Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Московский политехнический университет»

(Московский политех)

Отчёт по курсу «Программирование криптографических алгоритмов»

Лабораторная работа 1.2. Шифрование шифрами однозначной замены



Выполнил:

Студент группы 221-352

Иванов В. В.

Проверил преподаватель: Бутакова Н. Г.

Москва 2024г.

**Аннотация**

* **Среда программирования**
  + Visual Studio Code
* **Язык программирования**
  + Python
* **Процедуры для запуска программы** 
  + Visual Studio Code (main.py)
* **Пословица-тест**
  + Тот, кто ложится на два стула, падает на ребра.
* **Текст для проверки работы (не меньше 1000 знаков (1430))**

Жизнь - это удивительное приключение, полное разнообразных событий и встреч. В каждом моменте мы находим что-то новое и уникальное. Стремление к росту и саморазвитию вдохновляет нас на поиск новых горизонтов. Важно помнить, что каждый шаг вперед приносит с собой уроки и опыт.

Разнообразие культур, языков и традиций делает наш мир удивительно богатым. Общение с людьми разных национальностей расширяет кругозор, позволяя нам понимать и уважать друг друга. Взаимное уважение и терпимость создают основу для гармоничного сосуществования.

Природа тоже играет важную роль в нашей жизни. Красота закатов, шум океана, пение птиц - все это напоминает нам о величии мира природы. Забота о окружающей среде становится неотъемлемой частью ответственного образа жизни.

Работа и творчество придают смысл нашим усилиям. Стремление к достижению целей мотивирует нас на новые начинания. Каждый проект, даже самый маленький, приносит удовлетворение и чувство выполненного долга.

Семья и друзья являются надежной опорой в нашей жизни. Обмен историями, веселые посиделки и поддержка в трудные моменты создают теплую атмосферу взаимопонимания и любви.

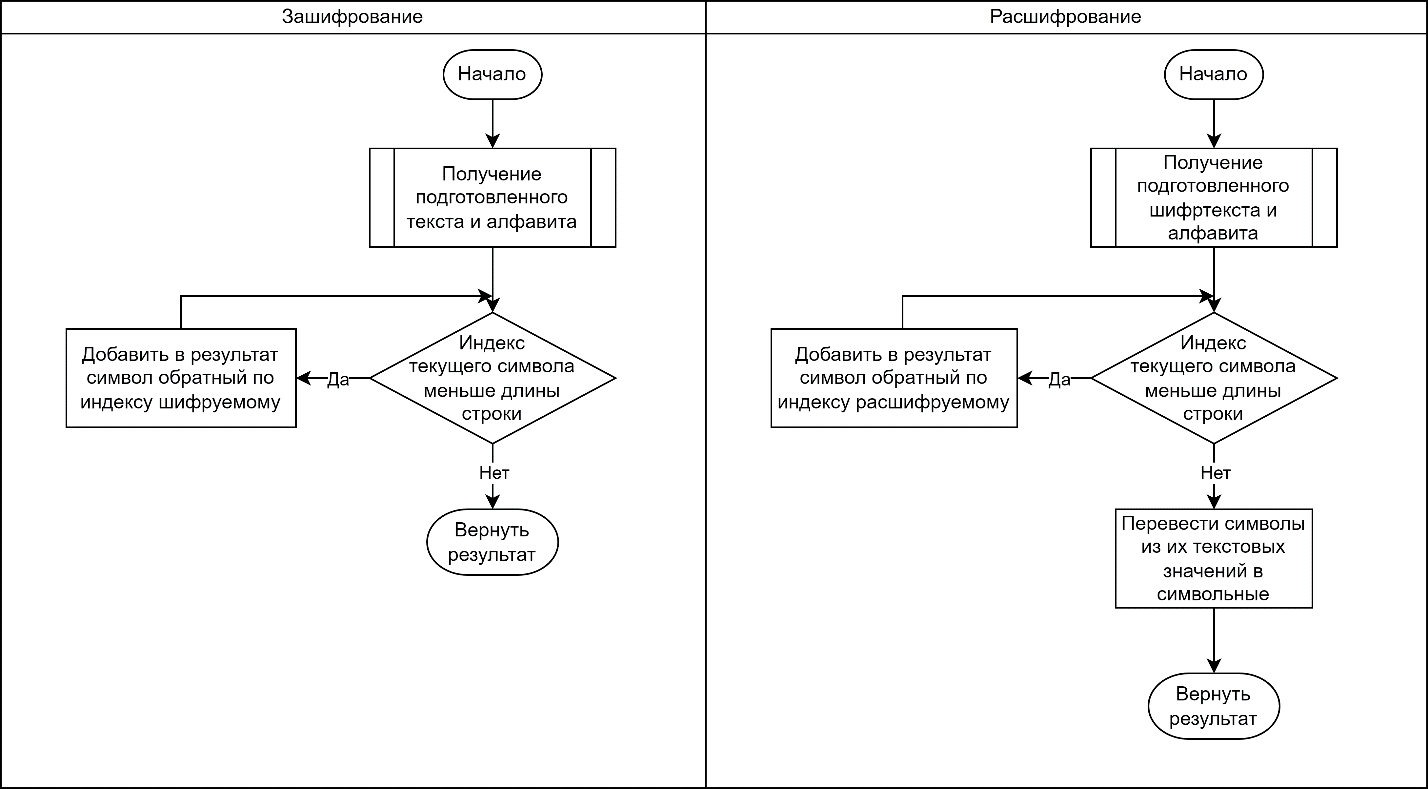
Таким образом, наша жизнь - это мозаика различных моментов, соединенных воедино. Важно ценить каждый момент и стремиться делать мир вокруг нас ярче и лучше. С любовью, терпением и целеустремленностью мы можем создавать свою уникальную историю, наполненную смыслом и радостью.

* **Код программы-интерфейса**
* import sys
* from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QComboBox, QTextEdit
* from atbash import atbash\_encrypt, atbash\_decrypt
* from cesar import cesar\_encrypt, cesar\_decrypt, cesar\_check\_parameters
* from polibia import polibia\_encrypt, polibia\_decrypt
* from tritemiy import tritemiy\_encrypt, tritemiy\_decrypt
* from belazo import belazo\_encrypt, belazo\_decrypt, belazo\_check\_parameters
* from vigener import vigener\_encrypt, vigener\_decrypt, vigener\_check\_parameters
* from matrix import matrix\_encrypt, matrix\_decrypt, matrix\_check\_parameters
* from playfair import playfair\_encrypt, playfair\_decrypt, playfair\_check\_parameters
* available\_ciphers = [
* "Шифр АТБАШ", "Шифр Цезаря", "Шифр Полибия",
* "Шифр Тритемия", "Шифр Белазо", "Шифр Виженера",
* "Шифр Матричный", "Шифр Плейфера",
* ]
* alphabet = [
* "а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к", "л", "м",
* "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ",
* "ъ", "ы", "ь", "э", "ю", "я"
* ]
* alphabet\_polibia = [
* ["а", "б", "в", "г", "д", "е"],
* ["ж", "з", "и", "й", "к", "л"],
* ["м", "н", "о", "п", "р", "с"],
* ["т", "у", "ф", "х", "ц", "ч"],
* ["ш", "щ", "ъ", "ы", "ь", "э"],
* ["ю", "я"]
* ]
* alphabet\_playfair = [
* "а", "б", "в", "г", "д", "е", "ж", "з", "и", "к", "л", "м", "н",
* "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ь",
* "ы", "э", "ю", "я"
* ]
* mem = {
* "bigTextFlag": False,
* "vigenerSwitch": False,
* "mode": "encrypt",
* }
* class CipherApp(QWidget):
* def \_\_init\_\_(self):
* super().\_\_init\_\_()
* self.initUI()
* def initUI(self):
* self.setWindowTitle('Шифры')
* layout = QVBoxLayout()
* *# Выбор шифра*
* cipher\_layout = QHBoxLayout()
* cipher\_label = QLabel('Выберите шифр:')
* self.cipher\_combo = QComboBox()
* self.cipher\_combo.addItems(available\_ciphers)
* cipher\_layout.addWidget(cipher\_label)
* cipher\_layout.addWidget(self.cipher\_combo)
* *# Ввод открытого текста*
* open\_text\_label = QLabel('Введите открытый текст:')
* self.open\_text\_edit = QTextEdit()
* *# Ввод зашифрованного текста*
* cipher\_text\_label = QLabel('Введите зашифрованный текст:')
* self.cipher\_text\_edit = QTextEdit()
* *# Ввод сдвига для шифра Цезаря*
* self.cesar\_shift\_edit = QLineEdit()
* self.cesar\_shift\_edit.setPlaceholderText('Введите сдвиг для шифра Цезаря')
* self.cesar\_shift\_edit.textChanged.connect(self.check\_cesar\_shift)
* *# Ввод ключевого слова для шифра Белазо или Плейфера*
* self.keyword\_edit = QLineEdit()
* self.keyword\_edit.setPlaceholderText('Введите ключевое слово для шифра Белазо или Плейфера')
* *# Ввод ключевой буквы для шифра Виженера*
* self.vigener\_key\_edit = QLineEdit()
* self.vigener\_key\_edit.setPlaceholderText('Введите ключевую букву для шифра Виженера')
* self.vigener\_key\_edit.textChanged.connect(self.check\_vigener\_key)
* *# Ввод ключевой матрицы для шифра Матричный*
* self.matrix\_edit = QLineEdit()
* self.matrix\_edit.setPlaceholderText('Введите ключевую матрицу для шифра Матричный')
* *# Режим работы шифра (шифрование или дешифрование)*
* mode\_layout = QHBoxLayout()
* mode\_label = QLabel('Выберите режим:')
* self.mode\_combo = QComboBox()
* self.mode\_combo.addItems(['Шифрование', 'Дешифрование'])
* mode\_layout.addWidget(mode\_label)
* mode\_layout.addWidget(self.mode\_combo)
* *# Кнопка для запуска шифрования/дешифрования*
* self.encrypt\_button = QPushButton('Выполнить')
* layout.addLayout(cipher\_layout)
* layout.addWidget(open\_text\_label)
* layout.addWidget(self.open\_text\_edit)
* layout.addWidget(cipher\_text\_label)
* layout.addWidget(self.cipher\_text\_edit)
* layout.addWidget(self.cesar\_shift\_edit)
* layout.addWidget(self.keyword\_edit)
* layout.addWidget(self.vigener\_key\_edit)
* layout.addWidget(self.matrix\_edit)
* layout.addLayout(mode\_layout)
* layout.addWidget(self.encrypt\_button)
* self.setLayout(layout)
* *# Подключение слотов к сигналам*
* self.encrypt\_button.clicked.connect(self.cipher\_parser)
* def check\_cesar\_shift(self):
* shift\_text = self.cesar\_shift\_edit.text()
* try:
* shift = int(shift\_text)
* if shift < 0 or shift >= len(alphabet):
* self.cesar\_shift\_edit.setStyleSheet("QLineEdit { color: red; }")
* else:
* self.cesar\_shift\_edit.setStyleSheet("")
* except ValueError:
* self.cesar\_shift\_edit.setStyleSheet("QLineEdit { color: red; }")
* def check\_vigener\_key(self):
* key\_text = self.vigener\_key\_edit.text()
* if len(key\_text) != 1 or key\_text.lower() not in alphabet:
* self.vigener\_key\_edit.setStyleSheet("QLineEdit { color: red; }")
* else:
* self.vigener\_key\_edit.setStyleSheet("")
* def text\_preparation(self, text):
* bigTextFlag = False  *# TODO:*
* if bigTextFlag:
* return text.replace("ё", "е").replace(".", "тчк").replace(",", "зпт").replace("-", "тире").replace(" ", "прбл").replace(":", "двтч").replace(";", "тчсзп").replace("(", "отскб").replace(")", "зкскб").replace("?", "впрзн").replace("!", "восклзн").replace("\n", "првст").lower()
* else:
* return text.replace("ё", "е").replace(".", "тчк").replace(",", "зпт").replace("-", "тире").replace(" ", "").replace(":", "").replace(";", "").replace("(", "").replace(")", "").replace("?", "").replace("!", "").replace("\n", "").lower()
* def cipher\_parser(self):
* cipher\_choose\_input = self.cipher\_combo.currentText()
* open\_text\_input = self.open\_text\_edit.toPlainText()
* cipher\_text\_input = self.cipher\_text\_edit.toPlainText()
* cesar\_shift = self.cesar\_shift\_edit.text()
* keyword = self.keyword\_edit.text()
* vigener\_keyletter = self.vigener\_key\_edit.text()
* matrix\_input = self.matrix\_edit.text()
* *# Определение режима работы (шифрование или дешифрование)*
* mode = 'encrypt' if self.mode\_combo.currentText() == 'Шифрование' else 'decrypt'
* *# Определение флага для обработки больших текстов*
* bigTextFlag = len(open\_text\_input) > 1000  *#ваш порог длины текста*
* if cipher\_choose\_input == "Шифр АТБАШ":
* if mode == "encrypt":
* cipher\_text\_input = atbash\_encrypt(self.text\_preparation(open\_text\_input), alphabet)
* elif mode == "decrypt":
* open\_text\_input = atbash\_decrypt(cipher\_text\_input, alphabet)
* elif cipher\_choose\_input == "Шифр Цезаря":
* if cesar\_shift:  *# Проверка на пустую строку*
* cesar\_shift = int(cesar\_shift)
* if cesar\_check\_parameters(cesar\_shift, alphabet):
* if mode == "encrypt":
* cipher\_text\_input = cesar\_encrypt(self.text\_preparation(open\_text\_input), cesar\_shift, alphabet)
* elif mode == "decrypt":
* open\_text\_input = cesar\_decrypt(cipher\_text\_input, cesar\_shift, alphabet)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода сдвига"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода сдвига"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите сдвиг для шифра Цезаря"
* elif mode == "decrypt":
* open\_text\_input = "Введите сдвиг для шифра Цезаря"
* elif cipher\_choose\_input == "Шифр Полибия":
* if mode == "encrypt":
* cipher\_text\_input = polibia\_encrypt(self.text\_preparation(open\_text\_input), alphabet\_polibia)
* elif mode == "decrypt":
* open\_text\_input = polibia\_decrypt(cipher\_text\_input, alphabet\_polibia)
* elif cipher\_choose\_input == "Шифр Тритемия":
* if mode == "encrypt":
* cipher\_text\_input = tritemiy\_encrypt(self.text\_preparation(open\_text\_input), alphabet)
* elif mode == "decrypt":
* open\_text\_input = tritemiy\_decrypt(cipher\_text\_input, alphabet)
* elif cipher\_choose\_input == "Шифр Белазо":
* if keyword:
* if belazo\_check\_parameters(keyword.lower(), alphabet):
* if mode == "encrypt":
* cipher\_text\_input = belazo\_encrypt(self.text\_preparation(open\_text\_input), keyword.lower(), alphabet)
* elif mode == "decrypt":
* open\_text\_input = belazo\_decrypt(cipher\_text\_input, keyword.lower(), alphabet)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода ключевого слова"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода ключевого слова"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите ключевое слово для шифра Белазо"
* elif mode == "decrypt":
* open\_text\_input = "Введите ключевое слово для шифра Белазо"
* elif cipher\_choose\_input == "Шифр Виженера":
* if vigener\_keyletter:
* if vigener\_check\_parameters(vigener\_keyletter, alphabet):
* if mode == "encrypt":
* *# Надо изменить эту строку*
* cipher\_text\_input = vigener\_encrypt(self.text\_preparation(open\_text\_input), vigener\_keyletter, mode, alphabet)
* elif mode == "decrypt":
* *# И изменить эту строку*
* open\_text\_input = vigener\_decrypt(cipher\_text\_input, vigener\_keyletter, mode, alphabet)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода ключевой буквы"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода ключевой буквы"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите ключевую букву для шифра Виженера"
* elif mode == "decrypt":
* open\_text\_input = "Введите ключевую букву для шифра Виженера"
* elif cipher\_choose\_input == "Шифр Матричный":
* if matrix\_input:
* if matrix\_check\_parameters(matrix\_input):
* if mode == "encrypt":
* cipher\_text\_input = matrix\_encrypt(self.text\_preparation(open\_text\_input), matrix\_input)
* elif mode == "decrypt":
* open\_text\_input = matrix\_decrypt(cipher\_text\_input, matrix\_input)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода матрицы"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода матрицы"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите ключевую матрицу для шифра Матричный"
* elif mode == "decrypt":
* open\_text\_input = "Введите ключевую матрицу для шифра Матричный"
* elif cipher\_choose\_input == "Шифр Плейфера":
* if keyword:
* if playfair\_check\_parameters(keyword, alphabet\_playfair):
* if mode == "encrypt":
* cipher\_text\_input = playfair\_encrypt(self.text\_preparation(open\_text\_input), keyword, alphabet\_playfair)
* elif mode == "decrypt":
* open\_text\_input = playfair\_decrypt(cipher\_text\_input, keyword, alphabet\_playfair)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода ключевого слова"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода ключевого слова"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите ключевое слово для шифра Плейфера"
* elif mode == "decrypt":
* open\_text\_input = "Введите ключевое слово для шифра Плейфера"
* else:
* pass
* *# Обновление текста в виджетах*
* self.open\_text\_edit.setPlainText(open\_text\_input)
* self.cipher\_text\_edit.setPlainText(cipher\_text\_input)
* if \_\_name\_\_ == '\_\_main\_\_':
* app = QApplication(sys.argv)
* ex = CipherApp()
* ex.show()
* sys.exit(app.exec\_())

**1. Шифр простой замены Тритемия**

**Шифр Тритемия** — система шифрования, разработанная [Иоганном Тритемием](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D1%82%D0%B5%D0%BC%D0%B8%D0%B9,_%D0%98%D0%BE%D0%B3%D0%B0%D0%BD%D0%BD). Представляет собой усовершенствованный [шифр Цезаря](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80_%D0%A6%D0%B5%D0%B7%D0%B0%D1%80%D1%8F), то есть [шифр подстановки](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80_%D0%BF%D0%BE%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B8). По алгоритму шифрования, каждый [символ](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB) сообщения смещается на символ, отстающий от данного на некоторый шаг. Здесь шаг смещения делается переменным, то есть зависящим от каких-либо дополнительных факторов. Например, можно задать закон смещения в виде линейной функции (уравнения зашифрования) позиции шифруемой буквы. Сама функция должна гарантировать целочисленное значение. Прямая функция шифрования должна иметь обратную функцию шифрования, тоже целочисленную.

**Блок-схема программы**

****

**Код программы с комментариями**

def tritemiy\_encrypt(open\_text, alphabet):

    encrypted\_text = ""  *# Шифртекст*

    for i in range(len(open\_text)):  *# Проход по всем символам открытого текста*

        element = open\_text[i]  *# Символ*

        encrypted\_text += alphabet[(alphabet.index(element) + i) % len(alphabet)]  *# Добавление в итоговый шифртекст зашифрованного символа*

    return encrypted\_text  *# Возврат шифртекста*

def tritemiy\_decrypt(encrypted\_text, alphabet):

    decrypted\_text = ""  *# Расшифрованный текст*

    for i in range(len(encrypted\_text)):  *# Проход по всем символам шифртекста*

        element = encrypted\_text[i]  *# Символ*

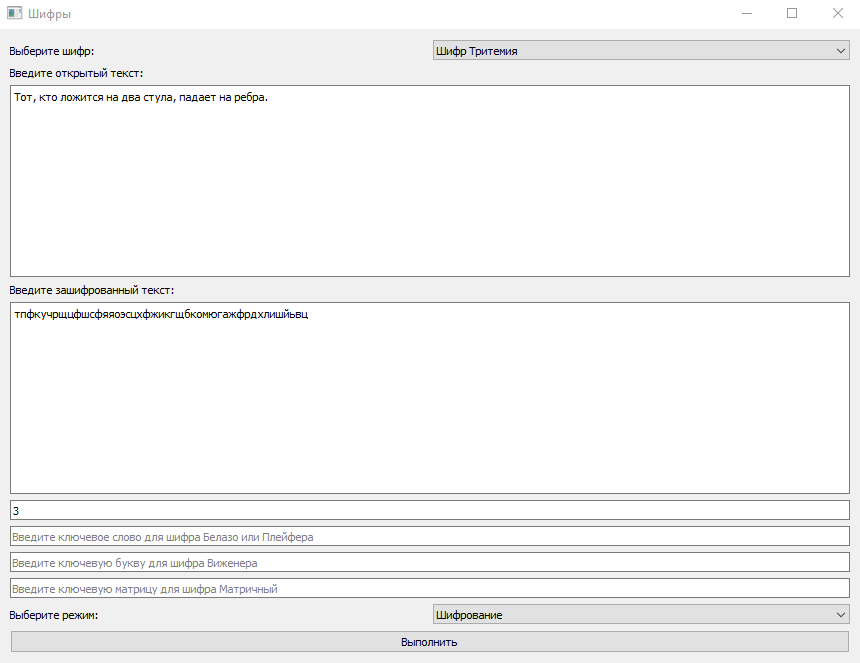
        decrypted\_text += alphabet[(alphabet.index(element) - i % len(alphabet) + len(alphabet)) % len(alphabet)]  *# Добавление в итоговый текст расшифрованного символа*

*# Перевод символов из их текстовых значений в символьные*

    decrypted\_text = decrypted\_text.replace("тчк", ".").replace("зпт", ",").replace("тире", "-").replace('прбл', ' ').replace('двтч', ':').replace('тчсзп', ';').replace('отскб', '(').replace('зкскб', ')').replace('впрзн', '?').replace('восклзн', '!').replace('првст', '\n')

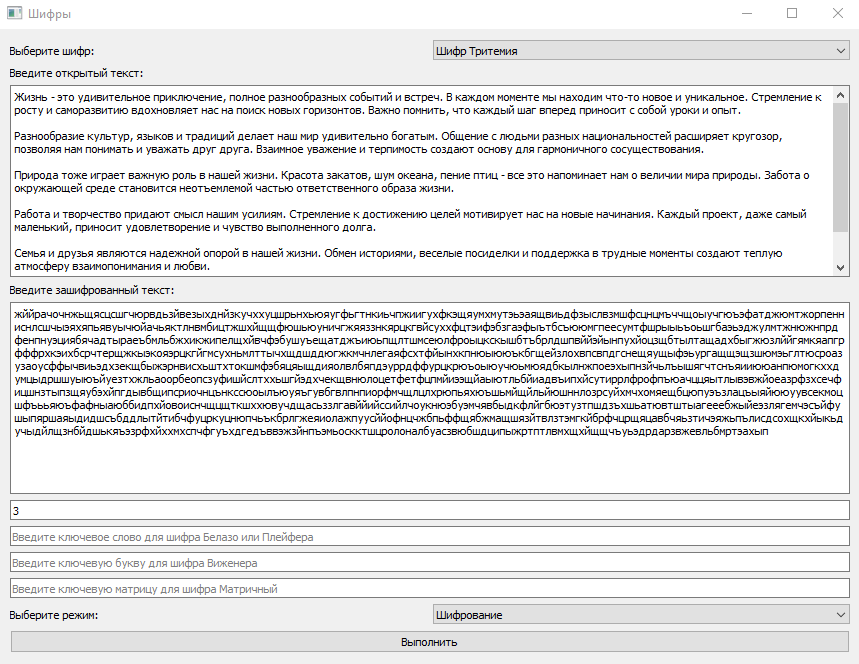
    return decrypted\_text  *# Возврат расшифрованного текста*

**Тестирование**

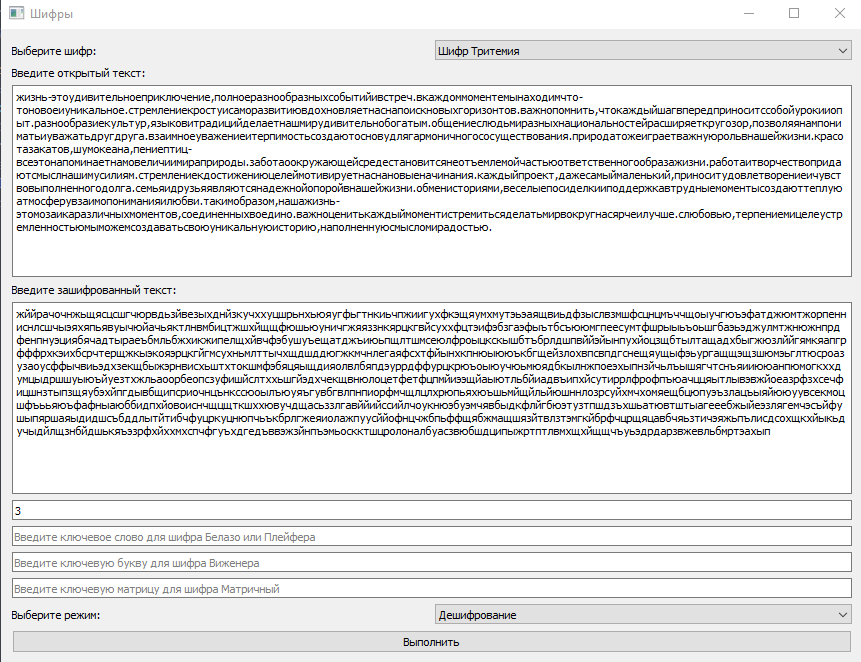


**Работа с текстом не менее 1000 знаков**

**Зашифрование**



**Расшифрование**

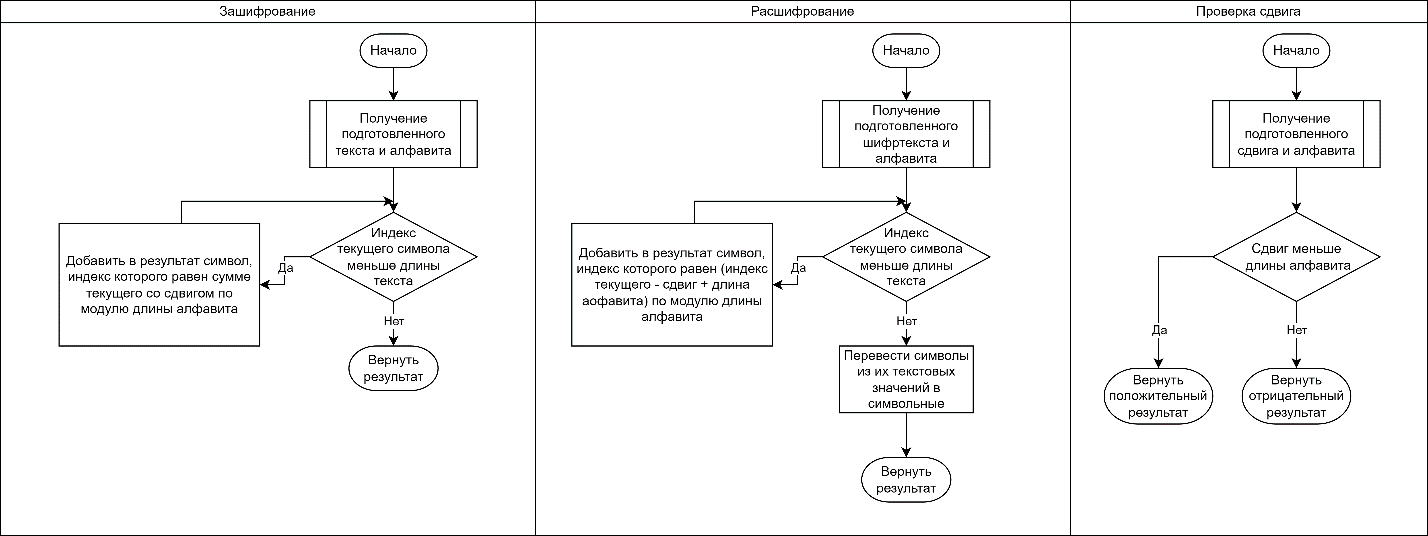


**2. Шифр Белазо**

Шифр Цезаря — это шифр, при использовании которого каждая буква из открытого текста заменяется на такую букву, которая в алфавите находится на некотором постоянном числе позиций левее или правее от рассматриваемой буквы.

Например, при сдвиге букв русского алфавита вправо на 3 позиции буква «А» заменяется на букву «Г», буква «Б» заменяется на букву «Д» и так далее

**Блок-схема программы**

****

**Код программы с комментариями**

def belazo\_check\_parameters(keyword, alphabet):

    if not keyword:

        return False  *# Возврат лжи, если ключевое слово не введено*

    for keyletter in keyword:

        if keyletter not in alphabet:

            return False  *# Возврат лжи, если в ключевом слове присутствуют недопустимые символы*

    return True  *# Возврат истины, если ключевое слово соответствует требованиям*

def belazo\_encrypt(open\_text, keyword, alphabet):

    encrypted\_text = ""  *# Шифртекст*

    for i in range(len(open\_text)):  *# Проход по всем символам открытого текста*

        element = open\_text[i]  *# Символ*

        encrypted\_text += alphabet[(alphabet.index(element) + alphabet.index(keyword[i % len(keyword)])) % len(alphabet)]  *# Добавление в итоговый шифртекст зашифрованного символа*

    return encrypted\_text  *# Возврат шифртекста*

def belazo\_decrypt(encrypted\_text, keyword, alphabet):

    decrypted\_text = ""  *# Расшифрованный текст*

    for i in range(len(encrypted\_text)):  *# Проход по всем символам шифртекста*

        element = encrypted\_text[i]  *# Символ*

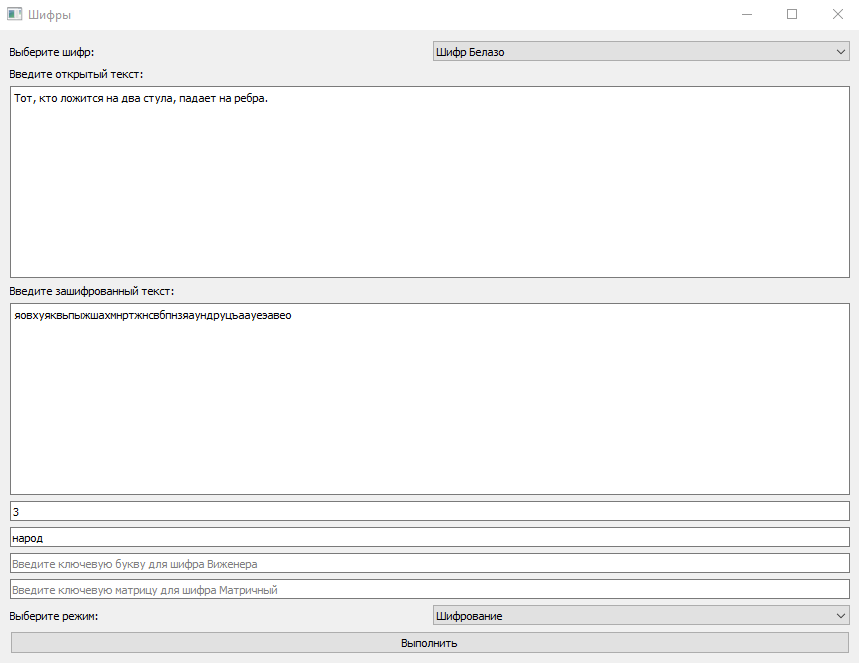
        decrypted\_text += alphabet[(alphabet.index(element) - alphabet.index(keyword[i % len(keyword)]) + len(alphabet)) % len(alphabet)]  *# Добавление в итоговый текст расшифрованного символа*

*# Перевод символов из их текстовых значений в символьные*

    decrypted\_text = decrypted\_text.replace("тчк", ".").replace("зпт", ",").replace("тире", "-").replace('прбл', ' ').replace('двтч', ':').replace('тчсзп', ';').replace('отскб', '(').replace('зкскб', ')').replace('впрзн', '?').replace('восклзн', '!').replace('првст', '\n')

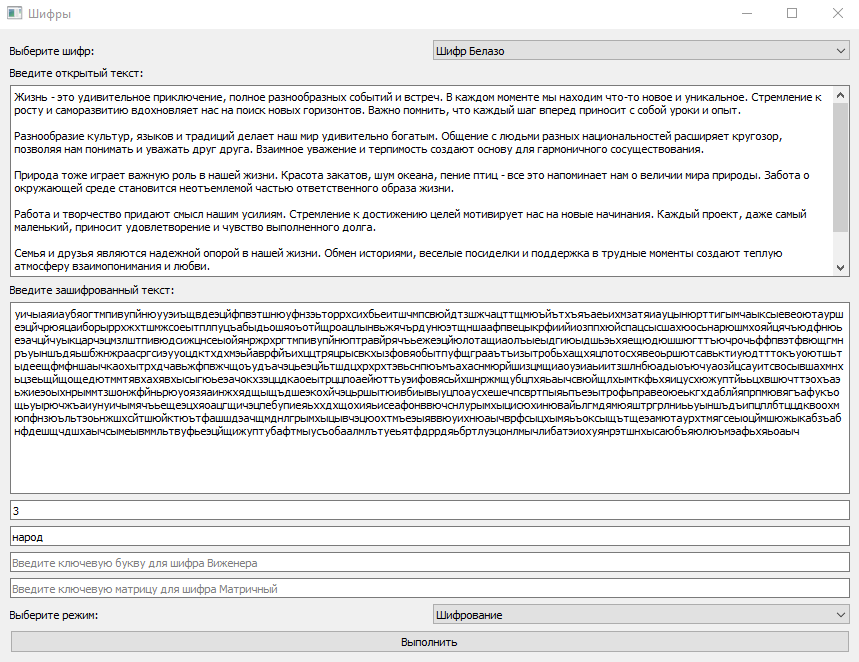
    return decrypted\_text  *# Возврат расшифрованного текста*

**Тестирование**

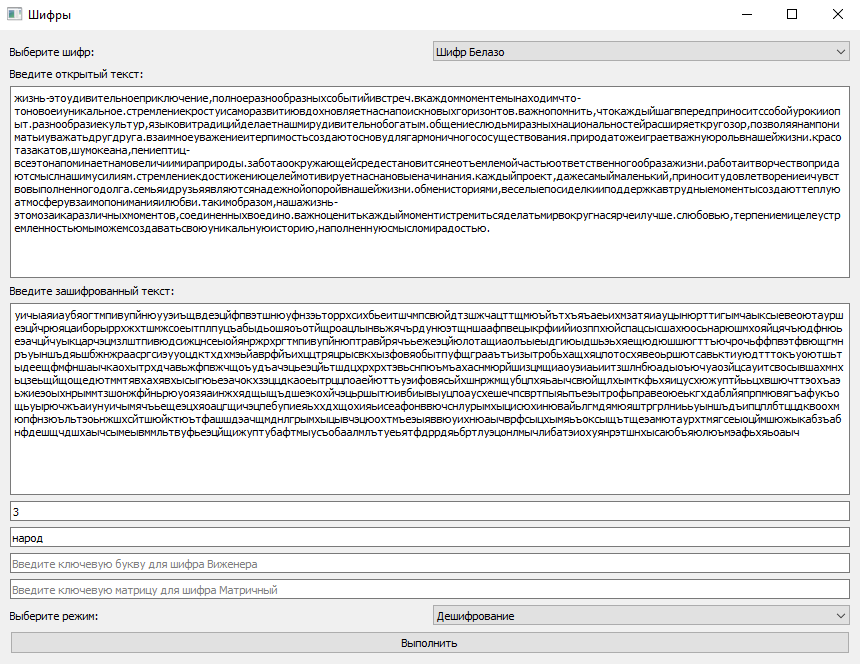


**Работа с текстом не менее 1000 знаков**

**Зашифрование**



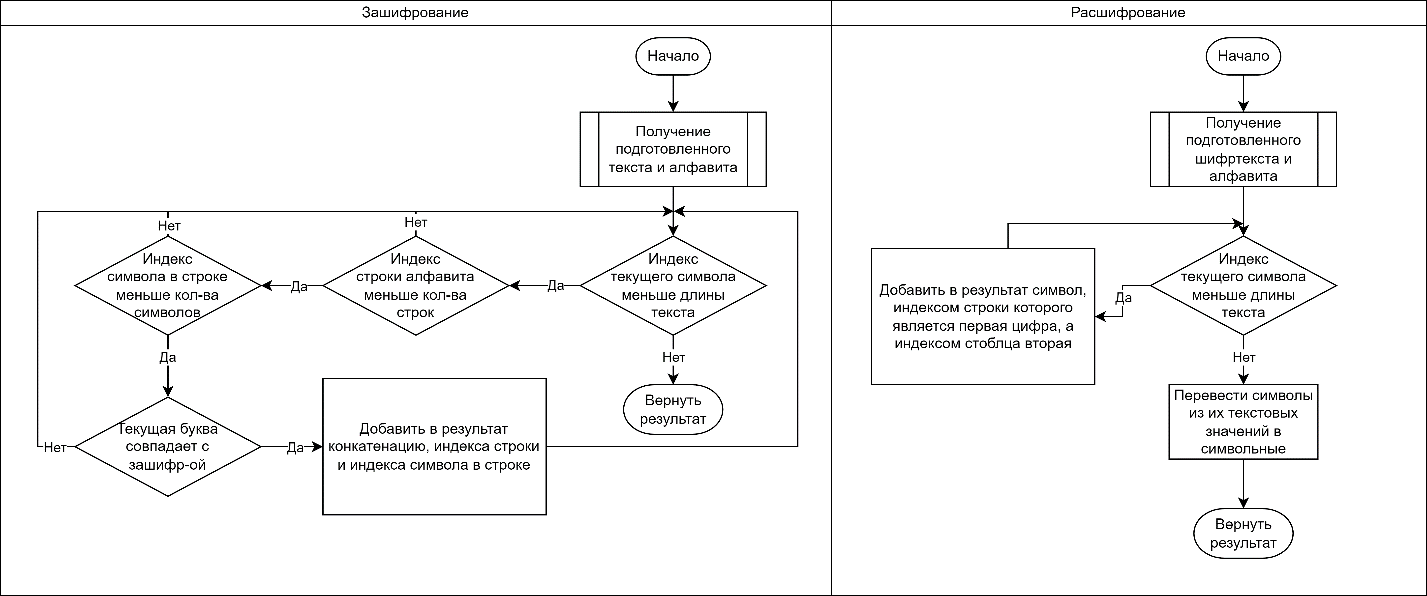
**Расшифрование**



**3. Шифр Виженера**

Шифр Виженера — метод полиалфавитного шифрования буквенного текста с использованием ключевого слова. Этот метод является простой формой многоалфавитной замены. Шифр Виженера изобретался многократно.

**Блок-схема программы**

****

**Код программы с комментариями**

def vigener\_check\_parameters(key\_letter, alphabet):

    if not key\_letter:

        return False  *# Возврат лжи, если ключевая буква не введена*

    if len(key\_letter) == 1 and key\_letter in alphabet:

        return True  *# Возврат истины, если ключевая буква соответствует требованиям*

    return False  *# Возврат лжи, если в ключевой букве более одного символа или некорректное значение буквы*

def vigener\_encrypt(open\_text, key\_letter, mode, alphabet):

    encrypted\_text = ""  *# Шифртекст*

    keyword = key\_letter

    for i in range(len(open\_text)):  *# Проход по всем символам открытого текста*

        element = open\_text[i]  *# Символ*

        encrypted\_text += alphabet[(alphabet.index(element) + alphabet.index(keyword[i % len(keyword)])) % len(alphabet)]  *# Добавление в итоговый шифртекст зашифрованного символа*

        if mode == "selfkey":

            keyword += open\_text[i]

        elif mode == "cipherkey":

            keyword += encrypted\_text[-1]

    return encrypted\_text  *# Возврат шифртекста*

def vigener\_decrypt(encrypted\_text, key\_letter, mode, alphabet):

    decrypted\_text = ""  *# Расшифрованный текст*

    keyword = key\_letter

    for i in range(len(encrypted\_text)):  *# Проход по всем символам шифртекста*

        element = encrypted\_text[i]  *# Символ*

        decrypted\_text += alphabet[(alphabet.index(element) - alphabet.index(keyword[i % len(keyword)]) + len(alphabet)) % len(alphabet)]  *# Добавление в итоговый текст расшифрованного символа*

        if mode == "selfkey":

            keyword += decrypted\_text[-1]  *# добавление к ключу расшифрованной буквы*

        elif mode == "cipherkey":

            keyword += encrypted\_text[i]

*# Перевод символов из их текстовых значений в символьные*

    decrypted\_text = decrypted\_text.replace("тчк", ".").replace("зпт", ",").replace("тире", "-").replace('прбл', ' ').replace('двтч', ':').replace('тчсзп', ';').replace('отскб', '(').replace('зкскб', ')').replace('впрзн', '?').replace('восклзн', '!').replace('првст', '\n')

    return decrypted\_text  *# Возврат расшифрованного текста*

**Тестирование**

**Работа с текстом не менее 1000 знаков**

**Зашифрование**

**Расшифрование**