# Аннотация

**Среда программирования:** Visual Studio Code

**Язык программирования:** Python 3

**Процедуры для запуска программы:** $ python3 <имя\_файла>.py

**Пословица-тест:** Красивыми словами пастернак не помаслишь

**Текст для проверки работы:** Вот пример статьи на тысячу символов. Это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазинах или для небольших информационных публикаций. В таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. Но можно и без него. На тысячу символов рекомендовано использовать один или два ключа и одну картину. Текст на тысячу символов это сколько примерно слов? Статистика показывает, что тысяча включает в себя сто пятьдесят или двести слов средней величины. Но, если злоупотреблять предлогами, союзами и другими частями речи на один или два символа, то количество слов неизменно возрастает. В копирайтерской деятельности принято считать тысячи с пробелами или без. Учет пробелов увеличивает объем текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. Считать пробелы заказчики не любят, так как это пустое место. Однако некоторые фирмы и биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. Согласитесь, читать слитный текст без единого пропуска, никто не будет. Но большинству нужна цена за тысячу знаков без пробелов.

**Интерфейс:** #в разработке#

**Блок E: ШИФРЫ ГАММИРОВАНИЯ**



## 

* **Гаммирование ГОСТ 28147-89**

**Код программы:**

# -\*- coding:utf-8 -\*-

import sys

import numpy.random

import itertools

from demo import alphabet, input\_for\_cipher\_short, input\_for\_cipher\_long, output\_from\_decrypted

import binascii

class GostCrypt(object):

    def \_\_init\_\_(self, key, sbox):

        self.\_key = None

        self.\_subkeys = None

        self.key = key

        self.sbox = sbox

    @staticmethod

    def \_bit\_length(value):

        return len(bin(value)[2:])

    @property

    def key(self):

        return self.\_key

    @key.setter

    def key(self, key):

        self.\_key = key

        self.\_subkeys = [(key >> (32 \* i)) & 0xFFFFFFFF for i in range(8)] #8 кусков

    def \_f(self, part, key):

        temp = part ^ key

        output = 0

        for i in range(8):

            output |= ((self.sbox[i][(temp >> (4 \* i)) & 0b1111]) << (4 \* i))

        return ((output >> 11) | (output << (32 - 11))) & 0xFFFFFFFF

    def \_decrypt\_round(self, left\_part, right\_part, round\_key):

        return left\_part, right\_part ^ self.\_f(left\_part, round\_key)

    def encrypt(self, plain\_msg):

        def \_encrypt\_round(left\_part, right\_part, round\_key):

            return right\_part, left\_part ^ self.\_f(right\_part, round\_key)

        left\_part = plain\_msg >> 32

        right\_part = plain\_msg & 0xFFFFFFFF

        for i in range(24):

            left\_part, right\_part = \_encrypt\_round(left\_part, right\_part, self.\_subkeys[i % 8])

        for i in range(8):

            left\_part, right\_part = \_encrypt\_round(left\_part, right\_part, self.\_subkeys[7 - i])

        return (left\_part << 32) | right\_part

    def decrypt(self, crypted\_msg):

        def \_decrypt\_round(left\_part, right\_part, round\_key):

            return right\_part ^ self.\_f(left\_part, round\_key), left\_part

        left\_part = crypted\_msg >> 32

        right\_part = crypted\_msg & 0xFFFFFFFF

        for i in range(8):

            left\_part, right\_part = \_decrypt\_round(left\_part, right\_part, self.\_subkeys[i])

        for i in range(24):

            left\_part, right\_part = \_decrypt\_round(left\_part, right\_part, self.\_subkeys[(7 - i) % 8])

        return (left\_part << 32) | right\_part

sbox = [numpy.random.permutation(l) for l in itertools.repeat(list(range(16)), 8)]

sbox = (

    (4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3),

    (14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9),

    (5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11),

    (7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3),

    (6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2),

    (4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14),

    (13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12),

    (1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12),

    )

key = 18318279387912387912789378912379821879387978238793278872378329832982398023031

text\_short = input\_for\_cipher\_short().encode().hex()

text\_short = int(text\_short, 16)

gost\_short = GostCrypt(key, sbox)

enc\_txt = gost\_short.encrypt(text\_short)

dec\_txt = gost\_short.decrypt(enc\_txt)

dec\_txt = bytes.fromhex(hex(dec\_txt)[2::]).decode('utf-8')

text\_long = input\_for\_cipher\_long().encode().hex()

text\_long = int(text\_long, 16)

print(f'''

Зашифрованный текст:

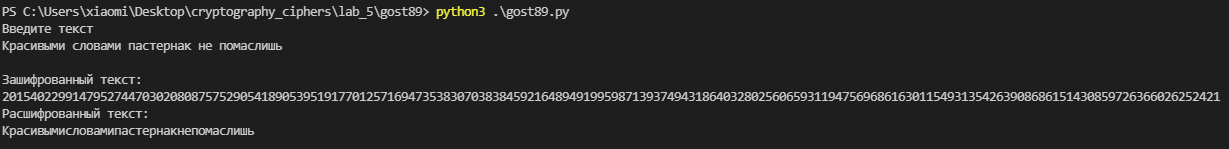
{enc\_txt}

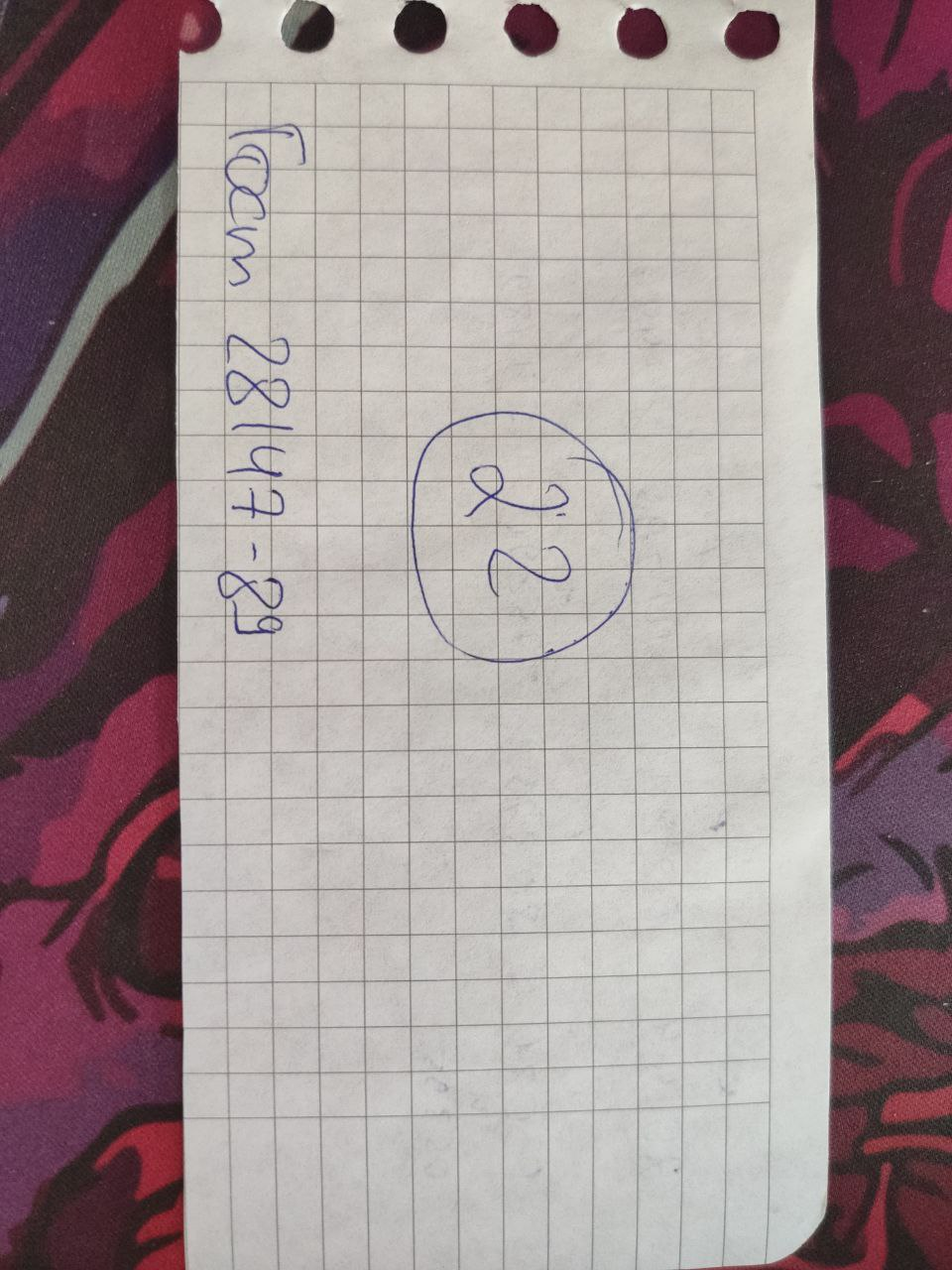
Расшифрованный текст:

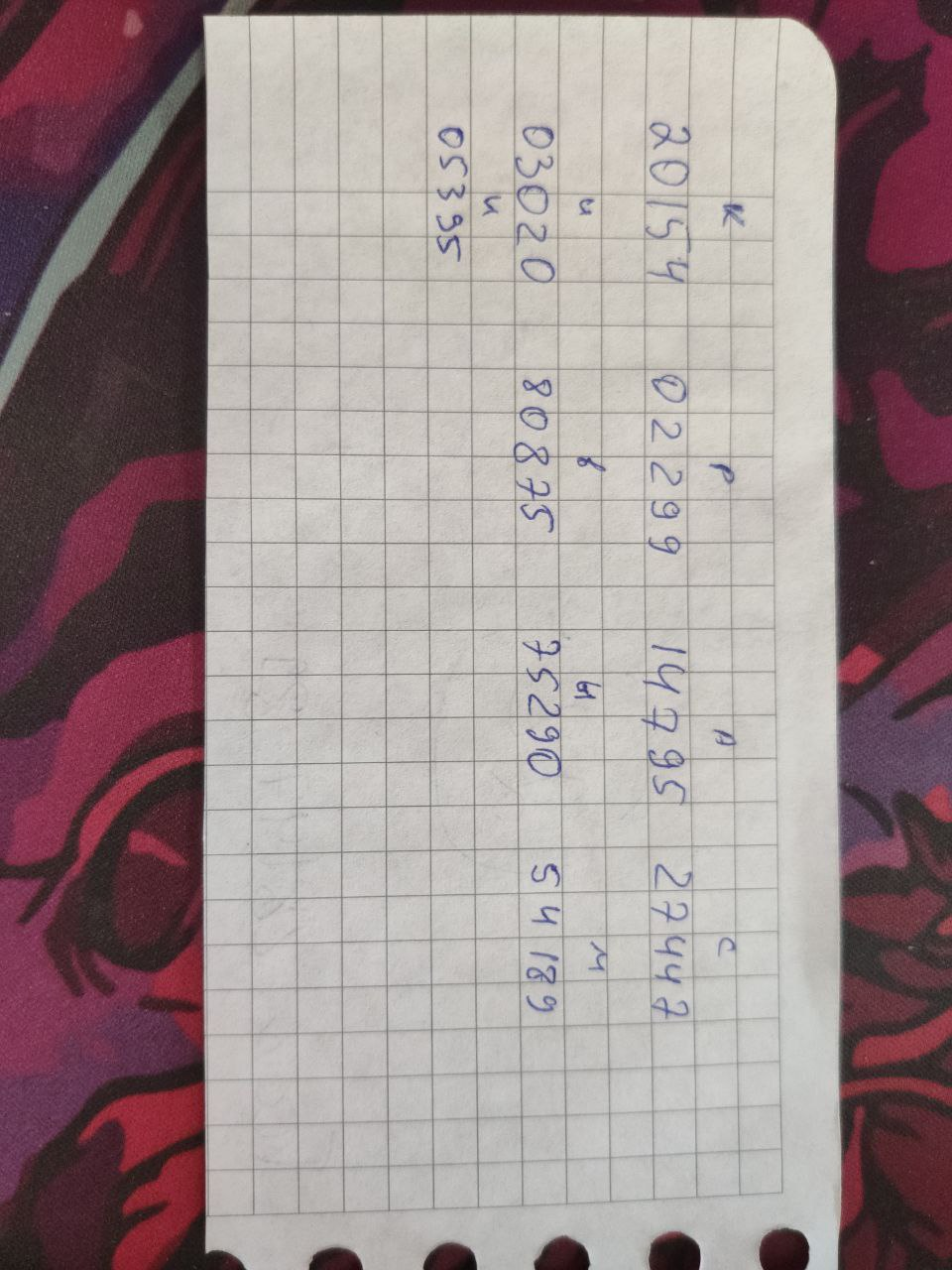
{output\_from\_decrypted(dec\_txt)}

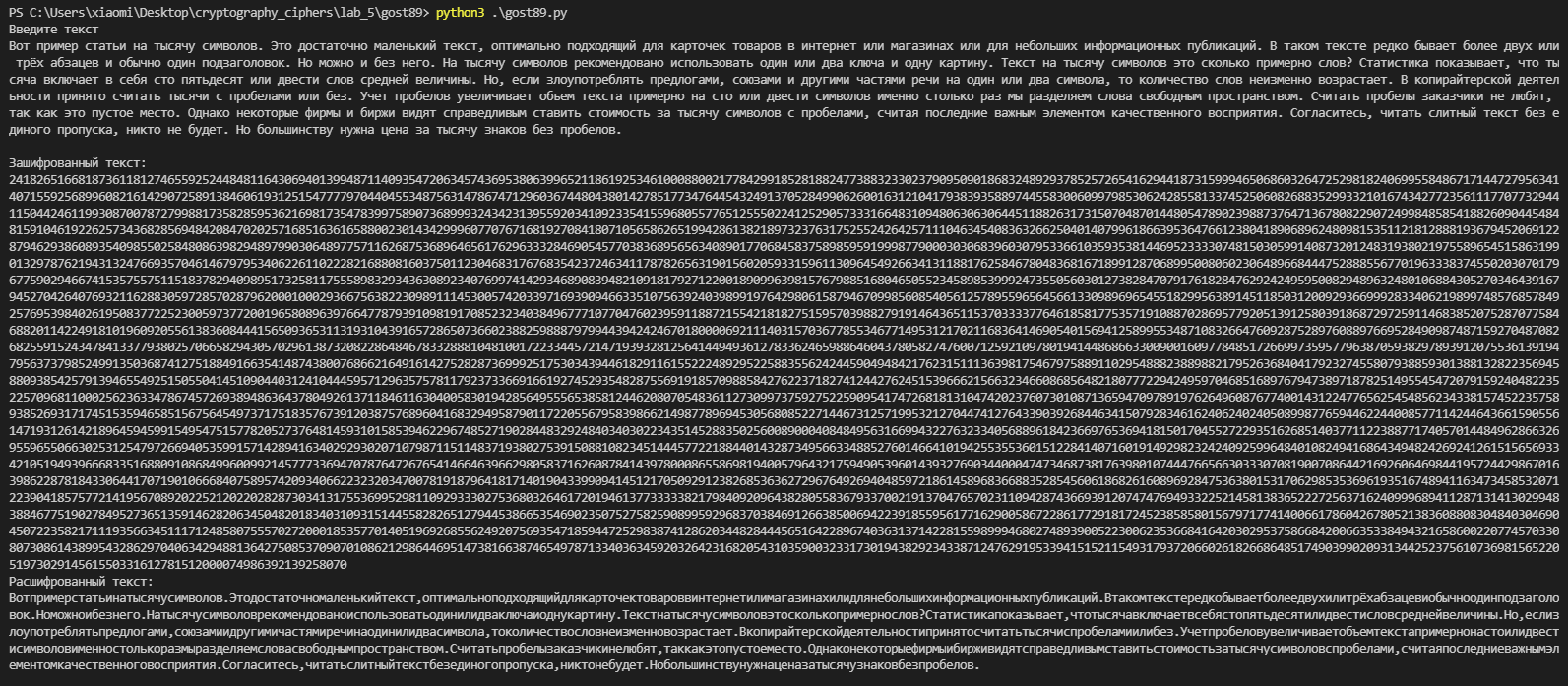
''')

**Тестирование:**









* **Одноразовый блокнот К.Шеннона**

**Код программы:**

import random

alphabet = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя "

alphabet = alphabet.replace(' ', '')

alphabet\_lower = {}

i = 0

while i < (len(alphabet)):

    alphabet\_lower.update({alphabet[i]: i})

    i += 1

def get\_key(d, value):

    for k, v in d.items():

        if v == value:

            return k

def encode(msg):

    msg\_list = list(msg)

    msg\_list\_len = len(msg\_list)

    msg\_code\_bin\_list = list()

    for i in range(len(msg\_list)):

        msg\_code\_bin\_list.append(alphabet\_lower.get(msg\_list[i]))

    key\_list = list()

    for i in range(msg\_list\_len):

        key\_list.append(random.randint(0, 32))

    cipher\_list = list()

    for i in range(msg\_list\_len):

        m = int(msg\_code\_bin\_list[i])

        k = int(key\_list[i])

        cipher\_list.append(int(bin(m ^ k), base=2))

    return cipher\_list, key\_list

def decode(msg, key\_list):

    decipher\_list = list()

    msg\_list\_len = len(msg)

    for i in range(msg\_list\_len):

        c = int(msg[i])

        k = int(key\_list[i])

        decipher\_list.append(int(bin(c ^ k), base=2))

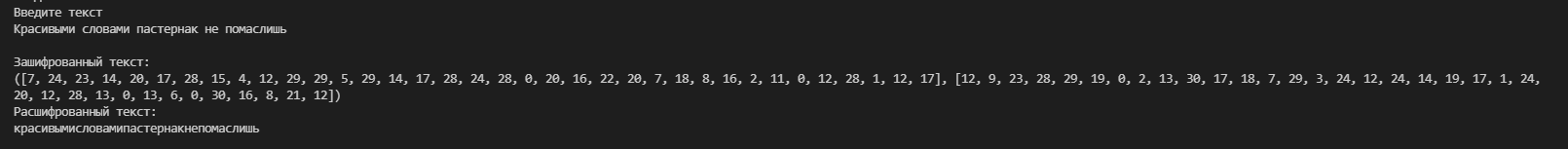
    deciphered\_str = ""

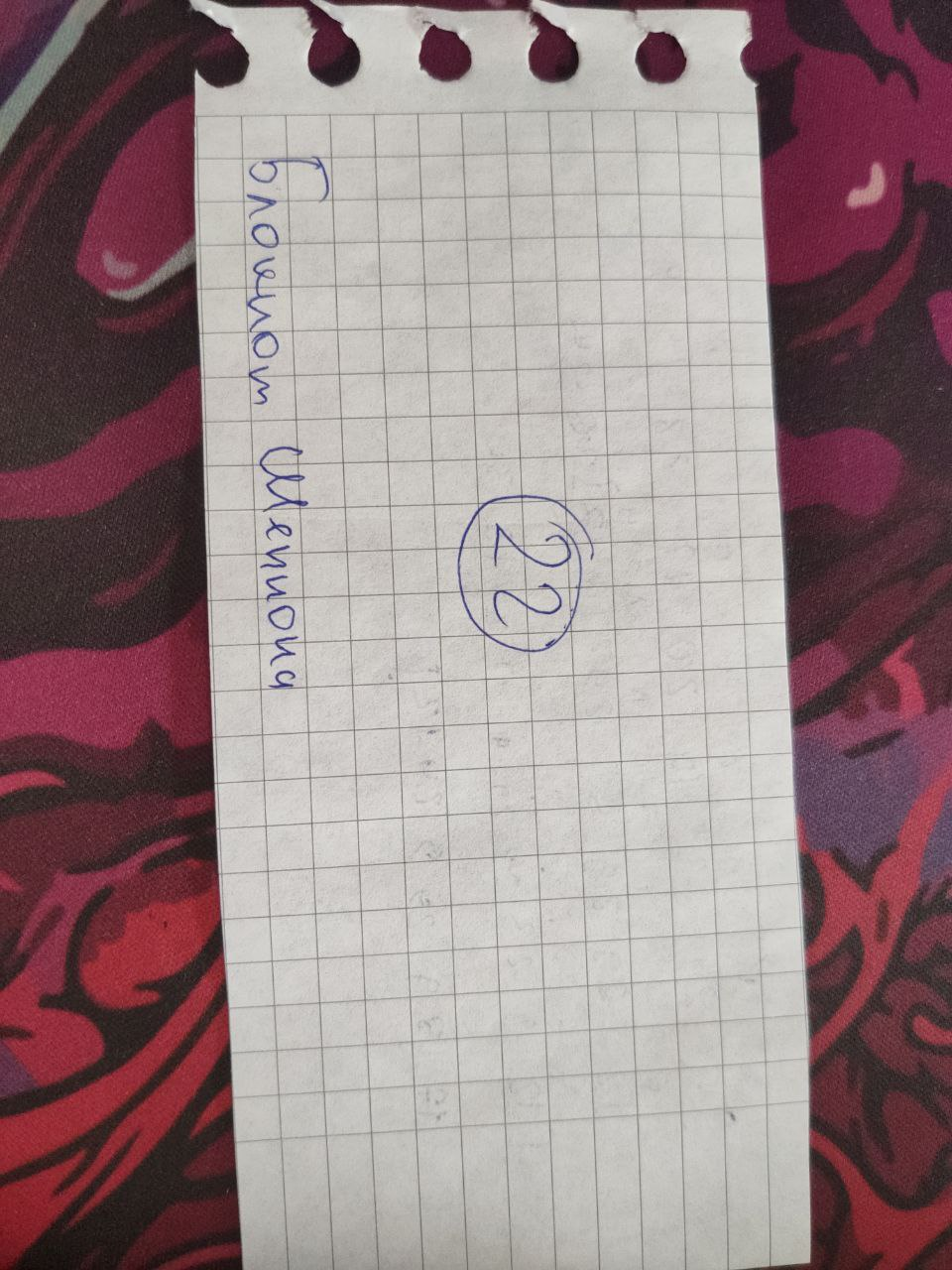
    for i in range(len(decipher\_list)):

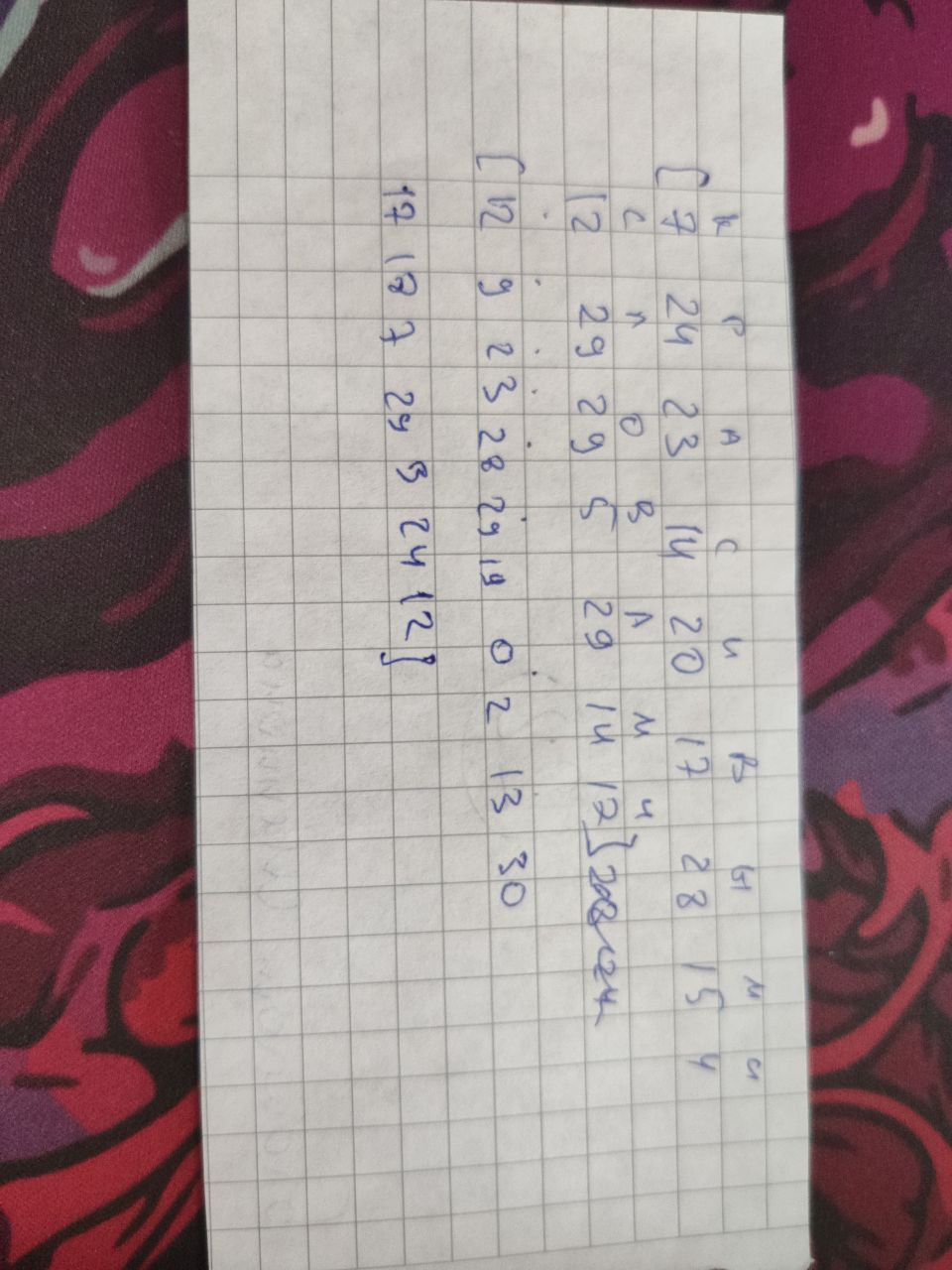
        deciphered\_str += get\_key(alphabet\_lower, decipher\_list[i])

    return deciphered\_str

**Тестирование:**

****





****