## Скриншоты программы:

СКРИНШОТЫ

# Задание №3

## Условие:

Реализовать двух алфавитный шифр Цезаря для шифрования и дешифрование строки любой длины и заданным ключем, используется латинский алфавит и цифры, а так же только нижний регистр.

## Исходный код программы:

*"""  
Реализовать двух алфавитный шифр Цезаря для шифрования и   
дешифрование строки любой длины и заданным ключем,  
используется латинский алфавит и цифры, а так же только нижний регистр.  
"""*dictionaries = {  
 0 : **"abcdefghijklmnopqrstuvwxyz0123456789"**,  
 1 : **"abc0123456789ldefghijkmnopqrstuvwxyz"**}  
  
**def** check\_values(check\_str):  
   
 *"""  
 Метод для возврата значения boolen того,   
 чтоб символы кодируемого слова содержались в алфавитах   
 """* bool\_flag = **True  
 for** char **in** check\_str:  
 **for** d **in** dictionaries:  
 **if** char **not in** dictionaries[d]:  
 bool\_flag = **False  
   
 return** bool\_flag  
  
**class** PolyWordClass(object):  
   
 **def** \_\_init\_\_(self, s, key):  
  
 self.s = s  
 self.key = key  
  
 *#Шифруем слово* self.main\_encoded()  
  
 *#Расшифровка слова* self.main\_decoded()  
   
  
 **def** char\_encoded(self, d, char):  
 *"""  
 Метод для шифрования данных по классическому шифру Цезаря  
 (взял со своего задания №1)  
 """* k = self.key  
 index = d.find(char)  
  
 *#Если длина ключа больше самой строки* **if** k > len(d):  
 k = k-len(d)  
  
 **if** index+k >= len(d):  
 **return** d[index+k-len(d)]  
 **else**:  
 **return** d[index+k]  
  
 **def** char\_decoded(self, d, char):  
 *"""  
 Метод для расшифровки данных по классическому шифру Цезаря  
 (взял со своего задания №1)  
 """* k = self.key  
 index = d.find(char)  
  
 *#Если длина ключа больше самой строки* **if** k > len(d):  
 k = k-len(d)  
  
 **if** index-k >= len(d):  
 **return** d[index-k+len(d)]  
 **else**:  
 **return** d[index-k]  
  
 **def** main\_encoded(self):  
   
 *"""  
 Метод для определения того, какой словарь необходимо использовать  
 В данном случае идёт чередование по вызову dict dictionaries (масло масленное)   
 в зависмости от чётности переменной цикла i  
 """* input\_str = self.s  
 out = **""  
 for** i **in** range(len(input\_str)):  
 out += self.char\_encoded(dictionaries[i % 2], input\_str[i])  
 self.encoded = out  
  
 **def** main\_decoded(self):  
  
 *"""  
 Метод аналогичен main\_encoded, но только с   
 вызовом self.char\_decoded вместо self.char\_encoded  
 """* input\_str = self.encoded  
 out = **""  
 for** i **in** range(len(input\_str)):  
 out += self.char\_decoded(dictionaries[i % 2], input\_str[i])  
 self.decoded = out  
  
**def** main():  
 **try**:  
  
 k = int(input(**"Введите сдвиг -> "**))  
 s = str(input(**"Введите строку -> "**))  
   
 **except**:  
 print(**"Что-то пошло не так при вводе данных"**)  
 **return  
   
 if** check\_values(s) == **False**:  
 print(**"Нет символов введенной строки в исходном алфавите!"**)  
 **return** obj = PolyWordClass(s, k)  
   
 print(**"\n\*\*Шифрование\*\*\nРезультат: "** + obj.encoded)  
 print(**"\n\*\*Расшифровка\*\*\nРезультат: "** + obj.decoded)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## Скриншоты программы:

СКРИНШОТЫ

# Задание №4

## Условие:

Реализовать шифр Виженера, который состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Для зашифровывания может использоваться таблица алфавитов, называемая tabula recta или квадрат (таблица) Виженера. Строка для шифрования должна быть на основе латинского алфавита, ключевое слово вводится с клавиатуры.

## Исходный код программы:

*""""  
Реализовать шифр Виженера, который состоит из последовательности нескольких шифров  
Цезаря с различными значениями сдвига. Для зашифровывания может использоваться таблица алфавитов,  
называемая tabula recta или квадрат (таблица) Виженера.   
Строка для шифрования должна быть на основе латинского алфавита,   
ключевое слово вводится с клавиатуры.  
"""***def** dict\_generator():  
 *"""  
 Создание словаря символов для шифрованного слова из char  
 //можно указать до 257, но тогда пойдут спецсимволы  
 - Возвращает сгенерированный словарь  
 """* d = {}  
 **for** i **in** range(129):  
 d[i] = chr(i)  
 **return** d  
  
**def** crypter(w, k):  
   
 *"""  
 Сравнение кол-ва итераций с длинной ключа на основе индекса w и общего индекса k  
 - Возвращает tuple с двумя словарями  
 """* outdict = {}  
 sum\_outdict = {}  
 buf = 0  
   
 **for** i **in** range(len(w)):  
  
 sum\_outdict[i] = w[i] + k[buf], w[i] - k[buf]  
 outdict[i] = w[i], k[buf]  
 buf += 1  
 **if** (buf >= len(k)):  
 buf = 0  
   
 **return** (outdict, sum\_outdict)  
  
**class** VigenereClass(object):  
 *"""  
 Основной класс для шифрования строк согласно шифру Виженера  
 """* **def** \_\_init\_\_(self, w, k):  
   
 self.w = w  
 self.k = k   
  
 self.input\_d = dict\_generator()  
  
 self.k\_indexes = self.encode\_index(k)  
 self.w\_indexes = self.encode\_index(w)  
  
 self.encoded\_list = self.full\_encode(self.w\_indexes, self.k\_indexes)  
 self.encoded = **""**.join(self.decode\_index(self.encoded\_list))  
  
 self.decoded\_list = self.full\_decode(self.encoded\_list, self.k\_indexes)  
 self.decoded = **""**.join(self.decode\_index(self.decoded\_list))  
  
 **def** encode\_index(self, input\_word):  
 *"""  
 Получение букв в словаре и присвоение им индексов  
 - Возвращает список с индексами элементов словаря  
 """* input\_d = self.input\_d  
 outlist = []  
 **for** char **in** range(len(input\_word)):  
 **for** element **in** input\_d:  
 **if** input\_word[char] == input\_d[element]:  
 outlist.append(element)  
 **return** outlist  
  
 **def** decode\_index(self, input\_list):  
 *"""  
 Дешифровка индексов на символы  
 - Возвращает список list с символами  
 """* input\_d = self.input\_d  
 outlist = []  
 **for** i **in** range(len(input\_list)):  
 **for** element **in** input\_d:  
 **if** input\_list[i] == element:  
 outlist.append(input\_d[element])   
 **return** outlist  
  
 **def** full\_encode(self, w, k):  
 *"""  
 Шифрование строк на основе сравнения индексов ключа и шифруемого слова  
 - Возвращает список индексов list  
 """* input\_d = self.input\_d  
 d = crypter(w, k)  
 compared\_dict = d[0]  
 summed\_dict = d[1]  
 outlist = []  
  
 **for** e **in** compared\_dict:  
 outlist.append(summed\_dict[e][0] % len(input\_d))   
 **return** outlist  
  
 **def** full\_decode(self, w, k):  
 *"""  
 Расшифровка строк строк на основе сравнения индексов ключа и шифруемого слова  
 - Возвращает список индексов list  
 """* input\_d = self.input\_d   
 d = crypter(w, k)  
 compared\_dict = d[0]  
 summed\_dict = d[1]  
 outlist = []  
   
 **for** e **in** compared\_dict:  
 new\_element = (summed\_dict[e][1]+len(input\_d)) % len(input\_d)  
 outlist.append(new\_element)  
  
 **return** outlist  
  
**def** main():  
  
 **try**:  
 s = str(input(**"Введите строку для шифрования -> "**))  
 k = str(input(**"Введите ключ -> "**))  
 **except**:  
 print(**"Что-то пошло не так при вводе данных"**)  
 **return** obj = VigenereClass(s, k)  
   
 print(**"\n\*\*Входные данные\*\*"**)  
 print(**'Слово: '**, obj.w, **"\nИндексы слова: "**, obj.w\_indexes)  
 print(**'Ключ: '**, obj.k, **"\nИндексы ключа: "**, obj.k\_indexes)  
  
 print(**"\n\*\*Шифрование\*\*"**)  
 print(**"List: "**, obj.encoded\_list, **"\nСлово: "**, obj.encoded)  
  
 print(**"\n\*\*Расшифровка\*\*"**)  
 print(**"List: "**, obj.decoded\_list, **"\nСлово: "**, obj.decoded)  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## Скриншоты программы:

СКРИНШОТЫ