

# ***Криптография с Python***

*Добро пожаловать!*

## Справочник?

Данный файл является скорее справочником, чем полноценной книгой, т.к. в привычном плане теории здесь не будет. В PDF файле будет разобрано 46 шифров из криптографии в виде программ, 4 программы для помощи в криптоанализе, 3 программы связанные со стеганографией, 2 шифра неописанных в виде программ, 3 программы реализующие кодировки и 2 дополнительные программы.

## Для кого создан этот справочник?

Этот справочник скорее создан для тех людей, кто хочет изучать Криптографию, но знают при этом лишь программирование, либо же наоборот, хотят изучать Программирование, но знают при этом лишь криптографию.

## Почему именно Python?

Python идеальный язык для работы со строками, а также различного рода вычислений. Из-за синтаксической лёгкости языка у нас будут получаться достаточно красивые и понятные программы.

## Версия Python?

Version – 3.6

## Обозначения синтаксиса?

<b>Black</b>	– переменные/константы
<b>Gray</b>	– комментарии
<b>Purple</b>	– строки
<b>Red</b>	– операторы / регулярные выражения
<b>Blue</b>	– функции / методы / исключения
<b>Green</b>	– объявление функций / булевы значения / регулярные выражения
<b>Orange</b>	– аргументы

## Частые обозначения в справочнике?

<b><u>TITLE</u></b>	– заголовков главы
<b>// main.py</b>	– python скрипт
<b>// *name*.exp*</b>	– файл какого-либо расширения.
<b>// terminal</b>	– ввод и вывод из под терминала
<b>START_[ ... ]_END</b>	– начало и конец программы / файла / терминала
<b>\$ python main.py</b>	– вызов скрипта.
<b>001, 002, 003, ...</b>	– нумерация объектов в главах

### Какие шифры находятся в этом справочнике?

001 шифр Цезаря	024 шифр Вернама (014)
002 шифр ROT13	025 степени шифра Виженера
003 шифр Тритемиуса	026 шифр Штакетник
004 шифр замены	027 шифр решётки
005 шифр Атбаш	028 шифр с омофонами
006 шифр A1Z26	029 невидимый шифр
007 шифр Бэкона	030 индексированный шифр
008 шифр пар	031 шифровальная машина <Турех>
009 тарабарская грамота	032 шифр двойной цифири
010 шифр Полибия	033 шифр Плейфера
011 шифр Виженера	034 шифр ADFGVX
012 шифр Вернама (001)	035 AES шифрование (модуль)
013 книжный шифр	036 Великий Шифр
014 XOR шифрование	037 RSA шифрование (модуль)
015 шифр с ложными символами	038 аффинный шифр
016 шифр Гронсфельда	039 шифр Хилла 2x2
017 псевдосимвольный шифр	040 шифр Хилла 3x3
018 кодирование	041 протокол Диффи-Хеллмана
019 (001) с ключевым словом	042 RSA шифрование (механизм работы)
020 шифр замены слогов	043 цифровая подпись
021 шифр Порты	044 RSA шифрование (цифровая подпись)
022 роторное шифрование	045 шифр с лавинным эффектом
023 криптографич. Хеш-функции	046 шифр транспонированной матрицы

### Какие программы для криптоанализа находятся в этом справочнике?

101 BruteForce шифра Цезаря	103 BruteForce крипт.хеш-функции
102 Частотный криптоанализ	104 BruteForce zip-архива

### Какие программы со стеганографией находятся в этом справочнике?

201 Внесение байтов в файл	203 Занесение архива в файл
202 Чтение байтов в файле	...

### Какие шифры, не описанные в виде программ, находятся в этом справочнике?

301 Масонский шифр	302 Пляшущие человечки
--------------------	------------------------

### Какие методы кодирования находятся в этом справочнике?

401 Азбука Морзе	403 Двоичный код
402 ASCII кодировка	...

### Какие дополнительные программы находятся в этом справочнике?

501 Шифровальщик файлов	502 Скрипт для создания .onion сайта
-------------------------	--------------------------------------

## Терминология связанная с криптографией?

**Криптография** – наука, позволяющая скрывать и защищать конфиденциальную информацию от третьих лиц методом шифрования.

**Стеганография** – способ передачи или хранения информации с учётом сохранения в тайне самого факта о существовании этой информации.

**Криптограф** – специалист, который в большей степени сосредоточен на шифровании информации.

**Криптоаналитик** – специалист, который в большей степени сосредоточен на дешифровании информации.

**Криптолог** – специалист по криптографии в целом, объединяющий в себе криптографа и криптоаналитика.

**Шифр** – какая-либо система преобразования текста с ключом для обеспечения секретности передаваемой информации.

**Открытое сообщение** – информация до преобразования в закрытое / зашифрованное сообщение.

**Закрытое сообщение** – информация после преобразования открытого сообщения в нечитаемый текст (шифротекст).

**Ключ** – параметр шифра способный зашифровывать/расшифровывать информацию.

**Кодирование** – процесс преобразования сигнала из формы, удобной для непосредственного использования информации, в форму, удобную для передачи, хранения или автоматической переработки.

**Кодирование (в криптографии)** – процесс преобразования целых слов в какие-либо другие слова или же символы с целью скрыть первоначальный смысл слов.

**Кодирование (в программировании)** – процесс написания программного кода, скриптов, с целью реализации определённого алгоритма на определённом языке программирования.

**Шифрование** – процесс преобразования информации из открытого сообщения в закрытое сообщение при помощи ключа.

**Расшифрование** – процесс преобразования информации из закрытого сообщения в открытое сообщение при помощи ключа.

**Дешифрование** – процесс преобразования информации из закрытого сообщения в открытое сообщение без знания ключа (взлом шифра).

**Симметричное шифрование** – способ шифрования, при котором существует лишь один ключ, способный как шифровать, так и расшифровывать. В основном симметричные шифры расшифровывают информацию зеркально шифрованию.

**Асимметричное шифрование** – способ шифрования, при котором генерируется 2 ключа, один из которых создан только для шифрования информации, второй для расшифрования информации.

**Открытый ключ** – ключ в асимметричных методах шифрования, который используется для шифрования информации.

**Закрытый ключ** – ключ в асимметричных методах шифрования, который используется для расшифрования информации.

## Ключевые слова и базовые определения в Python?

**# Comment** – Однострочный комментарий.

**value = 5** – Объявление целочисленной переменной.

**value = 5.75** – Объявление нецелочисленной переменной.

**value = "Hello"** - Объявление строковой переменной.

**def func(x):** - Объявление функции с параметром 'x'.

**func(5)** – Вызов функции с аргументом '5'.

**lambda x: x** – Лямбда-функция.

**print()** – Функция вывода.

**input()** – Функция ввода.

**try / except** – Исключения, позволяющие предотвращать ошибки.

**raise** – вызов исключения.

**return** – инструкция, возвращающая значение функции.

**if / elif / else** – Условия.

**for / while** – Циклы.

**break** – оператор, прерывающий цикл.

**continue** – оператор, пропускающий итерацию цикла.

**int** – Целочисленные значения. Функция **int()**.

**float** – Числа с плавающим значением (нецелочисленные). Функция **float()**.

**str** – Строки. Функция **str()**.

**chr()** - Функция преобразующая число в символ по ASCII таблице.

**ord()** - Функция преобразующая символ в число по ASCII таблице.

**[]** - Список. Значения списков возможно изменять, а сами списки редактировать.

**()** - Кортеж. Значения кортежей и сами кортежи редактировать нельзя.

**{}** - Множество. Множества не могут хранить одинаковые значения.

**{x:y}** – Словарь. Словари хранят в себе ключ:значение.

**list()** - Функция списка; **tuple()** - Функция кортежа;

**set()** - Функция множества; **dict()** – Функция словаря.

**r""** - “сырая” строка, используется чаще всего в регулярных выражениях.

**and** – И; **or** – ИЛИ; **not** – НЕ.

**True** – Правда; **False** – Ложь.

**=** - Присваивание.

**==** - Равно.

**<** - Меньше.

**>** - Больше.

**+** - сложение.

**-** - вычитание.

**\*** - умножение.

**/** - деление.

**\*\*** - возведение в степень.

**%** - деление по модулю.

**&** - операция И.

**|** - операция ИЛИ.

**^** - операция Исключающее ИЛИ.

**!=** - Не равно.

**<=** - Меньше-равно.

**>=** - Больше-равно.

**+=** прибавить к переменной какое-либо значение.

**-=** вычесть из переменной какое-либо значение.

**\*=** умножить переменную на какое-либо значение.

**/=** разделить переменную на какое-либо значение.

**\*\*=** возвести переменную в какую-либо степень.

**%=** поделить переменную на какой-либо модуль.

**&=** выполнить операцию И над переменной.

**|=** выполнить операцию ИЛИ над переменной.

**^=** выполнить операцию XOR над переменной.

### Дополнения в справочнике?

В конце справочника будут находиться различные ссылки указывающие на статьи, истории, книги связанные с криптографией, а также нераскрытые дела и не дешифрованные сообщения, которые остаются загадкой криптографии по сей день. Приятного чтения.

# Криптография

## 001. Шифр Цезаря.

(Шифр Цезаря – является классическим методом шифрования и одним из самых знаменитых. Принцип шифрования заключён в ключе-позиции по алфавиту.)

// main.py

Start\_

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
try: rotKey = int(input("Write the key: "))
```

```
except ValueError:
```

```
    print("Error: key is not int number")
```

```
    raise SystemExit
```

```
def encryptDecrypt(mode, message, key, final = ""):
```

```
    for symbol in message:
```

```
        if mode == 'E':
```

```
            final += chr((ord(symbol) + key - 13)%26 + ord('A'))
```

```
        else:
```

```
            final += chr((ord(symbol) - key - 13)%26 + ord('A'))
```

```
    return final
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage, rotKey))
```

\_End

// terminal

Start\_

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Write the key: 3
```

```
Final message: KHOORZRUOG
```

```
$ python main.py
[E]ncrypt[D]ecrypt: d
Write the message: KHOORZRUOG
Write the key: 3
Final message: HELLOWORLD
]_End
```

```
# Переключатель режимов шифрования.
```

```
cryptMode = input("[E]ncrypt[D]ecrypt: ").upper()
```

```
# Если cryptMode не будет равняться 'E' ИЛИ 'D', тогда вывести ошибку и
заккрыть программу.
```

```
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
```

```
# Сообщение которое мы хотим зашифровать ИЛИ расшифровать.
```

```
startMessage = input("Write the message: ").upper()
```

```
# Переменная-ключ, если ключ не является числом – выводится ошибка.
```

```
try: rotKey = int(input("Write the key: "))
except ValueError:
    print("Error: key is not int number")
    raise SystemExit
```

```
# Основная функция принимающая аргументы: переключатель, сообщение, ключ.
```

```
def encryptDecrypt(mode, message, key, final = ""):
```

```
# Посимвольный перебор подаваемого функции сообщения.
```

```
for symbol in message:
```

```
# Шифрование сообщения при помощи таблицы ASCII.
```

```
if mode == 'E':
    final += chr((ord(symbol) + key - 13)%26 + ord('A'))
```

```
# Если переключатель равен символу 'D', тогда делать тоже самое что и с 'E' лишь
за исключением вычитания переменной key, а не её сложения!
```

```
else:
    final += chr((ord(symbol) - key - 13)%26 + ord('A'))
```



# Вернуть получившееся сообщение после шифровки/расшифровки.

**return final**

# Вывод получившегося сообщения.

**print("Final message:", encryptDecrypt(cryptMode, startMessage, rotKey))**

## 002. Шифр ROT13.

(Шифр ROT13 – часть шифра Цезаря с позицией 13. Особенность ROT13 заключена в принципе инволюции, при которой не нужен переключатель режимов шифрования/расшифрования.)

// **main.py**

**Start\_**[

message = list(input("Write the message: ").upper())

for symbol in range(len(message)):

    message[symbol] = chr(ord(message[symbol])%26+ord('A'))

print("Final message:", "".join(message))

**End\_**

// **terminal**

**Start\_**[

\$ python main.py

Write the message: helloworld

Final message: URYYBJBEYQ

\$ python main.py

Write the message: URYYBJBEYQ

Final message: HELLOWORLD

**End\_**

# Ввод сообщения для шифрования/расшифрования.

message = list(input("Write the message: ").upper())

# Перебор индексов каждого символа в списке message.

for symbol in range(len(message)):

# Шифрование сообщения при помощи таблицы ASCII.

message[symbol] = chr(ord(message[symbol])%26+ord('A'))

# Вывод получившегося сообщения

print("Final message:", "".join(message))

### 003. Шифр Тритемиуса.

(Шифр Тритемиуса – улучшенный шифр Цезаря в котором ключём является не число, а какая-либо функция. В отличии от шифра Цезаря не подвержен частотному криптоанализу, т.к. шифрует сообщение по индексам.)

// **main.py**

**Start\_**[

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = list(input("Write the message: ").upper())
```

```
for symbol in startMessage:
```

```
    if symbol not in [chr(x) for x in range(65,91)]:
```

```
        startMessage.remove(symbol)
```

```
funcKey = lambda x: x*2
```

```
def encryptDecrypt(mode, message, key, final = ""):
```

```
    for index, symbol in enumerate(message):
```

```
        if mode == 'E':
```

```
            temp = ord(symbol) + key(index) - 13
```

```
        else:
```

```
            temp = ord(symbol) - key(index) - 13
```

```
            final += chr(temp%26 + ord('A'))
```

```
    return final
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage, funcKey))
```

**]\_End**

// **terminal**

**Start\_**[

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Final message: HGPRWGAFBV
```

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: d
```

```
Write the message: HGPRWGAFBV
```

```
Final message: HELLOWORLD
```

**]\_End**

```

# Переключатель режимов шифрования/расшифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit

# Сообщение для шифрования/расшифрования.
startMessage = list(input("Write the message: ").upper())

# Посимвольный перебор сообщения.
for symbol in startMessage:

# Если символ не находится в диапазоне от А до Z, то удалить этот символ.
if symbol not in [chr(x) for x in range(65,91)]:
    startMessage.remove(symbol)

# Ключ-функция, принимающая числовое значение и умножающая его на 2.
funcKey = lambda x: x*2

# Основная функция для шифрования/расшифрования принимающая 4 аргумента
mode, message, key, final.
def encryptDecrypt(mode, message, key, final = ""):

# Посимвольный перебор сообщения. Index – индекс символа в сообщении, Symbol
– символ сообщения.
for index, symbol in enumerate(message):

# Если переключатель равен 'E', тогда присвоить к переменной temp
получившееся числовое значение.
if mode == 'E':
    temp = ord(symbol) + key(index) - 13

# Если переключатель равен символу 'D', тогда делать тоже самое что и с 'E' лишь
за исключением вычитания функции key, а не её сложения!
else:
    temp = ord(symbol) - key(index) - 13

# Присвоить к переменной final получившийся символ.
final += chr(temp%26 + ord('A'))

```

# Вернуть получившееся сообщение.

**return final**

# Вывод получившегося сообщения

**print("Final message:", encryptDecrypt(cryptMode, startMessage, funcKey))**

## 004. Шифр замены.

(Шифр замены – один из самых распространённых методов шифрования. В отличие от шифра Цезаря не имеет конкретного ключа и алгоритма шифрования.)

```
// main.py
Start_[
symbolsAlpha = [chr(x) for x in range(65,91)]
symbolsCrypt = ('!','@','#','$','%','^','&','*','(',')','-', '=', '+', '?', ':', ';', '<', '>', '/', '[', ']', '{', '}', '|', '.,', '~')
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
keys = dict(zip(symbolsAlpha,symbolsCrypt))
def encryptDecrypt(mode, message, final = ""):
    if mode == 'E':
        for symbol in message:
            if symbol in keys: final += keys[symbol]
    else:
        for symbol in message:
            for key in keys:
                if symbol == keys[key]: final += key
    return final
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End
```

### // terminal

```
Start_[
$ python main.py
[E]ncrypt|[D]ecrypt: e
Write the message: helloworld
Final message: *%==:}>=$

$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the message: *%==:}>=$
Final message: HELLOWORLD
]_End
```

```
# Объявление списка символов от 'A' до 'Z'.
symbolsAlpha = [chr(x) for x in range(65,91)]
```

```
# Объявление списка символов для шифровки.
symbolsCrypt = ('!', '@', '#', '$', '%', '^', '&', '*', '(', ')', '-', '=',
'+', '?', ':', ';', '<', '>', '/', '[', ']', '{', '}', '|', '.,', '~')
```

```
# Переключатель для шифрования/расшифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit
```

```
# Сообщение для шифрования/расшифрования.
startMessage = input("Write the message: ").upper()
```

```
# Создание словаря подобного типа → символ_алфавита : символ_шифра.
keys = dict(zip(symbolsAlpha, symbolsCrypt))
```

```
# Объявление функции с 3 аргументами.
def encryptDecrypt(mode, message, final = ""):
```

```
# Если переключатель равен 'E', тогда сделать посимвольный перебор сообщения
и если символ будет находится в словаре keys (символ_алфавита), то к переменной
final прибавить значение ключа (символ_шифра).
```

```
if mode == 'E':
    for symbol in message:
        if symbol in keys: final += keys[symbol]
```

```
# Если переключатель равен 'D', тогда сделать посимвольный перебор сообщения
и перебор всех ключей в словаре keys. И если есть совпадение между символом в
сообщении и значением ключа (символ_шифра), то к переменной final прибавить
сам ключ (символ_алфавита).
```

```
else:
    for symbol in message:
        for key in keys:
            if symbol == keys[key]: final += key
```

```
# Вернуть получившееся сообщение.
return final
```

# Вывод получившегося сообщения.

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```



## 005. Шифр Атбаш.

(Шифр Атбаш является одним из самых лёгких методов шифрования. Создаёт зеркальную версию алфавита, где начало - буква 'Z', а конец алфавита буква - 'A'.)

**// main.py**

**Start\_**

```
message = input("Write the message: ").upper()
alphaDefault = [chr(x) for x in range(65,91)]
alphaReverse = list(alphaDefault); alphaReverse.reverse()
final = ""
for symbolMessage in message:
    for indexAlpha, symbolAlpha in enumerate(alphaDefault):
        if symbolMessage == symbolAlpha:
            final += alphaReverse[indexAlpha]
print("Final message:", final)
End
```

**// terminal**

**Start\_**

\$ python main.py

Write the message: helloworld

Final message: SVOOLDLIOW

\$ python main.py

Write the message: SVOOLDLIOW

Final message: HELLOWORLD

**End**

# Сообщение для шифрования/расшифрования.

```
message = input("Write the message: ").upper()
```

# Создание списка символом от 'A' до 'Z'.

```
alphaDefault = [chr(x) for x in range(65,91)]
```

# Создание списка символом от 'A' до 'Z' с реверсией.

```
alphaReverse = list(alphaDefault); alphaReverse.reverse()
```

# Создание переменной-строки final.

```
final = ""
```

# Посимвольный перебор в сообщении.

**for** symbolMessage **in** message:

# Посимвольный перебор в списке алфавита.

**for** indexAlpha, symbolAlpha **in** enumerate(alphaDefault):

# Если символ сообщения будет равен символу алфавита, тогда к переменной final добавить символ из списка реверсированного алфавита по индексу.

**if** symbolMessage == symbolAlpha:

**final** += alphaReverse[indexAlpha]

# Вывод получившегося сообщения.

**print**("Final message:", final)

## 006. Шифр A1Z26.

(Шифр A1Z26 является одним из самых лёгких методов шифрования. Принцип шифрования заключён в нумерации символа алфавита: A = 1, B = 2 ... Z = 26.)

// **main.py**

**Start\_**[

```
from re import findall
alpha = tuple("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
def regular(text):
    return findall(r"[0-9]+", text)
def encryptDecrypt(mode, message, final = ""):
    if mode == 'E':
        for symbol in message:
            final += "%hu "%(alpha.index(symbol)+1)
    else:
        for number in regular(message):
            final += "%c"%alpha[int(number)-1]
    return final
print("Final message:", encryptDecrypt(cryptMode, startMessage))
]_End
```

// **terminal**

**Start\_**[

\$ python main.py

[E]ncrypt|[D]ecrypt: e

Write the message: helloworld

Final message: 8 5 12 12 15 23 15 18 12 4

\$ python main.py

[E]ncrypt|[D]ecrypt: d

Write the message: 8 5 12 12 15 23 15 18 12 4

Final message: HELLOWORLD

**]**\_End

```
# Импортирование функции findall из регулярных выражений.
from re import findall

# Алфавит.
alpha = tuple("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

# Переключатель режимов шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit

# Открытое / зашифрованное сообщение.
startMessage = input("Write the message: ").upper()

# Функция возвращающая список найденных чисел в тексте.
def regular(text):
    return findall(r"[0-9]+", text)

# Функция шифрования / расшифрования.
def encryptDecrypt(mode, message, final = ""):

# Если переключатель равен 'E', тогда зашифровать сообщение.
if mode == 'E':

# Посимвольный перебор сообщения и занесение индекса алфавита в переменную
final.
for symbol in message:
    final += "%hu "%(alpha.index(symbol)+1)

# Иначе, если переключатель равен 'D', тогда расшифровать сообщение.
else:

# Перебор всех чисел и занесение символа алфавита по индексу в переменную
final.
for number in regular(message):
    final += "%c"%alpha[int(number)-1]

# Вернуть сообщение.
return final
```

# Вывод получившегося сообщения.

```
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```

## 007. Шифр Бэкона.

(Шифр Бэкона является шифром замены, который можно представить в виде двоичного кода, где А = 0, В = 1. Особенность шифра Бэкона заключается в гибкости использования, что позволяет применять его в различных методах стеганографии.)

// main.py

Start\_

```
from re import findall
```

```
keysBacon = {
```

'A': "AAAAA",	'B': "AAAAB",	'C': "AAABA",
'D': "AAABB",	'E': "AABAA",	'F': "AABAB",
'G': "AABBA",	'H': "AABBB",	'I': "ABAAA",
'J': "ABAAB",	'K': "ABABA",	'L': "ABABB",
'M': "ABBA",	'N': "ABBAB",	'O': "ABBBA",
'P': "ABBB",	'Q': "BAAAA",	'R': "BAAAB",
'S': "BAABA",	'T': "BAABB",	'U': "BABAA",
'V': "BABAB",	'W': "BABBA",	'X': "BABBB",
'Y': "BBAAA",	'Z': "BBAAB",	' ': "BBABA"

```
}
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E', 'D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
def regular(text):
```

```
    return findall(r"[A-Z]{5}", text)
```

```
def encryptDecrypt(mode, message, final = ""):
```

```
    if mode == 'E':
```

```
        for symbol in message:
```

```
            if symbol in keysBacon: final += keysBacon[symbol]
```

```
    else:
```

```
        for symbolsFive in regular(message):
```

```
            for key in keysBacon:
```

```
                if symbolsFive == keysBacon[key]: final += key
```

```
    return final
```

```
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```

```
_End
```

**// terminal**

**Start\_**

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Final message:
```

```
AABBBAABAABABBBABABBBABABBAABBBABAAABABABBBAAABB
```

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: d
```

```
Write the message:
```

```
AABBBAABAABABBBABABBBABABBAABBBABAAABABABBBAAABB
```

```
Final message: HELLOWORLD
```

**\_End**

```
# Импортирование функции findall из модуля 'регулярные выражения'.
```

```
from re import findall
```

```
# Словарь вида → символ_алфавита : 5символов_шифра.
```

```
keysBacon = {
```

```
    'A':"AAAAA", 'B':"AAAAB", 'C':"AAABA",  
    'D':"AAABB", 'E':"AABAA", 'F':"AABAB",  
    'G':"AABBA", 'H':"AABBB", 'I':"ABAAA",  
    'J':"ABAAB", 'K':"ABABA", 'L':"ABABB",  
    'M':"ABBA", 'N':"ABBAB", 'O':"ABBBA",  
    'P':"ABBBB", 'Q':"BAAAA", 'R':"BAAAB",  
    'S':"BAABA", 'T':"BAABB", 'U':"BABAA",  
    'V':"BABAB", 'W':"BABBA", 'X':"BABBB",  
    'Y':"BBAAA", 'Z':"BBAAB", ' ': "BBABA"
```

```
}
```

```
# Переключатель шифрования/расшифрования.
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
# Сообщение для шифрования/расшифрования.
```

```
startMessage = input("Write the message: ").upper()
```

# Функция регулярного выражения, помогающая расшифровывать сообщение для зашифрованное сообщение по 5 символов.

```
def regular(text):  
    return findall(r"[A-Z]{5}", text)
```

# Функция с 3 аргументами.

```
def encryptDecrypt(mode, message, final = ""):
```

# Если переключатель равен 'E', то сделать посимвольный перебор сообщения.

```
if mode == 'E':  
    for symbol in message:
```

# Если символ сообщения (символ\_алфавита) будет находится в ключах keysBacon, то к переменной final прибавить значение ключа (5символов\_шифра).

```
if symbol in keysBacon: final += keysBacon[symbol]
```

# Если переключатель равен 'D', то сделать перебор сообщения через регулярное выражение.

```
else:  
    for symbolsFive in regular(message):
```

# Сделать посимвольный перебор ключей в словаре keysBacon.

```
for key in keysBacon:
```

# И если пять символов равны значению ключа (5символов\_шифра), тогда к переменной final добавить сам ключ (символ\_алфавита).

```
if symbolsFive == keysBacon[key]: final += key
```

# Вернуть получившееся сообщение.

```
return final
```

# Вывод получившегося сообщения.

```
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```



## 008. Шифр пар.

(Шифр пар является шифром замены. Особенность данного метода шифрования в том, что есть привязка между двумя символами. То-есть, если в тексте попадаете символ А, тогда он заменяется на В ( $A \rightarrow B$ ), но это также действует и в обратную сторону ( $B \rightarrow A$ ).)

// **main.py**

**Start\_**[

```
keys = {
    'A':'B','C':'D','E':'F','G':'H','I':'J','K':'L',
    'M':'N','O':'P','Q':'R','S':'T','U':'V','W':'X',
    'Y':'Z'}
message = list(input("Write the message: ").upper())
for symbol in range(len(message)):
    for key in keys:
        if message[symbol] == key:
            message[symbol] = keys[key]
        elif message[symbol] == keys[key]:
            message[symbol] = key
        else: pass
print("Final message:", "".join(message))
]_End
```

// **terminal**

**Start\_**[

```
$ python main.py
Write the message: helloworld
Final message: GFKKPXPQKC
```

```
$ python main.py
Write the message: GFKKPXPQKC
Final message: HELLOWORLD
```

**]\_End**

# Создание словаря с привязками символов.

```
keys = {
    'A':'B','C':'D','E':'F','G':'H','I':'J','K':'L',
    'M':'N','O':'P','Q':'R','S':'T','U':'V','W':'X',
    'Y':'Z'}
```

# Сообщение для шифрования/расшифрования.

```
message = list(input("Write the message: ").upper())
```

# Посимвольный перебор сообщения.

```
for symbol in range(len(message)):
```

# Перебор всех ключей в словаре.

```
for key in keys:
```

# Если символ сообщения будет равен ключу, тогда этот символ сообщения будет равен значению ключа.

```
if message[symbol] == key:
```

```
    message[symbol] = keys[key]
```

# Если символ сообщения будет равен значению ключа, тогда этот символ сообщения будет равен ключу.

```
elif message[symbol] == keys[key]:
```

```
    message[symbol] = key
```

```
else: pass
```

# Вывод получившегося сообщения.

```
print("Final message: ", "".join(message))
```

## 009. Тарабарская грамота.

(Тарабарская грамота является своего рода шифром пар. Отличие заключено в замене только согласных символов.)

```
// main.py
Start_[]
keys = {
'B':'Z','C':'X','D':'W','F':'V','G':'T',
'H':'S','J':'R','K':'Q','L':'P','M':'N'}
message = list(input("Write the text: ").upper())
for symbol in range(len(message)):
    for key in keys:
        if message[symbol] == key:
            message[symbol] = keys[key]
        elif message[symbol] == keys[key]:
            message[symbol] = key
        else: pass
print("Final message:", "".join(message))
]_End
```

```
// terminal
Start_[]
$ python main.py
Write the text: helloworld
Final message: SEPPODOJPW

$ python main.py
Write the text: SEPPODOJPW
Final message: HELLOWORLD
]_End
```

# Создание словаря с привязкой согласных символов.

```
keys = {
'B':'Z','C':'X','D':'W','F':'V','G':'T',
'H':'S','J':'R','K':'Q','L':'P','M':'N'}
```

# Сообщение для шифрования/расшифрования.

```
message = list(input("Write the text: ").upper())
```

# Посимвольный перебор сообщения.

**for** symbol **in** range(len(message)):

# Перебор всех ключей в словаре.

**for** key **in** keys:

# Если символ сообщения будет равен ключу, тогда символ сообщения будет равен значению ключа.

**if** message[symbol] == key:

    message[symbol] = keys[key]

# Если символ сообщения будет равен значению ключа, тогда символ сообщения будет равен ключу.

**elif** message[symbol] == keys[key]:

    message[symbol] = key

**else:** pass

# Вывод получившегося сообщения.

**print**("Final message: ", "".join(message))

## 010. Шифр Полибия.

(Шифр Полибия является шифром замены. Шифр Полибия часто представляется в виде квадрата с прочерченными по горизонтали и вертикали числами от 1 до 6.)

```
// main.py
Start_
from re import findall
keysPolibiy = {
    'A':'11', 'B':'12', 'C':'13', 'D':'14',
    'E':'15', 'F':'16', 'G':'21', 'H':'22',
    'I':'23', 'J':'24', 'K':'25', 'L':'26',
    'M':'31', 'N':'32', 'O':'33', 'P':'34',
    'Q':'35', 'R':'36', 'S':'41', 'T':'42',
    'U':'43', 'V':'44', 'W':'45', 'X':'46',
    'Y':'51', 'Z':'52', '0':'53', '1':'54',
    '2':'55', '3':'56', '4':'61', '5':'62',
    '6':'63', '7':'64', '8':'65', '9':'66'
}
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
def regular(text):
    return findall(r"[0-9]{2}", text)
def encryptDecrypt(mode, message, final = []):
    if mode == 'E':
        for symbol in message:
            if symbol in keysPolibiy:
                final.append(keysPolibiy[symbol])
    else:
        for twoNumbers in regular(message):
            for key in keysPolibiy:
                if twoNumbers == keysPolibiy[key]:
                    final.append(key)
    return "".join(final)
print("Final message:", encryptDecrypt(cryptMode, startMessage))
_End
```

**// terminal**

**Start\_**

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Final message: 22.15.26.26.33.45.33.36.26.14
```

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: d
```

```
Write the message: 22.15.26.26.33.45.33.36.26.14
```

```
Final message: H.E.L.L.O.W.O.R.L.D
```

**End**

```
# Импортирование функции findall из регулярных выражений.
```

```
from re import findall
```

```
# Словарь с видом → символ : два_числа.
```

```
keysPolibiy = {
```

```
    'A':'11',    'B':'12',    'C':'13',    'D':'14',  
    'E':'15',    'F':'16',    'G':'21',    'H':'22',  
    'I':'23',    'J':'24',    'K':'25',    'L':'26',  
    'M':'31',    'N':'32',    'O':'33',    'P':'34',  
    'Q':'35',    'R':'36',    'S':'41',    'T':'42',  
    'U':'43',    'V':'44',    'W':'45',    'X':'46',  
    'Y':'51',    'Z':'52',    '0':'53',    '1':'54',  
    '2':'55',    '3':'56',    '4':'61',    '5':'62',  
    '6':'63',    '7':'64',    '8':'65',    '9':'66'
```

```
}
```

```
# Переключатель шифрования/расшифрования
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
# Сообщение для шифрования/расшифрования
```

```
startMessage = input("Write the message: ").upper()
```

```
# Функция с регулярным выражением, которая выделяет по два числа.
```

```
def regular(text):
```

```
    return findall(r"[0-9]{2}", text)
```

# Функция с 3 аргументами.

```
def encryptDecrypt(mode, message, final = []):
```

# Если переключатель равен 'E', тогда сделать посимвольный перебор сообщения.

```
if mode == 'E':
```

```
    for symbol in message:
```

# Если символ будет найден в ключах, тогда к списку final добавить значение ключа.

```
if symbol in keysPolibiy:
```

```
    final.append(keysPolibiy[symbol])
```

# Если переключатель равен 'D', тогда сделать перебор сообщения через регулярное выражение.

```
else:
```

```
    for twoNumbers in regular(message):
```

# Перебор всех ключей в словаре.

```
for key in keysPolibiy:
```

# Если два числа будут найдены в значениях ключа, тогда к списку final добавить сам ключ.

```
if twoNumbers == keysPolibiy[key]:
```

```
    final.append(key)
```

# Вернуть сообщение.

```
return ".".join(final)
```

# Вывод получившегося сообщения.

```
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```

## 011. Шифр Виженера.

(Шифр Виженера является самым знаменитым многоалфавитным шифром. Построен на конструкции шифра Цезаря. Представлен в виде квадрата и содержит в себе фактически 26 шифров Цезаря с разными смещениями по алфавиту.)

// main.py

Start\_

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
oneKey = input("Write the key: ").upper()
```

```
def encryptDecrypt(mode, message, key, final = ""):
```

```
    key *= len(message) // len(key) + 1
```

```
    for index, symbol in enumerate(message):
```

```
        if mode == 'E':
```

```
            temp = ord(symbol) + ord(key[index])
```

```
        else:
```

```
            temp = ord(symbol) - ord(key[index])
```

```
        final += chr(temp % 26 + ord('A'))
```

```
    return final
```

```
print("Final message:", encryptDecrypt(cryptMode, startMessage, oneKey))
```

End

// terminal

Start\_

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Write the key: war
```

```
Final message: DECHONKRCZ
```

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: d
```

```
Write the message: DECHONKRCZ
```

```
Write the key: war
```

```
Final message: HELLOWORLD
```

End



```
# Переключатель шифрования/расшифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit

# Сообщение для шифрования/расшифрования.
startMessage = input("Write the message: ").upper()

# Ключ-слово.
oneKey = input("Write the key: ").upper()

# Функция с 4 аргументами.
def encryptDecrypt(mode, message, key, final = ""):

    # Подгоняем длину ключа под длину сообщения.
    key *= len(message) // len(key) + 1

    # Посимвольный перебор сообщения.
    for index, symbol in enumerate(message):

        # Если переключатель будет равен 'E', тогда к переменной temp присвоить число
        # символа по таблице ASCII + число символа-ключа по таблице ASCII.
        if mode == 'E':
            temp = ord(symbol) + ord(key[index])

        # Если переключатель будет равен 'D', тогда к переменной temp присвоить число
        # символа по таблице ASCII - число символа-ключа по таблице ASCII.
        else:
            temp = ord(symbol) - ord(key[index])

        # К переменной final прибавить получившийся символ.
        final += chr(temp % 26 + ord('A'))

    # Вернуть сообщение.
    return final

# Вывод получившегося сообщения.
print("Final message:", encryptDecrypt(cryptMode, startMessage, oneKey))
```

## 012. Шифр Вернама.

(Шифр Вернама является шифром с абсолютной криптостойкостью, т.к. использует случайные ключи для каждого символа. В данном примере шифр Вернама основан на шифре Цезаря, но чаще всего имеет реализацию с операцией XOR (Исключающее ИЛИ).)

// **main.py**

**Start\_**[

```
from random import randint
```

```
from re import findall
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
def regular(text):
```

```
    return findall(r"[0-9]+", text)
```

```
def encryptDecrypt(mode, message, final = "", keys = []):
```

```
    if mode == 'E':
```

```
        for symbol in message:
```

```
            key = randint(0,25); keys.append(str(key))
```

```
            final += chr((ord(symbol) + key - 13)%26 + ord('A'))
```

```
        return final, ''.join(keys)
```

```
    else:
```

```
        keys = input("Write the keys: ")
```

```
        for index, symbol in enumerate(message):
```

```
            final += chr((ord(symbol) - int(regular(keys)[index]) - 13)%26 +
```

```
ord('A'))
```

```
        return final
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```

**End**

// **terminal**

**Start\_**[

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Final message: ('SXWZOTBFYA', '11/19/11/14/0/23/13/14/13/23/')
```

```
$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the message: SXWZOTBFYA
Write the keys: 11/19/11/14/0/23/13/14/13/23/
Final message: HELLOWORLD
]_End
```

```
# Импортирование функции randint из модуля random.
```

```
from random import randint
```

```
# Импортирование функции findall из регулярных выражений.
```

```
from re import findall
```

```
# Переключатель режимов шифрования
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
# Сообщение для шифрования/расшифрования
```

```
startMessage = input("Write the message: ").upper()
```

```
# Функция с регулярным выражением, возвращающая все числа
```

```
def regular(text):
```

```
    return findall(r"[0-9]+", text)
```

```
# Функция с 4 аргументами.
```

```
def encryptDecrypt(mode, message, final = "", keys = []):
```

```
# Если переключатель равен 'E', тогда сделать посимвольный перебор в сообщении.
```

```
if mode == 'E':
```

```
    for symbol in message:
```

```
# Создание случайного ключа.
```

```
key = randint(0,25); keys.append(str(key))
```

```
# Вычисление символа при помощи таблицы ASCII.
```

```
final += chr((ord(symbol) + key - 13)%26 + ord('A'))
```

# Вернуть зашифрованное сообщение и ключи

**return** final, ' '.join(keys)

# Если переключатель равен 'D', тогда сделать ввод ключа.

**else:**

    keys = input("Write the keys: ")

# Посимвольный перебор зашифрованного сообщения

**for** index, symbol **in** enumerate(message):

# Вычисление символа при помощи таблицы ASCII.

# regular(keys) возвращает список, [index] даёт нам определённое число из списка.

**final** += chr((ord(symbol) - int(regular(keys)[index]) - 13)%26 + ord('A'))

# Вернуть сообщение.

**return** final

# Вывод получившегося сообщения.

**print**("Final message:", encryptDecrypt(cryptMode, startMessage))

## 013. Книжный шифр.

(Книжный шифр является одним из самых криптостойких методов шифрования. Особенность шифра заключена в ключе-книге, которая может являться любой текстовой информацией.)

// main.py

Start\_

```
from random import choice
```

```
from re import findall
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ")
```

```
bookKey = input("Write the book-key: ")
```

```
def regular(text):
```

```
    return findall(r"[0-9]+", text)
```

```
def encryptDecrypt(mode, message, key, final = ""):
```

```
    with open(key) as bookKey:
```

```
        book = bookKey.read()
```

```
    if mode == 'E':
```

```
        for symbolMessage in message:
```

```
            listIndexKey = []
```

```
            for indexKey, symbolKey in enumerate(book):
```

```
                if symbolMessage == symbolKey:
```

```
                    listIndexKey.append(indexKey)
```

```
            try: final += str(choice(listIndexKey)) + '/'
```

```
            except IndexError: pass
```

```
    else:
```

```
        for numbers in regular(message):
```

```
            for indexKey, symbolKey in enumerate(book):
```

```
                if numbers == str(indexKey):
```

```
                    final += symbolKey
```

```
    return final
```

```
print("Final message:", encryptDecrypt(cryptMode, startMessage, bookKey))
```

End

**// book.txt**

**Start\_**

O how I faint when I of you do write Knowing a better spirit doth use your name And in the praise thereof spends all his might To make me tongue tied speaking of your fame But since your worth wide as the ocean is The humble as the proudest sail doth bear My saucy bark inferior far to his On your broad main doth wilfully appear Your shallowest help will hold me up afloat Whilst he upon your soundless deep doth ride Or being wracked I am a worthless boat He of tall building and of goodly pride Then if he thrive and I be cast away The worst was this my love was my decay

**\_End**

**// terminal**

**Start\_**

\$ python main.py

[E]ncrypt[D]ecrypt: e

Write the message: helloworld

Write the book-key: book.txt

Final message: 381/223/489/320/482/4/294/233/377/359/

\$ python main.py

[E]ncrypt[D]ecrypt: d

Write the message: 381/223/489/320/482/4/294/233/377/359/

Write the book-key: book.txt

Final message: helloworld

**\_End**

# Импорт функции choice из модуля random.

**from random import choice**

# Импорт функции findall из регулярных выражений.

**from re import findall**

# Переключатель режимов шифрования.

**cryptMode = input("[E]ncrypt[D]ecrypt: ").upper()**

**if cryptMode not in ['E', 'D']:**

**print("Error: mode is not found")**

**raise SystemExit**

# Сообщения для шифрования/расшифрования.

**startMessage** = **input**("Write the message: ")

# Ключ-книга.

**bookKey** = **input**("Write the book-key: ")

# Функция с регулярным выражением, возвращающая список чисел.

```
def regular(text):  
    return findall(r"[0-9]+", text)
```

# Функция с 4 аргументами.

```
def encryptDecrypt(mode, message, key, final = ""):
```

# Открытие ключа-книги на чтение.

```
with open(key) as bookKey:  
    book = bookKey.read()
```

# Если переключатель будет равен 'E', тогда перебрать посимвольно сообщение.

```
if mode == 'E':  
    for symbolMessage in message:
```

# Создание или обнуление списка

**listIndexKey** = []

# Посимвольный перебор книги

```
for indexKey, symbolKey in enumerate(book):
```

# Если символ сообщения будет равен символу в книге, тогда к списку **listIndexKey** добавить индекс этого символа в книге.

```
if symbolMessage == symbolKey:  
    listIndexKey.append(indexKey)
```

# Попытка добавить к переменной **final** ключ символа, если же список пустой – пропустить.

```
try: final += str(choice(listIndexKey)) + '/'  
except IndexError: pass
```

# Если переключатель равен 'D', то перебрать зашифрованное сообщение по ключам.

```
else:  
    for numbers in regular(message):
```

# Посимвольный перебор книги и если номер зашифрованного сообщения будет равен индексу этого номера в книге, тогда к переменной final добавить этот символ.

```
for indexKey, symbolKey in enumerate(book):  
    if numbers == str(indexKey):  
        final += symbolKey
```

# Вернуть сообщение.

```
return final
```

# Вывод получившегося сообщения.

```
print("Final message:",encryptDecrypt(cryptMode, startMessage, bookKey))
```



## 014. XOR шифрование.

(XOR шифрование основано на логической операции Иключающее ИЛИ. Данное шифрование не требует переключателя.)

**// main.py**

**Start\_**

```
message = list(input("Write the message: "))
```

```
key = input("Write the key: ")
```

```
for symbol in range(len(message)):
```

```
    try: message[symbol] = chr(ord(message[symbol]) ^ int(key))
```

```
    except ValueError: message[symbol] = chr(ord(message[symbol]) ^ ord(key))
```

```
print("Final message:", "".join(message))
```

**\_End**

**// terminal**

**Start\_**

```
$ python main.py
```

```
Write the message: helloworld
```

```
Write the key: 50
```

```
Final message: ZW^^]E]@^V
```

```
$ python main.py
```

```
Write the message: ZW^^]E]@^V
```

```
Write the key: 50
```

```
Final message: helloworld
```

**\_End**

# Сообщение для шифрования/расшифрования

```
message = list(input("Write the message: "))
```

# Ключ

```
key = input("Write the key: ")
```

# Посимвольный перебор сообщения

```
for symbol in range(len(message)):
```

# Если key = целое число

```
try: message[symbol] = chr(ord(message[symbol]) ^ int(key))
```

# Если key = символ

```
except ValueError: message[symbol] = chr(ord(message[symbol]) ^ ord(key))
```

# Вывод получившегося сообщения.

```
print("Final message:", "".join(message))
```

## 015. Шифр с ложными символами.

(Ложные символы используются для увеличения криптостойкости шифра. Могут быть использованы во множестве методов шифрования.)

// main.py

Start\_

```
from random import randint, choice
traps = ('"', '\\', '{', '}', '\'', '№', '\n')
symbolAlpha = [chr(x) for x in range(65,91)]
symbolCrypt = ('!', '@', '#', '$', '%', '^', '&', '*', ')', '(', '~', '_', '+', '-', '=', '<', '>', '?', '/', '[', ']', ',', '|', ':', ';', '!')
keys = dict(zip(symbolAlpha, symbolCrypt))
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
def encryptDecrypt(mode, message, final = ""):
    message = list(message)
    if mode == 'E':
        length = len(message) // 4
        for _ in range(length):
            message.insert(randint(0, len(message)), choice(traps))
        for symbol in message:
            if symbol in keys:
                final += keys[symbol]
            else: final += symbol
    else:
        for symbol in message:
            for key in keys:
                if symbol == keys[key]:
                    final += key
    return final
print("Final message:", encryptDecrypt(cryptMode, startMessage))
_End
```

**// terminal**

**Start\_**

\$ python main.py

[E]ncrypt|[D]ecrypt: e

Write the message: helloworld

Final message: \*%\_\_=|"?\_№

\$ python main.py

[E]ncrypt|[D]ecrypt: d

Write the message: \*%\_\_=|"?\_№

Final message: HELLOWORLD

**End**

# Импорт двух функций из модуля random.

**from random import randint, choice**

# Кортеж с ложными символами.

**traps = ('"', '\\', '{', '}', '\'', '№', '\')**

# Символы алфавита от 'A' до 'Z'.

**symbolAlpha = [chr(x) for x in range(65,91)]**

# Символы для замены.

**symbolCrypt = ('!', '@', '#', '\$', '%', '^', '&', '\*', ')', '(', '~',  
'\_', '+', '-', '=', '<', '>', '?', '/', '[', ']', ',', '|', ':', ';', '.')**

# Создание словаря вида → символ\_алфавита : символ\_шифра.

**keys = dict(zip(symbolAlpha, symbolCrypt))**

# Переключатель режимов шифрования.

**cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()**

**if cryptMode not in ['E', 'D']:**

**print("Error: mode is not found")**

**raise SystemExit**

# Сообщение для шифрования/расшифрования.

**startMessage = list(input("Write the message: ").upper())**

# Функция с 3 аргументами.

**def encryptDecrypt(mode, message, final = ""):**

# Если переключатель будет равен 'E', тогда создать переменную length.

```
if mode == 'E':  
    length = len(message) // 4
```

# Вносить в сообщение рандомные ложные символы в рандомные позиции.

```
for _ in range(length):  
    message.insert(randint(0,len(message)),choice(traps))
```

# Посимвольный перебор сообщения

```
for symbol in message:
```

# Если символ находится в ключах, тогда к переменной final добавить значение ключа.

```
if symbol in keys:  
    final += keys[symbol]
```

# Иначе к переменной final добавить сам символ.

```
else: final += symbol
```

# Если переключатель равен значению 'D', тогда сделать посимвольный перебор сообщения и перебор всех ключей.

```
else:  
    for symbol in message:  
        for key in keys:
```

# Если символ будет равен значению ключа, тогда к переменной final добавить ключ.

```
if symbol == keys[key]:  
    final += key
```

# Вернуть сообщение.

```
return final
```

# Вывод получившегося сообщения.

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```

## 016. Шифр Гронсфельда.

(Шифр Гронсфельда основан на шифре Виженера и соответственно является многоалфавитным методом шифрования. Вместо ключа-слова используются числа, тем самым вместо 26 возможных символов, получаем 10.)

**// main.py**

**Start\_**

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
keyNumber = input("Write the keyNumber: ")
```

```
def encryptDecrypt(mode, message, key, final = ""):
```

```
    key *= len(message) // len(key) + 1
```

```
    for index, symbol in enumerate(message):
```

```
        if mode == 'E':
```

```
            temp = ord(symbol) + int(key[index]) - 13
```

```
        else:
```

```
            temp = ord(symbol) - int(key[index]) - 13
```

```
        final += chr(temp%26 + ord('A'))
```

```
    return final
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage, keyNumber))
```

**\_End**

**// terminal**

**Start\_**

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Write the keyNumber: 2018
```

```
Final message: JEMTQWPZND
```

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: d
```

```
Write the message: JEMTQWPZND
```

```
Write the keyNumber: 2018
```

```
Final message: HELLOWORLD
```

**\_End**

```
# Переключатель режимов шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit

# Сообщение для шифрования/расшифрования.
startMessage = input("Write the message: ").upper()

# Ключ-номер.
keyNumber = input("Write the keyNumber: ")

# Функция с 4 аргументами.
def encryptDecrypt(mode, message, key, final = ""):

    # Подгоняем длину ключа к длине сообщения.
    key *= len(message) // len(key) + 1

    # Посимвольный перебор сообщения.
    for index, symbol in enumerate(message):

        # Если переключатель равен 'E', тогда прибавлять число ключа.
        if mode == 'E':
            temp = ord(symbol) + int(key[index]) - 13

        # Если переключатель равен 'D', тогда вычитать число ключа.
        else:
            temp = ord(symbol) - int(key[index]) - 13

        # К переменной final прибавить получившийся символ.
        final += chr(temp%26 + ord('A'))

    # Вернуть сообщение.
    return final

# Вывод получившегося сообщения.
print("Final message:", encryptDecrypt(cryptMode, startMessage, keyNumber))
```

## 017. Псевдосимвольный шифр.

(Псевдосимвольный шифр основан на шифре Бэкона. Исключительная особенность данного метода шифрования заключена в трёх одинаковых внешне символах, но различающихся в кодировке Unicode. Один символ 'А' взят из латиницы, другой из кириллицы, третий из греческого алфавита.)

// main.py

Start\_

```
from re import findall
```

```
keysPsevdo = {
```

```
    'A':"AAA", 'B':"AAA", 'C':"AAA",  
    'D':"AAA", 'E':"AAA", 'F':"AAA",  
    'G':"AAA", 'H':"AAA", 'I':"AAA",  
    'J':"AAA", 'K':"AAA", 'L':"AAA",  
    'M':"AAA", 'N':"AAA", 'O':"AAA",  
    'P':"AAA", 'Q':"AAA", 'R':"AAA",  
    'S':"AAA", 'T':"AAA", 'U':"AAA",  
    'V':"AAA", 'W':"AAA", 'X':"AAA",  
    'Y':"AAA", 'Z':"AAA", ' ':'AAA"
```

```
}
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
def regular(text):
```

```
    return findall(r"\w{3}", text)
```

```
def encryptDecrypt(mode, message, final = ""):
```

```
    if mode == 'E':
```

```
        for symbol in message:
```

```
            if symbol in keysPsevdo:
```

```
                final += keysPsevdo[symbol]
```

```
    else:
```

```
        for threeSymbols in regular(message):
```

```
            for key in keysPsevdo:
```

```
                if threeSymbols == keysPsevdo[key]:
```

```
                    final += key
```

```
    return final
```

```
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```



**]\_End**

**// terminal**

**Start\_**

\$ python main.py

[E]ncrypt|[D]ecrypt: e

Write the message: helloworld

Final message: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

\$ python main.py

[E]ncrypt|[D]ecrypt: d

Write the message: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Final message: HELLOWORLD

**]\_End**

# Импортирование функции из регулярных выражений.

**from re import findall**

# Словарь вида → символ\_алфавита : Зсимвола.

**keysPsevdo = {**

**'A':"AAA", 'B':"AAA", 'C':"AAA",**  
**'D':"AAA", 'E':"AAA", 'F':"AAA",**  
**'G':"AAA", 'H':"AAA", 'I':"AAA",**  
**'J':"AAA", 'K':"AAA", 'L':"AAA",**  
**'M':"AAA", 'N':"AAA", 'O':"AAA",**  
**'P':"AAA", 'Q':"AAA", 'R':"AAA",**  
**'S':"AAA", 'T':"AAA", 'U':"AAA",**  
**'V':"AAA", 'W':"AAA", 'X':"AAA",**  
**'Y':"AAA", 'Z':"AAA", ' ':'AAA"**

**}**

# Переключатель режимов шифрования.

**cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()**

**if cryptMode not in ['E','D']:**

**print("Error: mode is not found")**

**raise SystemExit**

# Сообщение для шифрования/расшифрования.

**startMessage = input("Write the message: ").upper()**

# Функция регулярного выражения возвращающая 3 символа.

```
def regular(text):  
    return findall(r"\w{3}", text)
```

# Функция с 3 аргументами.

```
def encryptDecrypt(mode, message, final = ""):
```

# Если переключатель будет равен 'E', тогда сделать посимвольный перебор сообщения.

```
if mode == 'E':  
    for symbol in message:
```

# Если символ будет находится в ключах, тогда добавить к переменной final значение ключа.

```
if symbol in keysPsevdo:  
    final += keysPsevdo[symbol]
```

# Если переключатель будет равен 'D', тогда сделать перебор через функцию с регулярным выражением и сделать перебор всех ключей в словаре.

```
else:  
    for threeSymbols in regular(message):  
        for key in keysPsevdo:
```

# Если три символа равны значению ключа в словаре, тогда к переменной final добавить сам ключ.

```
if threeSymbols == keysPsevdo[key]:  
    final += key
```

# Вернуть сообщение.

```
return final
```

# Вывод получившегося сообщения.

```
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```

## 018. Кодирование.

(Кодирование в криптографии – это замена целых слов на какой-либо шифртекст/слово. Относится к ветке шифров замены.)

// **main.py**

**Start\_**[

```
tupleWord = ('AND','THE','OR','ALL','ANY','WHAT','WHY','YES','NO',  
'ONE','YOU','HE','SHE','USE','IF','ELSE','THIS','THAN','YOUR',  
'ON','HOW','ARE','ME','IT','IS','THAT','WAS','OF','BE','OK')
```

```
tupleCode = ('!','@','#','$','%','^','&','*','(',')','_','-','  
'+','=','/','?','<','>',';',':', '{','}', '[',']', '~','.',',','"','|','\')
```

```
keys = dict(zip(tupleWord, tupleCode))
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
def encryptDecrypt(mode, message):
```

```
    for key in keys:
```

```
        if mode == 'E':
```

```
            if key in message:
```

```
                message = message.replace(key,keys[key])
```

```
        else:
```

```
            if keys[key] in message:
```

```
                message = message.replace(keys[key],key)
```

```
    return message
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```

**]\_End**

// **terminal**

**Start\_**[

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: yes or no
```

```
Final message: * # (
```

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: d
```

```
Write the message: * # (
```

Final message: YES OR NO

**l\_End**

# Кортеж слов.

```
tupleWord = ('AND','THE','OR','ALL','ANY','WHAT','WHY','YES','NO',  
'ONE','YOU','HE','SHE','USE','IF','ELSE','THIS','THAN','YOUR',  
'ON','HOW','ARE','ME','IT','IS','THAT','WAS','OF','BE','OK')
```

# Кортеж кодов.

```
tupleCode = ('!','@','#','$','%','^','&','*','(',')','-', '_',  
'+', '=', '/', '?', '<', '>', ';', ':', '{', '}', '[', ']', '~', ',', '.', '"', '|', '\\')
```

# Словарь вида → слово : код.

```
keys = dict(zip(tupleWord, tupleCode))
```

# Переключатель режимов шифрования.

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

# Сообщение для шифрования/расшифрования.

```
startMessage = input("Write the message: ").upper()
```

# Функция с 2 аргументами.

```
def encryptDecrypt(mode, message):
```

# Перебор всех ключей.

```
for key in keys:
```

# Если переключатель будет равен 'E' и ключ будет находится в сообщении, то заменить этот ключ на значение ключа.

```
if mode == 'E':
```

```
    if key in message:
```

```
        message = message.replace(key,keys[key])
```

# Если переключатель равен 'D' и значение ключа находится в сообщении, то заменить это значение ключа на сам ключ.

```
else:
```

```
    if keys[key] in message:
```

```
        message = message.replace(keys[key],key)
```

# Вернуть сообщение.

**return message**

# Вывод получившегося сообщения.

**print("Final message:", encryptDecrypt(cryptMode, startMessage))**

## 019. Шифр Цезаря с ключевым словом.

(Является обычным шифром Цезаря за исключением того, что данный метод шифрования меняет порядок символов в алфавите за счёт ключевого слова.)

// main.py

Start\_

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
alphaList = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
startMessage = input("Write the message: ").upper()
try: numberKey = int(input("Write the number key: "))
except ValueError:
    print("Error: key is not int number")
    raise SystemExit
stringKey = list(input("Write the string key: ").upper())
def remove(alpha, string):
    for symbol in string:
        if symbol in alpha: alpha.remove(symbol)
    for symbol in string:
        if symbol not in [chr(x) for x in range(65,91)] \
            or string.count(symbol) > 1: string.remove(symbol)
    return alpha, string
def insert(alpha_string):
    for index, symbol in enumerate(alpha_string[1]):
        alpha_string[0].insert(index, symbol)
    return alpha_string[0]
def replace(alpha, key):
    while key > 0:
        alpha.insert(0,alpha[-1])
        del alpha[-1]
        key -= 1
    return alpha
def encryptDecrypt(mode, message, key, final = ""):
    alphaS = [x for x in alphaList]
    alphaC = replace(insert(remove(alphaList, stringKey)), key)
    for symbol in message:
        if mode == 'E':
```

```

        final += alphaC[alphaS.index(symbol)]
    else:
        final += alphaS[alphaC.index(symbol)]
    return final
print("Final message:", encryptDecrypt(cryptMode, startMessage, numberKey))
]_End

```

**// terminal**

**Start\_**

```

$ python main.py
[E]ncrypt|[D]ecrypt: e
Write the message: helloworld
Write the number key: 3
Write the string key: something
Final message: GIRRTZTWRK

```

```

$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the message: GIRRTZTWRK
Write the number key: 3
Write the string key: something
Final message: HELLOWORLD
]_End

```

```

# Переключатель режимов шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit

# Используемый алфавит.
alphaList = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

# Сообщение для шифрования/расшифрования.
startMessage = input("Write the message: ").upper()

```

# Если ключ не является целым числом, тогда вывести ошибку.

```
try: numberKey = int(input("Write the number key: "))
except ValueError:
    print("Error: key is not int number")
    raise SystemExit
```

# Ключевое слово.

```
stringKey = list(input("Write the string key: ").upper())
```

# Функция удаления повторяющихся символов в алфавите и в ключевом слове.

```
def remove(alpha, string):
```

# Перебор всех символов с ключевом слове и если символ ключевого слова будет находиться в алфавите, тогда из алфавита удалить этот символ.

```
for symbol in string:
    if symbol in alpha: alpha.remove(symbol)
```

# Переор всех символов в ключевом слове и если символ ключевого слова не будет находится в диапазоне A-Z или будет встречаться более одного раза, тогда удалить этот символ из ключевого слова.

```
for symbol in string:
    if symbol not in [chr(x) for x in range(65,91)] \
    or string.count(symbol) > 1: string.remove(symbol)
```

# Вернуть новый алфавит и ключевое слово.

```
return alpha, string
```

# Функция добавления ключевого слова в алфавит.

```
def insert(alpha_string):
```

# Перебор всех символов ключевого слова и посимвольное внедрение в алфавит.

```
for index, symbol in enumerate(alpha_string[1]):
    alpha_string[0].insert(index, symbol)
```

# Вернуть новый алфавит.

```
return alpha_string[0]
```

# Функция перестановки символов в получившемся алфавите.

```
def replace(alpha, key):
```

# Пока значение ключа больше нуля.

```
while key > 0:
```



# В начало списка вставлять конечный элемент этого же списка.

**alpha.insert(0,alpha[-1])**

# Удалить последний элемент списка.

**del alpha[-1]**

# Вычесть из значения ключа единицу.

**key -= 1**

# Вернуть получившийся алфавит.

**return alpha**

# Основная функция.

**def encryptDecrypt(mode, message, key, final = ""):**

# Создание копии алфавита.

alphaS = [x for x in alphaList]

# Создание изменённого алфавита под шифрование.

alphaC = replace(insert(remove(alphaList, stringKey)), key)

# Посимвольный перебор сообщения.

**for symbol in message:**

# Если переключатель равен 'E', тогда к переменной final прибавлять символ изменённого алфавита по индексу из копии стандартного алфавита.

**if mode == 'E':**

final += alphaC[alphaS.index(symbol)]

# Иначе, если переключатель равен 'D', тогда к переменной final прибавлять символ стандартного алфавита по индексу из изменённого алфавита.

**else:**

final += alphaS[alphaC.index(symbol)]

# Вернуть сообщение.

**return final**

# Вывод получившегося сообщения.

**print("Final message:", encryptDecrypt(cryptMode, startMessage, numberKey))**

## 020. Шифр замены слогов.

(Шифр замены слогов используется для увеличения криптостойкости. Может быть использован во множестве методов шифрования.)

```
// main.py
```

```
Start_
```

```
syllables = ('TH','EE','OO','ING','ED','SS','DE','RE','AR',
'WH','AI','IS','BE','CH','SH','GH','EN','OU','LL','HE','US',
'ST','EV','WO','UI','IN','ER','OR','AT','RD','AL','LE','LD',
'UR','UP','SO','ME','SE','MY','NA','TE','NE','VE','LA','GE',
'ON','GU','RA','AN','AG','SH','CR','FO','OW','PY','WR','CA',
'EA','SP','PR','AS','AU','MA','KE','UT','DO','NT','WA','HU',
'AD','WI','RI','LO','FU','BR','OF','AP','TO','IF','AM','ND',
'LY','TA','KN','FA','TT','LP')
symbols = ('!$', '@@2', '#99', '$$', '^$', '&<<', '**?', ';;{',
'|}', ':#', '-./', '++', '//~', '=='], '[++', '?::', '>>&', '//?',
'&**', '!<<', '%(', ':>', '<:', '*++', '?//', '^$$', '~'", '!:',
':!', '&??', '//', '#$$', '(:,:)', '{[[', '[[]', ';;<', '[[[',
'$??', '0//', '1[[', '@]]', '[<:', ']]', ':[[', '::', '!', '@',
'#', '$', '%', '^', '&', '*', '(', ')', '-', '_', '=', '+', '{', '}', ':',
':', '"', ';<', '>', '?', '/', '~', '\', '|', '\\', '[', ']', '1', '2',
'3', '4', '5', '6', '7', '8', '9', '0')
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
def encryptDecrypt(mode, message):
    if mode == 'E':
        for syllable in syllables:
            if syllable in message:
                message =
message.replace(syllable, symbols[syllables.index(syllable)])
    else:
        for symbol in symbols:
            if symbol in message:
                message = message.replace(symbol,
syllables[symbols.index(symbol)])
    return message
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End
```

**// terminal**

**Start\_**

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Final message: !<<&**O*++R((:
```

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: d
```

```
Write the message: !<<&**O*++R((:
```

```
Final message: HELLOWORLD
```

**]\_End**

# Кортеж слогов.

```
syllables = ('TH','EE','OO','ING','ED','SS','DE','RE','AR',
'WH','AI','IS','BE','CH','SH','GH','EN','OU','LL','HE','US',
'ST','EV','WO','UI','IN','ER','OR','AT','RD','AL','LE','LD',
'UR','UP','SO','ME','SE','MY','NA','TE','NE','VE','LA','GE',
'ON','GU','RA','AN','AG','SH','CR','FO','OW','PY','WR','CA',
'EA','SP','PR','AS','AU','MA','KE','UT','DO','NT','WA','HU',
'AD','WI','RI','LO','FU','BR','OF','AP','TO','IF','AM','ND',
'LY','TA','KN','FA','TT','LP')
```

# Кортеж шифров.

```
symbols = ('!','$','@@2','#99','$$', '^$', '&<<', '**?', ';;{',
'||}', '::#', '--/', '+++;', '//~', '=]', '[++', '?::', '>>&', '//?',
'&**', '!<<', '%((', ':>', '<;', '*++', '?//', '^$$', '~"'''', '!::',
':::!', '&??', '//!', '#$$', '(:, :))', '{[[', '[[]', ';;<', '|[[',
'$??', '0//', '1[[', '@]]', '[[<', ':]]', ':[[', ']:', ':!', '@',
'#', '$', '%', '^', '&', '*', '(', ')', '-', '_', '=', '+', '{', '}', ':',
';', '"', ',', '<', '>', '?', '/', '~', '`', '|', '\\', '[', ']', '1', '2',
'3', '4', '5', '6', '7', '8', '9', '0')
```

```
# Переключатель режимов шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit

# Сообщения для шифрования/расшифрования.
startMessage = input("Write the message: ").upper()

# Функция с 2 аргументами.
def encryptDecrypt(mode, message):

    # Если переключатель равен 'E', тогда сделать перебор всех слогов в кортеже.
    if mode == 'E':
        for syllable in syllables:

            # Если слог находится в сообщении, то заменить этот слог на символ из шифров.
            if syllable in message:
                message = message.replace(syllable, symbols[syllables.index(syllable)])

            # Если переключатель равен 'D', тогда сделать перебор всех шифров в кортеже.
            else:
                for symbol in symbols:

                    # Если шифр находится в сообщении, то заменить шифр на слог.
                    if symbol in message:
                        message = message.replace(symbol, syllables[symbols.index(symbol)])

    # Вернуть сообщение.
    return message

# Вывод получившегося сообщения.
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```

## 021. Шифр Порты.

(Шифр порты является биграммным методом шифрования. Смысл заключён в шифровании двух символов одной триадой чисел. Чаще всего шифр Порты представляют в виде таблицы, где по горизонтали и вертикали прочертаны символы алфавита, а в самой таблице находятся триады чисел.)

// main.py

Start\_

```
from re import findall
stageOne = ['00'+str(x) for x in range(1,10)]
stageTwo = ['0'+str(x) for x in range(10,100)]
stageThree = [str(x) for x in range(100,676+1)]
N = tuple(stageOne + stageTwo + stageThree)
del stageOne, stageTwo, stageThree
coordinateX = tuple([chr(alpha) for alpha in range(65,91)])
coordinateY = tuple([chr(alpha) for alpha in range(65,91)])
cryptKeys = {x:None for x in N}
keys = tuple([key for key in cryptKeys])
counter = 0
for x in coordinateX:
    for y in coordinateY:
        cryptKeys[keys[counter]] = x + y
        counter += 1
del N, coordinateX, coordinateY, counter, keys
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
def regular(mode, text):
    if mode == 'E': template = r"[A-Z]{2}"
    else: template = r"[0-9]{3}"
    return findall(template, text)
def encryptDecrypt(mode, message, final = []):
    if mode == 'E':
        for symbol in message:
            if symbol not in [chr(x) for x in range(65,91)]:
                message = message.replace(symbol,"")
        if len(message)%2 != 0: message += 'Z'
```

```

        for symbols in regular(mode, message):
            for key in cryptKeys:
                if symbols == cryptKeys[key]:
                    final.append(key)
    else:
        for number in regular(mode, message):
            if number in cryptKeys:
                final.append(cryptKeys[number])
    return ".".join(final)
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End

```

**// terminal**

**Start\_**

\$ python main.py

[E]ncrypt|[D]ecrypt: e

Write the message: helloworld

Final message: 187.298.387.382.290

\$ python main.py

[E]ncrypt|[D]ecrypt: d

Write the message: 187.298.387.382.290

Final message: HE.LL.OW.OR.LD

**]\_End**

# Импортир функции из регулярных выражений

**from re import findall**

# Создание списков содержащих числа от 001 до 676.

**stageOne = ['00'+str(x) for x in range(1,10)]**

**stageTwo = ['0'+str(x) for x in range(10,100)]**

**stageThree = [str(x) for x in range(100,676+1)]**

# Ввод всех списков в один кортеж N и удаление предыдущих списков.

**N = tuple(stageOne + stageTwo + stageThree)**

**del stageOne, stageTwo, stageThree**

```

# Создание координат с алфавитом от A-Z.
coordinateX = tuple([chr(alpha) for alpha in range(65,91)])
coordinateY = tuple([chr(alpha) for alpha in range(65,91)])

# Создание ключей.
cryptKeys = {x:None for x in N}
keys = tuple([key for key in cryptKeys])

# Перебор всех возможных пар алфавита.
counter = 0
for x in coordinateX:
    for y in coordinateY:

# Занесение в словарь значений ключа.
cryptKeys[keys[counter]] = x + y
counter += 1

# Удаление ненужных элементов.
del N, coordinateX, coordinateY, counter, keys

# Создание переключателя шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit

# Сообщение для шифрования/расшифрования.
startMessage = input("Write the message: ").upper()

# Регулярное выражение с условием, если зашифровать сообщение, значит
разграничить сообщение по 2 символа, если же расшифровать, значить
разграничить сообщение по 3 числа и вернуть список.
def regular(mode, text):
    if mode == 'E': template = r"[A-Z]{2}"
    else: template = r"[0-9]{3}"
    return findall(template, text)

# Создание главной функции
def encryptDecrypt(mode, message, final = []):

```

# Если переключатель равен шифрованию, значит перебрать все символы в сообщении.

```
if mode == 'E':  
    for symbol in message:
```

# Если символ не будет находится в диапазоне A-Z, тогда удалить этот символ.

```
if symbol not in [chr(x) for x in range(65,91)]:  
    message = message.replace(symbol, '')
```

# Если длина сообщения не делится на 2 без остатка – добавить в конец сообщения символ 'Z'.

```
if len(message)%2 != 0: message += 'Z'
```

# Перебор пар символов через регулярное выражение.

```
for symbols in regular(mode, message):  
    for key in cryptKeys:
```

# Если символ будет равен значению ключа, значит заменить на сам ключ.

```
if symbols == cryptKeys[key]:  
    final.append(key)
```

# Если же переключатель равен расшифровки, тогда сделать перебор трёх чисел через функцию с регулярным выражением.

```
else:  
    for number in regular(mode, message):
```

# Если число находится в ключах – заменить на значение ключа.

```
if number in cryptKeys:  
    final.append(cryptKeys[number])
```

# Вернуть сообщение.

```
return ".".join(final)
```

# Вывод получившегося сообщения.

```
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```



## 022. Роторное шифрование.

(Роторное шифрование построено на принципе шифра Вернама, то-есть с каждым новым поступающим на вход символом ключ будет изменяться по роторам, имитируя случайность.)

```
// main.py
```

```
Start_
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
rotors = (
```

```
    (10,24,14,12,23,2,7,15,24,2,7,5,22,6,2,1,22,12,6,9,7,2,11,23,14,2),
```

```
    (1,7,11,26,12,5,11,20,11,7,18,6,17,18,19,1,13,5,2,9,11,13,6,17,26,24),
```

```
    (9,1,21,6,4,19,25,6,17,10,26,1,23,6,1,17,19,17,25,21,3,21,17,1,18,20)
```

```
)
```

```
def encryptDecrypt(mode, message, final = ""):
```

```
    x,y,z = 1,2,3
```

```
    for symbol in message:
```

```
        rotor = rotors[0][x] + rotors[1][y] + rotors[2][z]
```

```
        if mode == 'E':
```

```
            if symbol in [chr(x) for x in range(65,91)]:
```

```
                final += chr((ord(symbol) - 13 + rotor)%26 + ord('A'))
```

```
            else: continue
```

```
        else:
```

```
            final += chr((ord(symbol) - 13 - rotor)%26 + ord('A'))
```

```
        if x != 25: x += 1
```

```
        else:
```

```
            x = 0
```

```
            if y != 25: y += 1
```

```
            else:
```

```
                y = 0
```

```
                if z != 25: z += 1
```

```
                else: z = 0
```

```
    return final
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```

```
_End
```

**// terminal**

**Start\_**

\$ python main.py

[E]ncrypt|[D]ecrypt: e

Write the message: helloworld

Final message: WJOZHUUGEB

\$ python main.py

[E]ncrypt|[D]ecrypt: d

Write the message: WJOZHUUGEB

Final message: HELLOWORLD

**End**

# Переключатель режимов шифрования.

**cryptMode** = **input**("[E]ncrypt|[D]ecrypt: ").**upper**()

**if** cryptMode **not in** ['E','D']:

**print**("Error: mode is not found")

**raise** **SystemExit**

# Сообщение для шифрования/расшифрования.

**startMessage** = **input**("Write the message: ").**upper**()

# Роторные позиции.

**rotors** = (

    (10,24,14,12,23,2,7,15,24,2,7,5,22,6,2,1,22,12,6,9,7,2,11,23,14,2),

    (1,7,11,26,12,5,11,20,11,7,18,6,17,18,19,1,13,5,2,9,11,13,6,17,26,24),

    (9,1,21,6,4,19,25,6,17,10,26,1,23,6,1,17,19,17,25,21,3,21,17,1,18,20)

)

# Главная функция.

**def** **encryptDecrypt**(mode, message, final = ""):

# Начальная позиция роторов.

**x,y,z** = 1,2,3

# Посимвольный перебор сообщения и прибавление всех позиций роторов.

**for** symbol **in** message:

    rotor = rotors[0][x] + rotors[1][y] + rotors[2][z]

# Если переключатель равен режиму шифрования, значит проверить встречается ли символ в диапазоне A-Z. И если это так, то зашифровать.

```
if mode == 'E':  
    if symbol in [chr(x) for x in range(65,91)]:  
        final += chr((ord(symbol) - 13 + rotor)%26 + ord('A'))  
    else: continue
```

# Если переключатель равен режиму расшифрования, значит расшифровывать посимвольно сообщение.

```
else:  
    final += chr((ord(symbol) - 13 - rotor)%26 + ord('A'))
```

# Если x не равняется 25 (конечному значению в кортеже), значит к x прибавлять единицу. Если же x равен 25 – округлить значение и к y прибавить единицу. Также работает и с другими кортежами y,z.

```
if x != 25: x += 1  
else:  
    x = 0  
    if y != 25: y += 1  
    else:  
        y = 0  
        if z != 25: z += 1  
        else: z = 0
```

# Вернуть сообщение.

```
return final
```

# Вывод получившегося сообщения.

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```

## 023. Криптографические хеш-функции.

(Криптографические хеш-функции не предназначены для расшифровки. Они созданы лишь для проверки и подтверждения информации.  
(На примере показана криптографическая хеш-функция sha256, но в библиотеке hashlib также есть и другие, наподобие md5, sha512 и т.д.))

**// main.py**

**Start\_**[

```
from hashlib import sha256
def encrypt(string):
    signature = sha256(string.encode()).hexdigest()
    return signature
staticPassword = encrypt("secret")
dynamicPassword = encrypt(input("Write the password: "))
if staticPassword == dynamicPassword:
    print("Password is True!")
else:
    print("Password is False!")
```

**End\_**

**// terminal**

**Start\_**[

```
$ python main.py
Write the password: secret
Password is True!
```

```
$ python main.py
Write the password: hello
Password is False!
```

**End\_**

# Импортирование функции sha256 из библиотеки hashlib

```
from hashlib import sha256
```

# Создание функции, которая будет возвращать хешированное сообщение.

```
def encrypt(string):
    signature = sha256(string.encode()).hexdigest()
    return signature
```

# Пароль, который является статичным.

**staticPassword** = **encrypt**("secret")

# Пароль, который мы пытаемся сравнить со статичным.

**dynamicPassword** = **encrypt**(**input**("Write the password: "))

# Если статичный пароль равен динамическому, значит пароль верный.

**if** **staticPassword** == **dynamicPassword**:

**print**("Password is True!")

# Иначе пароль не верный.

**else**:

**print**("Password is False!")

## 024. Шифр Вернама (XOR).

(Шифр Вернама является шифром с абсолютной криптостойкостью, т.к. использует случайные ключи для каждого символа. В данном примере шифр Вернама основан на XOR шифровании.)

// **main.py**

**Start\_**[

```
from random import randint
```

```
from re import findall
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ")
```

```
def regular(text):
```

```
    return findall(r"[0-9]+", text)
```

```
def encryptDecrypt(mode, message, final = [], keys = []):
```

```
    if mode == 'E':
```

```
        for symbol in message:
```

```
            key = randint(0,25); keys.append(str(key))
```

```
            final.append(str(ord(symbol) ^ key))
```

```
        return ''.join(final), ''.join(keys)
```

```
    else:
```

```
        keys = input("Write the keys: ")
```

```
        for index, symbol in enumerate(regular(message)):
```

```
            final.append(chr(int(symbol) ^ int(regular(keys)[index])))
```

```
        return ''.join(final)
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```

**End\_**

// **terminal**

**Start\_**[

```
$ python main.py
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: Hello World!
```

```
Final message: ('71.115.127.120.127.47.84.98.112.98.115.36',  
'15.22.19.20.16.15.3.13.2.14.23.5')
```

```
$ python main.py
[E]ncrypt[D]ecrypt: d
Write the message: 71.115.127.120.127.47.84.98.112.98.115.36
Write the keys: 15.22.19.20.16.15.3.13.2.14.23.5
Final message: Hello World!
```

**l\_End**

```
# Импортирование функций randint и findall из модуля random и re.
```

```
from random import randint
from re import findall
```

```
# Переключатель режимов шифрования.
```

```
cryptMode = input("[E]ncrypt[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit
```

```
# Сообщение для шифрования/расшифрования.
```

```
startMessage = input("Write the message: ")
```

```
# Регулярное выражение, которое вычерпывает из текста все числа и возвращает
полученный список.
```

```
def regular(text):
    return findall(r"[0-9]+", text)
```

```
# Основная функция шифрования/расшифрования.
```

```
def encryptDecrypt(mode, message, final = [], keys = []):
```

```
# Если переключатель равен 'E', тогда зашифровать сообщение.
```

```
if mode == 'E':
```

```
# Посимвольный перебор сообщения.
```

```
for symbol in message:
```

```
# Создание случайного ключа и добавление ключа в список keys.
```

```
key = randint(0,25); keys.append(str(key))
```

```
# К списку final добавить результат XOR операции между переменной key и
переменной symbol.
```

```
final.append(str(ord(symbol) ^ key))
```

# Вернуть сообщение и ключ.

**return** **'.'.join**(final), **'.'.join**(keys)

# Иначе, если переключатель равен 'D', тогда расшифровать сообщение.

**else:**

# Ввод ключа.

**keys** = **input**("Write the keys: ")

# Перебор всех чисел зашифрованного сообщения.

**for** index, symbol **in enumerate**(**regular**(message)):

# К списку final добавить результат XOR операции между элементом списка keys и переменной symbol.

**final.append**(**chr**(**int**(symbol) ^ **int**(**regular**(keys)[index])))

# Вернуть расшифрованное сообщение.

**return** **'.'.join**(final)

# Вывод получившегося сообщения.

**print**("Final message:",**encryptDecrypt**(cryptMode, startMessage))



## 025. Степени шифра Виженера.

(Многоалфавитные шифры обладают способностью множественного шифрования, то-есть способны шифровать одно и то же сообщение многократно усиливая криптостойкость зашифрованного сообщения.)

// main.py

Start\_

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
try: numberKeys = int(input("How much keys: "))
```

```
except:
```

```
    print("Error: key is not int number")
```

```
    raise SystemExit
```

```
listKeys = []
```

```
for index in range(numberKeys):
```

```
    listKeys.append(input("Write the keyWord["+str(index)+"]: ").upper())
```

```
def encryptDecrypt(mode, message, keys):
```

```
    for key in listKeys:
```

```
        final = ""
```

```
        key *= len(message) // len(key) + 1
```

```
        for index, symbol in enumerate(message):
```

```
            if mode == 'E':
```

```
                temp = ord(symbol) + ord(key[index])
```

```
            else:
```

```
                temp = ord(symbol) - ord(key[index])
```

```
            final += chr(temp % 26 + ord('A'))
```

```
        message = final
```

```
    return final
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage, listKeys))
```

\_End

**// terminal**

**Start\_**

```
$ python main.py
[E]ncrypt|[D]ecrypt: e
Write the message: helloworld
How much keys: 3
Write the keyWord[0]: python
Write the keyWord[1]: good
Write the keyWord[2]: language
Final message: NQFBCXXWVY
```

```
$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the message: NQFBCXXWVY
How much keys: 3
Write the keyWord[0]: python
Write the keyWord[1]: good
Write the keyWord[2]: language
Final message: HELLOWORLD
```

**\_End**

# Переключатель для шифрования.

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
```

# Сообщение для шифрования/расшифрования.

```
startMessage = input("Write the message: ").upper()
```

# Ключ является целым числом, если введено не целое число – вывод ошибки.

```
try: numberKeys = int(input("How much keys: "))
except:
    print("Error: key is not int number")
    raise SystemExit
```

# Создание списка ключей.

```
listKeys = []
```

# Добавление в список ключей.

```
for index in range(numberKeys):  
    listKeys.append(input("Write the keyWord["+str(index)+"]: ").upper())
```

# Создание главной функции.

```
def encryptDecrypt(mode, message, keys):
```

# Перебор всех ключей в списке ключей. Создание переменной final. Подгон длины ключа под длину сообщения.

```
for key in listKeys:  
    final = ""  
    key *= len(message) // len(key) + 1
```

# Посимвольный перебор сообщения.

```
for index, symbol in enumerate(message):
```

# Если переключатель равен шифрованию, значит присваивать к переменной temp сложение.

```
if mode == 'E':  
    temp = ord(symbol) + ord(key[index])
```

# Если переключатель равен расшифрованию, значит присваивать к переменной temp вычитание.

```
else:  
    temp = ord(symbol) - ord(key[index])
```

# Посимвольное добавление в переменную final зашифрованных символов. Присвоить переменную final к переменной message.

```
final += chr(temp % 26 + ord('A'))  
message = final
```

# Вернуть сообщение.

```
return final
```

# Вывод получившегося сообщения.

```
print("Final message:", encryptDecrypt(cryptMode, startMessage, listKeys))
```

## 026. Шифр Штакетник.

(Шифр Штакетник относится к шифрам перестановки и является одним из самых лёгких в своём роде. Принцип шифрования – разделение всех символов на чётные и нечётные позиции, а далее совмещение групп символов, сначала чётные, потом нечётные.)

// main.py

Start\_

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ")
def encryptDecrypt(mode, message, final = ""):
    if mode == 'E':
        encryptList = [
            message[x] for x in range(len(message)) if x%2 == 0],
            message[x] for x in range(len(message)) if x%2 != 0]
        for index in range(len(encryptList)):
            final += "".join(encryptList[index])
    else:
        if len(message)%2 != 0: message += ' '
        length, half = len(message), len(message)//2
        decryptList = [
            message[x] for x in range(half)],
            message[x] for x in range(half,length)]
        for index in range(half):
            final += decryptList[0][index]+decryptList[1][index]
    return final
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End
```

// terminal

Start\_

\$ python main.py

[E]ncrypt|[D]ecrypt: e

Write the message: helloworld

Final message: hloolelwrđ

```
$ python main.py
```

```
[E]ncrypt[D]ecrypt: d
```

```
Write the message: hloolelwrđ
```

```
Final message: helloworld
```

```
]_End
```

```
# Переключатель режимов шифрования
```

```
cryptMode = input("[E]ncrypt[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E', 'D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
# Сообщение для шифрования/расшифрования
```

```
startMessage = input("Write the message: ")
```

```
# Главная функция
```

```
def encryptDecrypt(mode, message, final = ""):
```

```
# Если переключатель равен шифрованию, значит создать списки всех чётных и  
# нечётных символов из сообщения.
```

```
if mode == 'E':
```

```
    encryptList = [
```

```
        [message[x] for x in range(len(message)) if x%2 == 0],
```

```
        [message[x] for x in range(len(message)) if x%2 != 0]
```

```
    ]
```

```
# Перебор всех списков и заполнение переменной final символами.
```

```
for index in range(len(encryptList)):
```

```
    final += "".join(encryptList[index])
```

```
# Если переключатель равен расшифрованию и если длина сообщения не кратно  
# двум, значит добавить в конец сообщения символ-пробел.
```

```
else:
```

```
    if len(message)%2 != 0: message += ' '
```

```
# Создание переменных длины сообщения и половины длины сообщения.
```

```
length, half = len(message), len(message)//2
```

# Создание списков с символами от нуля до середины сообщения и от середины до конца сообщения.

```
decryptList = [  
[message[x] for x in range(half)],  
[message[x] for x in range(half,length)]  
]
```

# Занесение в переменную final сразу двух символов с одинаковым индексом из двух разных списков.

```
for index in range(half):  
    final += decryptList[0][index]+decryptList[1][index]
```

# Вернуть сообщение.

```
return final
```

# Вывод получившегося сообщения.

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```

## 027. Шифр решётки.

(Шифр решётки скрывает открытое сообщение в ложных символах.)

// main.py

Start\_

```
from random import choice, randint
squade = 10
alphaList = [chr(x) for x in range(65,91)] + [chr(y) for y in range(97,123)]
stringList = [choice(alphaList) for _ in range(squade*squade)]
def getKey(text):
    while True:
        keys = [randint(0,99) for _ in range(len(text))]
        for number in keys:
            switch = False
            if keys.count(number) > 1:
                switch = True; break
        if switch == False:
            return keys
def lattice(index = 0):
    print(end = ' ')
    for string in range(squade):
        print(string, end = ' ')
    for string in range(squade):
        print()
        for column in range(squade):
            if index%squade == 0:
                print(index//squade, end = ' | ')
            print(stringList[index], end = ' | ')
            index += 1
    print()
message = input("Write the message: ")
keyList = getKey(message)
keyList.sort()
print("Keys:",keyList)
for index, symbol in enumerate(message):
    del stringList[keyList[index]]
    stringList.insert(keyList[index],symbol)
lattice()
]_End
```

// terminal

Start\_1

\$ python main.py

```
Write the message: helloworld
Keys: [0, 9, 16, 19, 29, 31, 38, 39, 81, 86]
 0  1  2  3  4  5  6  7  8  9
0 | h | A | g | R | D | g | b | v | G | e |
1 | V | Q | r | u | s | E | l | K | X | l |
2 | t | Y | f | a | a | L | y | D | L | o |
3 | f | w | T | Y | i | F | F | T | o | r |
4 | x | Z | W | H | r | J | p | q | Q | y |
5 | t | Q | m | V | p | d | y | k | f | D |
6 | u | A | R | c | O | J | q | q | J | L |
7 | y | L | l | d | o | g | P | M | d | A |
8 | A | l | v | b | q | D | d | O | A | w |
9 | Y | f | J | n | A | q | f | i | Y | b |
```

1\_End

# Импортирование двух функций из модуля random.

**from random import choice, randint**

# Создание переменной, которая является длиной квадрата.

**squade = 10**

# Создание списка символов диапазона A-Z и a-z.

**alphaList = [chr(x) for x in range(65,91)] + [chr(y) for y in range(97,123)]**

# Создание списка с случайно разброшенными символами.

**stringList = [choice(alphaList) for \_ in range(squade\*squade)]**

# Создание функции getKey (Создание ключа).

**def getKey(text):**

# Создание списка случайных ключей по длине сообщения.

**while True:**

**keys = [randint(0,99) for \_ in range(len(text))]**

# Перебор всех чисел в ключе.

**for number in keys:**

**switch = False**



# Если находятся одинаковые ключи, то переменная switch становится равной True и цикл for останавливается.

```
if keys.count(number) > 1:  
    switch = True; break
```

# Если же переменная switch равна False (в ключах нет одинаковых значений), то вернуть ключи.

```
if switch == False:  
    return keys
```

# Создание новой функции lattice (Создание визуальной решётки).

```
def lattice(index = 0):  
    print(end = '  ' )  
    for string in range(squade):  
        print(string, end = '  ' )  
    for string in range(squade):  
        print()  
        for column in range(squade):  
            if index%squade == 0:  
                print(index//squade, end = ' | ' )  
            print(stringList[index], end = ' | ' )  
            index += 1  
    print()
```

# Сообщение для шифрования.

```
message = input("Write the message: ")
```

# Присваивание списка ключей к переменной и их отсортировка.

```
keyList = getKey(message)  
keyList.sort()
```

# Вывод ключей.

```
print("Keys:",keyList)
```

# Добавление в stringList символов сообщения.

```
for index, symbol in enumerate(message):  
    del stringList[keyList[index]]  
    stringList.insert(keyList[index],symbol)  
lattice()
```

## 028. Шифр с омофонами.

(Омофонический шифр построен на шифре замены. Особенность шифра заключена во множестве шифр-символов под один символ текста. То-есть буква 'А' может быть зашифрована разными шифр-символами: '1' или '?' или 'Z'.)

```
// main.py
```

```
Start_1
```

```
from random import choice
values = ('1','2','3','4','5','6','7','8','9','0','a','b','c','\
'd','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','\
't','u','v','w','x','y','z','!','@','\','#','№','$',';','%','^','\
','&','?','(',')','_','+','=','~','[',']','{','\
}','/','|','A','B','C','D','E','F','G','H','J','K','L','\
'M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','<','\
'>','A','M','B','C','y','E','T','a','X','3')
dictHom = {
    'A':values[0:8],    'B':values[8:10],
    'C':values[10:13],  'D':values[13:17],
    'E':values[17:29],  'F':values[29:31],
    'G':values[31:33],  'H':values[33:39],
    'I':values[39:45],  'J':values[45],
    'K':values[46],     'L':values[47:51],
    'M':values[51:53],  'N':values[53:59],
    'O':values[59:66],  'P':values[66:68],
    'Q':values[68],     'R':values[69:75],
    'S':values[75:81],  'T':values[81:90],
    'U':values[90:93],  'V':values[93],
    'W':values[94:96],  'X':values[96],
    'Y':values[97:99],  'Z':values[99]
}
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ")
def encryptDecrypt(mode, message, final = ""):
    if mode == 'E':
        for symbol in message.upper():
            if symbol in dictHom:
```

```

        final += choice(dictHom[symbol])
    else:
        for symbol in message:
            for key in dictHom:
                if symbol in dictHom[key]:
                    final += key
    return final
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End

```

**// terminal**

**Start\_**

```

$ python main.py
[E]ncrypt|[D]ecrypt: e
Write the message: helloworld
Final message: \r)-.y,G?d

```

```

$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the message: \r)-.y,G?d
Final message: HELLOWORLD

```

**]\_End**

# Импорт функции из модуля random.

**from random import choice**

# Значения для шифрования.

```

values = ['1','2','3','4','5','6','7','8','9','0','a','b','c',\
'd','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s',\
't','u','v','w','x','y','z','!','@','\','#','N_e','$',';','%','^',\
':','&','?','(',')','-','_','+','=','\'','~','[','l','{',\
'}',',','.','/','|','A','B','C','D','E','F','G','H','J','K','L',\
'M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','<','\
'>','A','M','B','C','y','E','T','a','X','3']

```

# Словарь с ключами и значениями-списками.

```

dictHom = {
    'A':values[0:8],  'B':values[8:10],
    'C':values[10:13], 'D':values[13:17],

```

```

'E':values[17:29], 'F':values[29:31],
'G':values[31:33], 'H':values[33:39],
'I':values[39:45], 'J':[values[45]],
'K':[values[46]], 'L':values[47:51],
'M':values[51:53], 'N':values[53:59],
'O':values[59:66], 'P':values[66:68],
'Q':[values[68]], 'R':values[69:75],
'S':values[75:81], 'T':values[81:90],
'U':values[90:93], 'V':[values[93]],
'W':values[94:96], 'X':[values[96]],
'Y':values[97:99], 'Z':[values[99]]
}

# Переключатель шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit

# Сообщение для шифрования/расшифрования
startMessage = input("Write the message: ")

# Главная функция
def encryptDecrypt(mode, message, final = ""):

    # Если переключатель равен шифрованию, значит сделать посимвольный перебор
    # сообщения и если символ будет находится в ключах, значит к переменной final
    # добавить случайно-выбранное значение ключа из списка.
    if mode == 'E':
        for symbol in message.upper():
            if symbol in dictHom:
                final += choice(dictHom[symbol])

    # Если же переключатель равен расшифрованию, значит сделать посимвольный
    # перебор сообщения и перебор всех ключей. И если символ будет находится в
    # значениях ключа, значит к переменной final добавлять сам ключ.
    else:
        for symbol in message:
            for key in dictHom:
                if symbol in dictHom[key]:
                    final += key

```

# Вернуть сообщение.

**return final**

# Вывод получившегося сообщения.

**print("Final message:", encryptDecrypt(cryptMode, startMessage))**

## 029. Невидимый шифр.

(Данный метод шифрования основан на конструкции шифра Бэкона, где вместо символа А находится символ пробела, а вместо символа В - символ табуляции.

Невидимый шифр сочетает в себе элементы криптографии и стеганографии.)

// main.py

Start\_1

```
from re import findall
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
fileName = input("Write the fileName: ")
```

```
dictKeys = {
```

```
    'A': ' ',          'N': '\t\t\t',
```

```
    'B': '\t',         'O': '\t\t\t',
```

```
    'C': '\t\t',       'P': '\t\t\t\t',
```

```
    'D': '\t\t\t',     'Q': '\t\t\t',
```

```
    'E': '\t\t\t\t',   'R': '\t\t\t\t',
```

```
    'F': '\t\t\t\t\t', 'S': '\t\t\t\t\t',
```

```
    'G': '\t\t\t\t\t\t', 'T': '\t\t\t\t\t\t',
```

```
    'H': '\t\t\t\t\t\t\t', 'U': '\t\t\t\t\t\t\t',
```

```
    'I': '\t\t\t\t\t\t\t\t', 'V': '\t\t\t\t\t\t\t\t',
```

```
    'J': '\t\t\t\t\t\t\t\t\t', 'W': '\t\t\t\t\t\t\t\t\t',
```

```
    'K': '\t\t\t\t\t\t\t\t\t\t', 'X': '\t\t\t\t\t\t\t\t\t\t',
```

```
    'L': '\t\t\t\t\t\t\t\t\t\t\t', 'Y': '\t\t\t\t\t\t\t\t\t\t\t',
```

```
    'M': '\t\t\t\t\t\t\t\t\t\t\t\t', 'Z': '\t\t\t\t\t\t\t\t\t\t\t\t',
```

```
}
```

```
def regular(text):
```

```
    return findall(r"[\t ]{5}", text)
```

```
def encryptDecrypt(mode, file, final = ""):
```

```
    if mode == 'E':
```

```
        with open(file, 'w') as f:
```

```
            for symbol in input("Write the message: ").upper():
```

```
                if symbol in dictKeys:
```

```
                    final += dictKeys[symbol]
```

```
            f.write(final)
```

```
        return "File successfully saved"
```

```
    else:
```

```

        with open(file, 'r') as f:
            for cipher in regular(f.read()):
                for key in dictKeys:
                    if cipher == dictKeys[key]:
                        final += key

        return final
print("Final message:", encryptDecrypt(cryptMode, fileName))
]_End

```

**// terminal**

**Start\_**

```

$ python main.py
[E]ncrypt|[D]ecrypt: e
Write the fileName: file.txt
Write the message: helloworld
Final message: File successfully saved

```

```

$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the fileName: file.txt
Final message: HELLOWORLD

```

**]\_End**

**// file.txt**

**Start\_**

**]\_End**

```

# Импортирование функции findall из регулярных выражений.
from re import findall

```

```

# Переключатель режимов шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit

```

# Ввод имени файла.

```
fileName = input("Write the fileName: ")
```

# Словарь вида символ : пять\_символов\_шифра

```
dictKeys = {  
    'A': '  ',      'N': '\t\t\t',  
    'B': ' \t',      'O': '\t\t\t ',  
    'C': ' \t ',      'P': '\t\t\t\t',  
    'D': ' \t\t',      'Q': '\t  ',  
    'E': ' \t ',      'R': '\t \t',  
    'F': ' \t\t',      'S': '\t \t ',  
    'G': ' \t\t ',      'T': '\t \t\t',  
    'H': ' \t\t\t',      'U': '\t \t ',  
    'I': ' \t ',      'V': '\t \t \t',  
    'J': ' \t \t',      'W': '\t \t\t ',  
    'K': ' \t \t ',      'X': '\t \t\t\t',  
    'L': ' \t \t\t',      'Y': '\t\t ',  
    'M': ' \t\t ',      'Z': '\t\t \t',  
}
```

# Функция возвращающая список из пяти элементов содержащих в себе пробелы и табы.

```
def regular(text):  
    return findall(r"[ \t]{5}", text)
```

# Функция шифрования / расшифрования.

```
def encryptDecrypt(mode, file, final = ""):
```

# Если переключатель равен 'E', тогда зашифровать сообщение.

```
if mode == 'E':
```

# Создать файл.

```
with open(file, 'w') as f:
```

# Посимвольный перебор открытого сообщения.

```
for symbol in input("Write the message: ").upper():
```

# Если символ находится в словаре, тогда к переменной final присвоить значение этого символа из словаря.

```
if symbol in dictKeys:  
    final += dictKeys[symbol]
```



# Внести строку final в файл.

**f.write(final)**

# Вернуть подтверждение.

**return "File successfully saved"**

# Иначе, если переключатель равен 'D', тогда расшифровать сообщение.

**else:**

# Открыть файл на чтение.

**with open(file, 'r') as f:**

# Перебирать каждые 5 символов в файле.

**for cipher in regular(f.read()):**

# Перебирать все ключи в словаре.

**for key in dictKeys:**

# Если 5 символов из файла равны значению ключа, тогда к переменной final присвоить сам ключ.

**if cipher == dictKeys[key]:**  
    **final += key**

# Вернуть сообщение.

**return final**

# Вывод получившегося сообщения.

**print("Final message:", encryptDecrypt(cryptMode, fileName))**

### 030. Индексированный шифр.

(Данный метод шифрования создаёт ключ, который будет являться последовательностью символов открытого сообщения без повторений.)

```
// main.py
Start_
from re import findall
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ")
def regular(text):
    return findall(r"[0-9]+", text)
def encryptDecrypt(mode, message, final = "", key = ""):
    if mode == 'E':
        for x in message: key += x if x not in key else ""
        encrypt = {key[x]:str(x) for x in range(len(key))}
        for symbol in message:
            final += encrypt[symbol] + ' '
        return (key, final)
    else:
        key = input("Write the key: ")
        for num in regular(message):
            final += key[int(num)]
        return final
print("Final message:", encryptDecrypt(cryptMode, startMessage))
]_End
```

// terminal

Start\_

\$ python main.py

[E]ncrypt|[D]ecrypt: e

Write the message: Python is an interpreted high-level programming language for general-purpose programming.

Final message: ('Python isaerpdg-lvmuf.', '0 1 2 3 4 5 6 7 8 6 9 5 6 7 5 2 10 11 12 11 10 2 10 13 6 3 7 14 3 15 16 10 17 10 16 6 12 11 4 14 11 9 18 18 7 5 14 6 16 9 5 14 19 9 14

10 6 20 4 11 6 14 10 5 10 11 9 16 15 12 19 11 12 4 8 10 6 12 11 4 14 11 9 18 18 7 5 14 21 ')

\$ python main.py

[E]ncrypt|[D]ecrypt: d

Write the message: 0 1 2 3 4 5 6 7 8 6 9 5 6 7 5 2 10 11 12 11 10 2 10 13 6 3 7 14 3 15 16 10 17 10 16 6 12 11 4 14 11 9 18 18 7 5 14 6 16 9 5 14 19 9 14 10 6 20 4 11 6 14 10 5 10 11 9 16 15 12 19 11 12 4 8 10 6 12 11 4 14 11 9 18 18 7 5 14 21

Write the key: Python isaerpdg-lvmuf.

Final message: Python is an interpreted high-level programming language for general-purpose programming.

**l\_End**

# Импортирование функции findall из регулярных выражений.

**from re import findall**

# Переключатель шифрования.

**cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()**

**if cryptMode not in ['E','D']:**

**print("Error: mode is not found")**

**raise SystemExit**

# Открытое/зашифрованное сообщение.

**startMessage = input("Write the message: ")**

# Функция с регулярным выражением возвращающая все числа в сообщении в виде списка.

**def regular(text):**

**return findall(r"[0-9]+", text)**

# Функция шифрования/расшифрования.

**def encryptDecrypt(mode, message, final = "", key = ""):**

# Если переключатель равен 'E', тогда зашифровать сообщение.

**if mode == 'E':**

# Занесение всех символов текста без повторений в переменную key.

**for x in message: key += x if x not in key else ""**

# Создание словаря вида символ\_текста : индекс\_ключа.

**encrypt = {key[x]:str(x) for x in range(len(key))}**

# Посимвольный перебор сообщения и занесение в переменную final индексы ключа.

```
for symbol in message:  
    final += encrypt[symbol] + ' '
```

# Вернуть ключ и зашифрованное сообщение.

```
return (key, final)
```

# Иначе, если переключатель равен 'D', тогда расшифровать сообщение.

```
else:
```

# Ввод ключа.

```
key = input("Write the key: ")
```

# Перебор всех чисел зашифрованного сообщения и вычисление символов по этим индексам.

```
for num in regular(message):  
    final += key[int(num)]
```

# Вернуть расшифрованное сообщение.

```
return final
```

# Вывод получившегося сообщения.

```
print("Final message: ", encryptDecrypt(cryptMode, startMessage))
```

## 031. Шифровальная машина <Турех>.

(Шифровальная машина <Турех> основана на роторном шифровании и шифре пар. Фактически является шифровальной машиной <Enigma> с убранным изъёмом, который заключался в инволюции (рефлекторах). Именно поэтому <Турех> требует переключателя режимов шифрования.)

// main.py

Start\_1

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
rotors = (
    (10,24,14,12,23,2,7,15,24,2,7,5,22,6,2,1,22,12,6,9,7,2,11,23,14,2),
    (1,7,11,26,12,5,11,20,11,7,18,6,17,18,19,1,13,5,2,9,11,13,6,17,26,24),
    (9,1,21,6,4,19,25,6,17,10,26,1,23,6,1,17,19,17,25,21,3,21,17,1,18,20)
)
switch = {
    'H':'Z', 'S':'N', 'L':'M',
    'P':'Q', 'R':'W', 'X':'Y'
}
def stageOne(message):
    message = list(message)
    for symbol in range(len(message)):
        for key in switch:
            if message[symbol] == key:
                message[symbol] = switch[key]
            elif message[symbol] == switch[key]:
                message[symbol] = key
            else: pass
    return "".join(message)
def stageTwo(mode, message, final = ""):
    X,Y,Z = 2,0,1; x,y,z = 1,2,3
    for symbol in message:
        rotor = rotors[X][x] + rotors[Y][y] + rotors[Z][z]
        if mode == 'E':
            if symbol in [chr(x) for x in range(65,91)]:
                final += chr((ord(symbol) - 13 + rotor)%26 + ord('A'))
```

```

        else: continue
    else:
        final += chr((ord(symbol) - 13 - rotor)%26 + ord('A'))
    if x != 25: x += 1
    else:
        x = 0
        if y != 25: y += 1
        else:
            y = 0
            if z != 25: z += 1
            else: z = 0

    return final
def encryptDecrypt(mode, message):
    if mode == 'E':
        message = stageOne(message)
        message = stageTwo(mode, message)
    else:
        message = stageTwo(mode, message)
        message = stageOne(message)
    return message
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End

```

**// terminal**

**Start\_1**

\$ python main.py

[E]ncrypt/[D]ecrypt: e

Write the message: helloworld

Final message: RSFDVJIRJW

\$ python main.py

[E]ncrypt/[D]ecrypt: d

Write the message: RSFDVJIRJW

Final message: HELLOWORLD

**]\_End**

```

# Переключатель режимов шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit

# Сообщение для шифрования/расшифрования.
startMessage = input("Write the message: ").upper()

# Роторы.
rotors = (
    (10,24,14,12,23,2,7,15,24,2,7,5,22,6,2,1,22,12,6,9,7,2,11,23,14,2),
    (1,7,11,26,12,5,11,20,11,7,18,6,17,18,19,1,13,5,2,9,11,13,6,17,26,24),
    (9,1,21,6,4,19,25,6,17,10,26,1,23,6,1,17,19,17,25,21,3,21,17,1,18,20)
)

# Коммутаторы.
switch = {
    'H':'Z', 'S':'N', 'L':'M',
    'P':'Q', 'R':'W', 'X':'Y'
}

# Функция stageOne выполняет полностью аналогичные действия шифра пар.
def stageOne(message):
    message = list(message)
    for symbol in range(len(message)):
        for key in switch:
            if message[symbol] == key:
                message[symbol] = switch[key]
            elif message[symbol] == switch[key]:
                message[symbol] = key
            else: pass
    return "".join(message)

# Функция stageTwo выполняет полностью аналогичные действия роторного
# шифрования, за исключением возможности переставлять сами роторы местами.
def stageTwo(mode, message, final = ""):
    X,Y,Z = 2,0,1; x,y,z = 1,2,3
    for symbol in message:
        rotor = rotors[X][x] + rotors[Y][y] + rotors[Z][z]
        if mode == 'E':
            if symbol in [chr(x) for x in range(65,91)]:

```

```

        final += chr((ord(symbol) - 13 + rotor)%26 + ord('A'))
    else: continue
else:
    final += chr((ord(symbol) - 13 - rotor)%26 + ord('A'))
if x != 25: x += 1
else:
    x = 0
    if y != 25: y += 1
    else:
        y = 0
        if z != 25: z += 1
        else: z = 0

return final

```

# Главная функция служит переключателем режимов шифрования.

```

def encryptDecrypt(mode, message):
    if mode == 'E':
        message = stageOne(message)
        message = stageTwo(mode, message)
    else:
        message = stageTwo(mode, message)
        message = stageOne(message)
    return message

```

# Вывод получившегося сообщения.

```

print("Final message:", encryptDecrypt(cryptMode, startMessage))

```



## 032. Шифр двойной цифри.

(Двойная цифирь является шифром перестановки по определённому ключу. Принцип шифрования – разделение текста на две части в виде таблицы образуя тем самым пары символов. Далее пары символов индексируются и переставляются местами. В итоге индексы будут являться ключами.)

// main.py

Start\_

```
from re import findall
from random import choice
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = list(input("Write the message: ").upper())
def regular(text):
    return findall(r"[0-9]+", text)
def encryptDecrypt(mode, message, final = "", key = []):
    if mode == 'E':
        if len(message) % 2 != 0: message.append(' ')
        listHalf = [
            message[x] for x in range(len(message)//2, len(message)),
            message[y] for y in range(len(message)//2)]
        keys = {x:[listHalf[0][x],listHalf[1][x]] for x in range(len(message)//2)}
        listKey = [x for x in range(len(keys))]
        newList = []
        for _ in range(len(keys)):
            choiceKey = choice(listKey); key.append(str(choiceKey))
            newList.append(keys[choiceKey]); listKey.remove(choiceKey)
        for listIndex in range(len(newList)):
            for symbol in newList[listIndex]:
                final += symbol
        return final, ''.join(key)
    else:
        listHalf = [
            message[x] for x in range(len(message) if x%2 != 0),
            message[y] for y in range(len(message) if y%2 == 0)]
        key = regular(input("Write the key: "))
        key = [int(x) for x in key]
```

```

        keys = {y:[listHalf[0][x],listHalf[1][x]] for x,y in enumerate(key)}
        finalList = [
[keys[x][0] for x in range(len(keys)) if x in keys],
[keys[y][1] for y in range(len(keys)) if y in keys]]
        for i in range(2):
            for index in range(len(message)//2):
                final += finalList[i][index]
        return final
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End

```

**// terminal**

**Start\_**

\$ python main.py

[E]ncrypt|[D]ecrypt: e

Write the message: helloworld

Final message: ('OEDOWHRLLL', '1.4.0.2.3')

\$ python main.py

[E]ncrypt|[D]ecrypt: d

Write the message: OEDOWHRLLL

Write the key: 1.4.0.2.3

Final message: HELLOWORLD

**]\_End**

# Импортирование функций из модулей re и random.

**from re import findall**

**from random import choice**

# Переключатель режимов шифрования

**cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()**

**if cryptMode not in ['E','D']:**

**print("Error: mode is not found")**

**raise SystemExit**

# Сообщение для шифрования/расшифрования.

**startMessage = list(input("Write the message: ").upper())**

# Функция для создания списка с числами.

```
def regular(text):  
    return findall(r"[0-9]+", text)
```

# Создание главной функции.

```
def encryptDecrypt(mode, message, final = "", key = []):
```

# Если переключатель равен шифрованию и если длина сообщения не кратна двум, тогда добавить в конец сообщения символ Z.

```
if mode == 'E':  
    if len(message) % 2 != 0: message.append('Z')
```

# Создать два списка. В первом все символы от середины до конца сообщения. Во втором все символы от начала до середины сообщения.

```
listHalf = [  
    [message[x] for x in range(len(message)//2, len(message))],  
    [message[y] for y in range(len(message)//2)]
```

# Создание словаря ключей со списком в виде значений. Создание списка ключей.

```
keys = {x:[listHalf[0][x],listHalf[1][x]] for x in range(len(message)//2)}  
listKey = [x for x in range(len(keys))]
```

# Создание списка для добавления последовательности ключей.

```
NewList = []
```

# Перебор всех ключей и выбор случайных из списка listKey.

```
for _ in range(len(keys)):  
    choiceKey = choice(listKey); key.append(str(choiceKey))  
    newList.append(keys[choiceKey]); listKey.remove(choiceKey)
```

# Перебор всех ключей и всех символов в списке newList, и занесение получившихся символов в переменную final.

```
for listIndex in range(len(newList)):  
    for symbol in newList[listIndex]:  
        final += symbol
```

# Вернуть сообщение и ключи.

```
return final, ' '.join(key)
```

# Если переключатель равен расшифрованию, тогда создать два списка. В первом списке все нечётные символы сообщения, во втором все чётные символы сообщения.

**else:**

```
listHalf = [  
[message[x] for x in range(len(message)) if x%2 != 0],  
[message[y] for y in range(len(message)) if y%2 == 0]]
```

# Указание ключей и перевод их в целые значения.

```
key = regular(input("Write the key: "))
```

```
key = [int(x) for x in key]
```

# Создание словаря с ключами и списком значений ключа.

```
keys = {y:[listHalf[0][x],listHalf[1][x]] for x,y in enumerate(key)}
```

# Создание двух словарей. В первом находятся все первые элементы значений словаря key. Во втором находятся все вторые элементы значений словаря key.

```
finalList = [  
[keys[x][0] for x in range(len(keys)) if x in keys],  
[keys[y][1] for y in range(len(keys)) if y in keys]]
```

# Занесение в переменную final всех символов двух предыдущих списков.

```
for i in range(2):  
    for index in range(len(message)//2):  
        final += finalList[i][index]  
return final
```

# Вывод получившегося сообщения.

```
print("Final message:",encryptDecrypt(cryptMode, startMessage))
```

### 033. Шифр Плейфера.

(Шифр Плейфера является биграммным шифром. Особенность шифра заключена в матрице 5x5 в которой мы заменяем символы нашего сообщения. Принцип шифрования – если два символа находятся на одной строке матрицы, значит сдвинуть индексы этих двух символов на одну позицию вперёд в этой же строке. Если символы находятся на разных строках, значит прочертить прямоугольник и заменить символы на другие края прямоугольника.)

// main.py

Start\_1

```
from re import findall
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = list(input("Write the message: ").upper())
matrixKey = [
    ['S','O','M','E','T'],
    ['H','I','N','G','A'],
    ['B','C','D','F','K'],
    ['L','P','Q','R','U'],
    ['V','W','X','Y','Z']
]; addSymbol = 'X'
def regular(text):
    return findall(r"[A-Z]{2}", text)
def encryptDecrypt(mode, message, final = ""):
    if mode == 'E':
        for symbol in message:
            if symbol not in [chr(x) for x in range(65,91)]:
                message.remove(symbol)
        for index in range(len(message)):
            if message[index] == 'J': message[index] = 'I'
        for index in range(1,len(message)):
            if message[index] == message[index - 1]:
                message.insert(index,addSymbol)
        if len(message) % 2 != 0:
            message.append(addSymbol)
    binaryList = regular("".join(message))
    for binary in range(len(binaryList)):
```

```

        binaryList[binary] = list(binaryList[binary])
        for indexString in range(len(matrixKey)):
            for indexSymbol in range(len(matrixKey[indexString])):
                if binaryList[binary][0] == matrixKey[indexString]
[indexSymbol]:
                    y0, x0 = indexString, indexSymbol
                    if binaryList[binary][1] == matrixKey[indexString]
[indexSymbol]:
                        y1, x1 = indexString, indexSymbol
                        for indexString in range(len(matrixKey)):
                            if matrixKey[y0][x0] in matrixKey[indexString] and
matrixKey[y1][x1] in matrixKey[indexString]:
                                if mode == 'E':
                                    x0 = x0 + 1 if x0 != 4 else 0
                                    x1 = x1 + 1 if x1 != 4 else 0
                                else:
                                    x0 = x0 - 1 if x0 != 0 else 4
                                    x1 = x1 - 1 if x1 != 0 else 4
                        y0,y1 = y1,y0
                        binaryList[binary][0] = matrixKey[y0][x0]
                        binaryList[binary][1] = matrixKey[y1][x1]
        for binary in range(len(binaryList)):
            for symbol in binaryList[binary]:
                final += symbol
    return final
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End

```

**// terminal**

**Start\_1**

\$ python main.py

[E]ncrypt/[D]ecrypt: e

Write the message: helloworld

Final message: SGVQSPowUPXD

\$ python main.py

[E]ncrypt/[D]ecrypt: d

Write the message: SGVQSPowUPXD

Final message: HELXLOWORLDX

**]**\_End

# Импорт функции из модуля re.

**from re import findall**

# Переключатель режимов шифрования.

**cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()**

**if cryptMode not in ['E','D']:**

**print("Error: mode is not found")**

**raise SystemExit**

# Сообщение для шифрования/расшифрования.

**startMessage = list(input("Write the message: ").upper())**

# Создание матрицы-ключа и символа для редактирования текста.

**matrixKey = [**

**['S','O','M','E','T'],**

**['H','T','N','G','A'],**

**['B','C','D','F','K'],**

**['L','P','Q','R','U'],**

**['V','W','X','Y','Z']**

**]; addSymbol = 'X'**

# Функция возвращающая пары символов.

**def regular(text):**

**return findall(r"[A-Z]{2}", text)**

# Создание главной функции.

**def encryptDecrypt(mode, message, final = ""):**

# Перебор всех символов сообщения и если символ не находится в диапазоне A-Z, то удалить этот символ.

**if mode == 'E':**

**for symbol in message:**

**if symbol not in [chr(x) for x in range(65,91)]:**

**message.remove(symbol)**

# Перебор сообщения и если символ равен символу 'J', то заменить его на 'I'.

**for index in range(len(message)):**

**if message[index] == 'J': message[index] = 'I'**

# Перебор символов сообщения и если стоит два одинаковых символа вместе, то между ними поставить символ 'X'.

```
for index in range(1,len(message)):
    if message[index] == message[index - 1]:
        message.insert(index,addSymbol)
```

# Если длина сообщения не кратна двум, то добавить в конец сообщения символ 'X'.

```
if len(message) % 2 != 0:
    message.append(addSymbol)
```

# Создание списка из пар символов.

```
binaryList = regular("".join(message))
```

# Перебор всех пар символов.

```
for binary in range(len(binaryList)):
    binaryList[binary] = list(binaryList[binary])
```

# Перебор всех строк и всех символов в матрице.

```
for indexString in range(len(matrixKey)):
    for indexSymbol in range(len(matrixKey[indexString])):
```

# Если первый символ пары будет равняться символу из матрицы, тогда запомнить координаты первого символа пары.

```
if binaryList[binary][0] == matrixKey[indexString][indexSymbol]:
    y0, x0 = indexString, indexSymbol
```

# Если второй символ пары будет равняться символу из матрицы, тогда запомнить координаты второго символа пары.

```
if binaryList[binary][1] == matrixKey[indexString][indexSymbol]:
    y1, x1 = indexString, indexSymbol
```

# Перебор всех строк в матрице и если первый и второй символ находятся на одной и той же строке матрицы, тогда работает переключатель режимов шифрования.

```
for indexString in range(len(matrixKey)):
    if matrixKey[y0][x0] in matrixKey[indexString] and matrixKey[y1][x1] in
matrixKey[indexString]:
```



# Если переключатель равен шифрованию, тогда перемещать позицию элементов пары по горизонтали на одну позицию вперёд. Если же позиция на которой стоит символ является последней, то позиция перемещается на нулевой элемент строки.

**if mode == 'E':**

**x0 = x0 + 1 if x0 != 4 else 0**

**x1 = x1 + 1 if x1 != 4 else 0**

# Если переключатель равен расшифрованию, тогда перемещать позицию элементов пары по горизонтали на одну позицию назад. Если же позиция на которой стоит символ является нулевой, то позиция перемещается на последний элемент строки.

**else:**

**x0 = x0 - 1 if x0 != 0 else 4**

**x1 = x1 - 1 if x1 != 0 else 4**

# Две переменные меняют значения между собой.

**y0,y1 = y1,y0**

# Замена позиций пары в матрице.

**binaryList[binary][0] = matrixKey[y0][x0]**

**binaryList[binary][1] = matrixKey[y1][x1]**

# Перебор всех пар и посимвольный ввод в переменную final.

**for binary in range(len(binaryList)):**

**for symbol in binaryList[binary]:**

**final += symbol**

# Вернуть сообщения.

**return final**

# Вывод получившегося сообщения.

**print("Final message:",encryptDecrypt(cryptMode, startMessage))**

## 034. Шифр ADFGVX.

(Шифр ADFGVX состоит из шифров замены и шифров перестановки. Шифр замены – аналогия с шифром Полибия. Шифр перестановки работает по ключу.)

```
// main.py
Start_[]
from re import findall
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
keyStageTwo = list(input("Write the key: ").upper())
for symbol in keyStageTwo:
    if keyStageTwo.count(symbol) > 1:
        keyStageTwo.remove(symbol)
keyStageOne = {
    'A':'AA','N':'FD','O':'VF',
    'B':'AD','O':'FF','1':'VG',
    'C':'AF','P':'FG','2':'VV',
    'D':'AG','Q':'FV','3':'VX',
    'E':'AV','R':'FX','4':'XA',
    'F':'AX','S':'GA','5':'XD',
    'G':'DA','T':'GD','6':'XF',
    'H':'DD','U':'GF','7':'XG',
    'I':'DF','V':'GG','8':'XV',
    'J':'DG','W':'GV','9':'XX',
    'K':'DV','X':'GX',
    'L':'DX','Y':'VA',
    'M':'FA','Z':'VD',
}
def regular(text):
    return findall(r"[A-Z]{2}", text)
def stageOne(mode, message, final = ""):
    if mode == 'E':
        for symbol in message:
            if symbol in keyStageOne:
                final += keyStageOne[symbol]
    else:
```

```

    for symbols in regular(message):
        for key in keyStageOne:
            if symbols == keyStageOne[key]:
                final += key
    return final
def stageTwo(mode, message, final = "", listCutWords = []):
    if mode == 'E':
        while len(message) % len(keyStageTwo) != 0:
            message += 'XX'
        lengthList = len(message) // len(keyStageTwo)
        for _ in range(lengthList):
            listCutWords.append([])
        index = 0; counter = 1
        for symbol in message:
            if counter % len(keyStageTwo) != 0:
                listCutWords[index].append(symbol)
                counter += 1
            else:
                listCutWords[index].append(symbol)
                index += 1; counter = 1
        keys = {x:[] for x in keyStageTwo}
        index = 0
        for key in keyStageTwo:
            for x in range(len(listCutWords)):
                keys[key].append(listCutWords[x][index])
            index += 1
        keySort = list(keyStageTwo); keySort.sort()
        keys = {key:keys[key] for key in keySort if key in keys}
        for listSymbol in keys:
            for symbol in keys[listSymbol]:
                final += symbol
    else:
        keySort = list(keyStageTwo); keySort.sort()
        lengthList = len(message) // len(keyStageTwo)
        for _ in range(len(keyStageTwo)):
            listCutWords.append([])
        index = 0; counter = 1
        for symbol in message:
            if counter % lengthList != 0:

```

```

        listCutWords[index].append(symbol)
        counter += 1
    else:
        listCutWords[index].append(symbol)
        index += 1; counter = 1
    keys = {keySort[symbol]:listCutWords[symbol] for symbol in
range(len(keySort))}
    keys = {key:keys[key] for key in keyStageTwo if key in keys}
    index = 0
    for _ in range(lengthList):
        for symbolOne in keys:
            final += keys[symbolOne][index]
        index += 1
    return final
def encryptDecrypt(mode, message):
    if mode == 'E':
        message = stageOne(mode, message)
        message = stageTwo(mode, message)
    else:
        message = stageTwo(mode, message)
        message = stageOne(mode, message)
    return message
print("Final message:",encryptDecrypt(cryptMode, startMessage))
]_End

```

**// terminal**

**Start\_**

```

$ python main.py
[E]ncrypt|[D]ecrypt: e
Write the message: helloworld
Write the key: delta
Final message: DFFGDXXGDDVDAXFXVFFA

```

```

$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the message: DFFGDXXGDDVDAXFXVFFA
Write the key: delta
Final message: HELLOWORLD

```

**]\_End**

# Импорт функции из модуля re.

```
from re import findall
```

# Переключатель режимов шифрования.

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E', 'D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

# Сообщение для шифрования/расшифрования.

```
startMessage = input("Write the message: ").upper()
```

# Создание ключа.

```
keyStageTwo = list(input("Write the key: ").upper())
```

# Посимвольный перебор ключа и если символ в ключе встречается больше одного раза, тогда удалить этот символ.

```
for symbol in keyStageTwo:
```

```
    if keyStageTwo.count(symbol) > 1:
```

```
        keyStageTwo.remove(symbol)
```

# Создание словаря для шифрования (шифр Полибия).

```
keyStageOne = {
```

```
    'A': 'AA', 'N': 'FD', '0': 'VF',
```

```
    'B': 'AD', 'O': 'FF', '1': 'VG',
```

```
    'C': 'AF', 'P': 'FG', '2': 'VV',
```

```
    'D': 'AG', 'Q': 'FV', '3': 'VX',
```

```
    'E': 'AV', 'R': 'FX', '4': 'XA',
```

```
    'F': 'AX', 'S': 'GA', '5': 'XD',
```

```
    'G': 'DA', 'T': 'GD', '6': 'XF',
```

```
    'H': 'DD', 'U': 'GF', '7': 'XG',
```

```
    'I': 'DF', 'V': 'GG', '8': 'XV',
```

```
    'J': 'DG', 'W': 'GV', '9': 'XX',
```

```
    'K': 'DV', 'X': 'GX',
```

```
    'L': 'DX', 'Y': 'VA',
```

```
    'M': 'FA', 'Z': 'VD',
```

```
}
```

# Функция возвращающая пары символов.

```
def regular(text):
```

```
    return findall(r"[A-Z]{2}", text)
```

# Функция stageOne является аналогом шифра Полибия.

```
def stageOne(mode, message, final = ""):
    if mode == 'E':
        for symbol in message:
            if symbol in keyStageOne:
                final += keyStageOne[symbol]
    else:
        for symbols in regular(message):
            for key in keyStageOne:
                if symbols == keyStageOne[key]:
                    final += key
    return final
```

# Основная функция шифрования.

```
def stageTwo(mode, message, final = "", listCutWords = []):
```

# Если переключатель равен шифрованию, тогда сделать цикл, при котором будут добавляться символы 'XX' пока длина сообщения не будет делиться на длину ключа без остатка.

```
if mode == 'E':
    while len(message) % len(keyStageTwo) != 0:
        message += 'XX'
```

# Длина полного деления сообщения на ключ.

```
lengthList = len(message) // len(keyStageTwo)
```

# Добавление в список вложенных списков.

```
for _ in range(lengthList):
    listCutWords.append([])
```

# Добавление во вложенные списки символов по длине ключа.

```
index = 0; counter = 1
for symbol in message:
    if counter % len(keyStageTwo) != 0:
        listCutWords[index].append(symbol)
        counter += 1
    else:
        listCutWords[index].append(symbol)
        index += 1; counter = 1
```

# Создание словаря.

```
keys = {x:[] for x in keyStageTwo}
```

# Добавление в словарь на место значений ключа символов.

**index = 0**

**for key in keyStageTwo:**

**for x in range(len(listCutWords)):**

**keys[key].append(listCutWords[x][index])**

**index += 1**

# Сортировка ключа по алфавиту.

**keySort = list(keyStageTwo); keySort.sort()**

**keys = {key:keys[key] for key in keySort if key in keys}**

# Перебор всех всех символов значений ключа и добавление их к переменной final.

**for listSymbol in keys:**

**for symbol in keys[listSymbol]:**

**final += symbol**

# Если же переключатель равен расшифрованию, тогда отсортировать ключи и вычислить длину вложенных списков.

**else:**

**keySort = list(keyStageTwo); keySort.sort()**

**lengthList = len(message) // len(keyStageTwo)**

# Создание вложенных списков.

**for \_ in range(len(keyStageTwo)):**

**listCutWords.append([])**

# Добавление во вложенные списки символов по длине ключа.

**index = 0; counter = 1**

**for symbol in message:**

**if counter % lengthList != 0:**

**listCutWords[index].append(symbol)**

**counter += 1**

**else:**

**listCutWords[index].append(symbol)**

**index += 1; counter = 1**

# Создание ключей и их отсортировка.

**keys = {keySort[symbol]:listCutWords[symbol] for symbol in range(len(keySort))}**

**keys = {key:keys[key] for key in keyStageTwo if key in keys}**

```
# Перебор всех ключей и добавление в переменную final всех символов.  
index = 0  
for _ in range(lengthList):  
    for symbolOne in keys:  
        final += keys[symbolOne][index]  
    index += 1  
  
# Вернуть сообщение.  
return final  
  
# Главная функция, которая является переключателем режимов шифрования.  
def encryptDecrypt(mode, message):  
    if mode == 'E':  
        message = stageOne(mode, message)  
        message = stageTwo(mode, message)  
    else:  
        message = stageTwo(mode, message)  
        message = stageOne(mode, message)  
    return message  
  
# Вывод получившегося сообщения.  
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```



### 035. AES шифрование (модуль).

(AES шифрование позволяет зашифровывать файлы любого формата. В данном примере я не расписывал скрипт с нуля, а установил готовый модуль pyAesCrypt. Библиотека pyAesCrypt использует AES256-CBC.

Осторожно: Файл перезаписывается!)

// **main.py**

**Start\_**[

```
from pyAesCrypt import encryptFile, decryptFile
from os import remove
from os.path import splitext
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
fileName = input("Write the file: ")
paswFile = input("Write the password: ")
bufferSize = 64*1024
def encryptDecrypt(mode, file, password, final = ""):
    if mode == 'E':
        try:
            encryptFile(str(file), str(file)+".crp", password, bufferSize)
            remove(file)
        except FileNotFoundError: return "[x] File not found!"
        else: return "[+] File '{name}' overwritten!".format(name = str(file))
    else:
        try:
            decryptFile(str(file), str(splitext(file)[0]), password, bufferSize)
            remove(file)
        except FileNotFoundError: return "[x] File not found!"
        except ValueError: return "[x] Password is False!"
        else: return "[+] File '{name}' overwritten!".format(name = str(file))
print(encryptDecrypt(cryptMode, fileName, paswFile))
]_End
```

**// terminal**

**Start\_**

```
$ python main.py  
[E]ncrypt|[D]ecrypt: e  
Write the file: book.txt  
Write the password: helloworld  
[+] File 'book.txt' overwritten!
```

```
$ python main.py  
[E]ncrypt|[D]ecrypt: d  
Write the file: book.txt.crp  
Write the password: helloworld  
[+] File 'book.txt.crp' overwritten!
```

**\_End**

```
# Импортирование функций из моделей aes и os.  
from pyAesCrypt import encryptFile, decryptFile  
from os import remove  
from os.path import splitext  
  
# Переключатель режимов шифрования.  
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()  
if cryptMode not in ['E','D']:  
    print("Error: mode is not found")  
    raise SystemExit  
  
# Имя файла и пароль для шифрования/расшифрования.  
fileName = input("Write the file: ")  
paswFile = input("Write the password: ")  
  
# Размер шифрования/расшифрования данных.  
bufferSize = 64*1024  
  
# Главная функция.  
def encryptDecrypt(mode, file, password, final = ""):
```

# Если переключатель равен шифрованию, тогда попытаться зашифровать файл и удалить открытую копию. Если же файл не был найден – вывести ошибку. Но если же всё удачно прошло, тогда написать сообщение о перезаписи файла.

**if mode == 'E':**

**try:**

**encryptFile(str(file), str(file)+".crp", password, bufferSize)**  
**remove(file)**

**except FileNotFoundError: return "[x] File not found!"**

**else: return "[+] File '{name}' overwritten!".format(name = str(file))**

# Если же переключатель равен расшифрованию, тогда попытаться расшифровать файл и удалить зашифрованную копию. Если же файл не был найден или пароль не верный – вывести ошибку. Но если же всё удачно прошло, тогда написать сообщение о перезаписи файла.

**else:**

**try:**

**decryptFile(str(file), str(splitext(file)[0]), password, bufferSize)**  
**remove(file)**

**except FileNotFoundError: return "[x] File not found!"**

**except ValueError: return "[x] Password is False!"**

**else: return "[+] File '{name}' overwritten!".format(name = str(file))**

# Вывод результата.

**print(encryptDecrypt(cryptMode, fileName, paswFile))**

## 036. Великий Шифр.

(Великий Шифр собрал в себя пять различных методов шифрования: омофоническое шифрование, кодирование, замена слогов, ложные символы, специальные символы (символы редактирования текста). Шифр является одним из самых криптостойких.)

// main.py

Start\_

```
from memory import Key, Limit
```

```
from random import randint, choice
```

```
from re import findall
```

```
keysCrypt = {
```

```
    'A':Key[0:8],      'B':Key[8:10],
    'C':Key[10:13],    'D':Key[13:17],
    'E':Key[17:29],    'F':Key[29:31],
    'G':Key[31:33],    'H':Key[33:39],
    'I':Key[39:45],    'J':Key[45:],
    'K':Key[46:],       'L':Key[47:51],
    'M':Key[51:53],    'N':Key[53:59],
    'O':Key[59:66],    'P':Key[66:68],
    'Q':Key[68:],       'R':Key[69:75],
    'S':Key[75:81],    'T':Key[81:90],
    'U':Key[90:93],    'V':Key[93:],
    'W':Key[94:96],    'X':Key[96:],
    'Y':Key[97:99],    'Z':Key[99:],
    ' ':Key[100:118]
```

```
}
```

```
listWord = ('TO','WHY','WITH','WAR','NOT','IN','OR','ELSE','THE','THAT','BY',
'AND','HOW','BUT','IF','ONE','YOU','ME','USE','HIS','YOUR','ON','OF','WAS','BE',
'THIS','WHAT','THEY','NO','YES','TRUE','FALSE','CALL','FEEL','CLOSE','VERY',
'WHICH','CAR','ANY','HOLD','WORK','RUN','NEVER','START','EVEN','LIGHT','THAN',
'AFTER','PUT','STOP','OLD','WATCH','FIRST','MAY','TALK','ANOTHER','BEHIND','CUT',
'MEAN','SMILE','OUR','MUCH','IT','HE','SHE','ITS','HOUSE','KEEP','YEAH','PLACE',
'BEGIN','NOTHING','YEAR','MAN','WOMAN','BECAUSE','THREE','SEEM','ARE','WAIT',
'NEED','LAST','LATE','SURE','BIG','SMALL','FRONT','REALLY','NAME','ALL','NEW','G',
'UY','ANYTHING','SHOULD','KILL','POINT','WALL','BLACK',
'STEP','SECOND','LIFE','MAYBE','FALL','OWN','FAR','WHILE','FOR','HELP','END',
```

```

'THOSE','SAME','REACH','GIRL','STREET','NEXT','FEW','FEET','SHOW','MUST','TABLE',
'E','OK','IS','OKAY','BODY','PHONE','ADD','WATER','FIRE','INSIDE','BREAK','EVER','SH',
'AKE','MEET','GREAT','MIND','ENOUGH','MINUTE','FOLLOW','ATTACK',
'DEAD','ALMOST')
position = 118
keyCode = {listWord[x]:Key[x + position] for x in range(len(listWord))}
listSyllables = ('TH','WH','EE','AI','OO','IS','ING','ED','BE','ON','OR','ER',
'CH','SH','GH','EN','EA','OU','LL','US','SE','AL','ST','EV','WO','UI','IN','RE',
'!','?','.',',','@','#','$','%','*','^','-',',','+','=','/',':',';','&','~')
position = len(listWord) + 118
keySyllables = {listSyllables[x]:Key[x + position] for x in
range(len(listSyllables))}
listSpecial = ('<-', '->', '<+', '+>')
position = len(listWord) + len(listSyllables) + 118
keySpecial = {listSpecial[x]:Key[x + position] for x in range(len(listSpecial))}
position = len(listWord) + len(listSyllables) + len(listSpecial) + 118
traps = tuple([Key[x] for x in range(position, Limit)])
del listWord, listSpecial, position
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
def regular(text):
    return findall(r"[0-9]{3}", text)
def encryptDecrypt(mode, message, final = "", string = ""):
    if cryptMode == 'E':
        secondText = findall(r"^[^\s]+", message)
        del message
        for indexWord in range(len(secondText)):
            if secondText[indexWord] in keySpecial:
                secondText[indexWord] =
keySpecial[secondText[indexWord]]
            for indexWord in range(len(secondText)):
                if secondText[indexWord] in keyCode:
                    secondText[indexWord] =
keyCode[secondText[indexWord]]
            for indexWord in range(len(secondText)):
                for syllable in keySyllables:

```

```

        if syllable in secondText[indexWord]:
            secondText[indexWord] =
secondText[indexWord].replace(syllable,keysSyllables[syllable])
        for indexWord in range(len(secondText)):
            secondText[indexWord] = list(secondText[indexWord])
        for indexWord in range(len(secondText)):
            secondText[indexWord].append(' ')
        for indexWord in range(len(secondText)):
            for indexSymbol in range(len(secondText[indexWord])):
                symbol = secondText[indexWord][indexSymbol]
                if symbol in keysCrypt:
                    length = len(keysCrypt[symbol])
                    secondText[indexWord][indexSymbol] =
keysCrypt[symbol][randint(0, length - 1)]
            for word in secondText:
                string += "".join(word)
            finalList = list(regular(string))
            for indexList in range(len(finalList)):
                randSwitch = randint(0,2); randPosition =
randint(0,len(finalList))
                if not randSwitch: finalList.insert(randPosition,choice(traps))
            for word in finalList:
                final += "".join(word)
            return ".".join(regular(final))
    else:
        for symbolText in regular(message):
            for element in keysSpecial:
                if symbolText == keysSpecial[element]: final += element
            for word in keysCode:
                if symbolText == keysCode[word]: final += word
            for syllable in keysSyllables:
                if symbolText == keysSyllables[syllable]: final += syllable
            for symbol in keysCrypt:
                if symbolText in keysCrypt[symbol]: final += symbol
        listWord = findall(r"[^\s]+",final)
        for _ in range(len(listWord)):
            for element in keysSpecial:
                if element in listWord:
                    if element == '<-':

```

```

        del listWord[listWord.index(element) - 1]
        listWord.remove(element)
    elif element == '->':
        del listWord[listWord.index(element) + 1]
        listWord.remove(element)
    elif element == '<+':
        listWord[listWord.index(element)] =
listWord[listWord.index(element) - 1]
    elif element == '+>':
        listWord[listWord.index(element)] =
listWord[listWord.index(element) + 1]
    else: pass
    return " ".join(listWord)
print("Final message:", encryptDecrypt(cryptMode, startMessage))
]_End

```

**// memory.py**

**Start\_**

```

Limit = 350; Key = (
'275','078','276','230','343','127','006','325','307','102',
'334','185','004','002','008','283','277','260','256','305',
'300','143','159','248','160','309','104','222','136','317',
'264','053','218','137','177','315','301','244','040','072',
'023','182','323','154','076','048','213','330','109','164',
'257','179','043','166','207','240','220','205','228','073',
'017','093','186','027','157','080','009','195','289','278',
'116','238','028','271','058','001','132','236','044','029',
'282','339','131','069','100','107','255','135','226','198',
'335','035','187','119','293','133','270','347','097','273',
'068','250','311','140','212','120','024','303','188','103',
'209','114','034','267','061','184','261','225','349','167',
'059','106','272','168','326','165','269','279','019','247',
'087','156','239','088','092','026','071','304','241','180',
'306','284','310','065','021','234','162','202','060','047',
'039','318','321','152','117','324','067','348','031','308',
'265','070','181','077','217','041','032','124','125','290',
'235','037','231','003','342','158','020','322','216','091',
'176','079','199','259','129','138','113','099','312','215',

```

```
'115','145','122','193','254','314','153','344','123','340',  
'292','101','083','286','262','183','121','237','242','249',  
'246','139','018','042','295','178','245','022','148','163',  
'052','233','189','229','090','000','341','015','346','336',  
'082','345','281','036','005','089','038','064','141','046',  
'151','062','266','243','010','287','066','149','030','095',  
'147','171','173','332','055','045','253','298','227','299',  
'051','170','331','280','224','011','111','105','112','130',  
'192','268','169','128','033','098','075','194','211','085',  
'223','025','296','196','155','320','054','126','174','108',  
'118','144','172','302','146','294','190','012','210','086',  
'191','014','208','142','084','337','203','219','201','327',  
'197','057','221','016','338','206','333','134','150','063',  
'251','291','313','285','204','274','319','013','056','328',  
'214','263','200','096','288','007','252','161','050','110',  
'329','049','258','081','094','074','297','316','232','175',  
)
```

**l\_End**

**// terminal**

**Start\_**

```
$ python main.py
```

```
[E]ncrypt[D]ecrypt: e
```

```
Write the message: -> hello world <+
```

```
Final message: 014.250.177.260.169.093.184.133.333.011.109.277.120.208.291.225
```

```
$ python main.py
```

```
[E]ncrypt[D]ecrypt: d
```

```
Write the message: 014.250.177.260.169.093.184.133.333.011.109.277.120.208.291.225
```

```
Final message: WORLD WORLD
```

**l\_End**

```
# Импортирование функций из модуля random и re. Импортирование переменных  
из модуля memory.
```

```
from memory import Key, Limit
```

```
from random import randint, choice
```

```
from re import findall
```



# Омофоническое шифрование.

```
keysCrypt = {  
    'A':Key[0:8],    'B':Key[8:10],  
    'C':Key[10:13], 'D':Key[13:17],  
    'E':Key[17:29], 'F':Key[29:31],  
    'G':Key[31:33], 'H':Key[33:39],  
    'I':Key[39:45], 'J':[Key[45]],  
    'K':[Key[46]],  'L':Key[47:51],  
    'M':Key[51:53], 'N':Key[53:59],  
    'O':Key[59:66], 'P':Key[66:68],  
    'Q':[Key[68]],  'R':Key[69:75],  
    'S':Key[75:81], 'T':Key[81:90],  
    'U':Key[90:93], 'V':[Key[93]],  
    'W':Key[94:96], 'X':[Key[96]],  
    'Y':Key[97:99], 'Z':[Key[99]],  
    ' ':Key[100:118]  
}
```

# Кодирование.

```
listWord =  
(('TO','WHY','WITH','WAR','NOT','IN','OR','ELSE','THE','THAT','BY','AND','  
HOW','BUT','IF','ONE','YOU','ME','USE','HIS','YOUR','ON','OF','WAS','BE','  
THIS','WHAT','THEY','NO','YES','TRUE','FALSE','CALL','FEEL','CLOSE','V  
ERY','WHICH','CAR','ANY','HOLD','WORK','RUN','NEVER','START','EVEN'  
, 'LIGHT','THAN','AFTER','PUT','STOP','OLD','WATCH','FIRST','MAY','TAL  
K','ANOTHER','BEHIND','CUT','MEAN','SMILE','OUR','MUCH','IT','HE','SH  
E','ITS','HOUSE','KEEP','YEAH','PLACE','BEGIN','NOTHING','YEAR','MAN'  
, 'WOMAN','BECAUSE','THREE','SEEM','ARE','WAIT','NEED','LAST','LATE',  
'SURE','BIG','SMALL','FRONT','REALLY','NAME','ALL','NEW','GUY','ANY  
THING','SHOULD','KILL','POINT','WALL','BLACK','STEP','SECOND','LIFE'  
, 'MAYBE','FALL','OWN','FAR','WHILE','FOR','HELP','END','THOSE','SAME'  
, 'REACH','GIRL','STREET','NEXT','FEW','FEET','SHOW','MUST','TABLE',  
'OK','IS','OKAY','BODY','PHONE','ADD','WATER','FIRE','INSIDE','BREAK',  
'EVER','SHAKE','MEET','GREAT','MIND','ENOUGH','MINUTE','FOLLOW',  
'ATTACK','DEAD','ALMOST')
```

# Стартовая позиция (конец шифра с омофонами).

```
position = 118
```

# Создание словаря с + позицией.

```
keysCode = {listWord[x]:Key[x + position] for x in range(len(listWord))}
```

# Шифр замены/шифр с заменой слогов.

```
listSyllables = ('TH','WH','EE','AI','OO','IS','ING','ED','BE','ON','OR','ER',  
'CH','SH','GH','EN','EA','OU','LL','US','SE','AL','ST','EV','WO','UI','IN','RE',  
'!','?','.',',','@','#','$','%','*','^','-','+','=','/',':',';','&','~')
```

# Стартовая позиция (конец кодирования).

```
position = len(listWord) + 118
```

# Создание словаря с + позицией.

```
keysSyllables = {listSyllables[x]:Key[x + position] for x in range(len(listSyllables))}
```

# Создание специальных символов (для редактирования текста).

```
ListSpecial = ('<-','->','<+','+>')
```

# Стартовая позиция (конец шифра замены).

```
position = len(listWord) + len(listSyllables) + 118
```

# Создание словаря с + позицией.

```
keysSpecial = {listSpecial[x]:Key[x + position] for x in range(len(listSpecial))}
```

# Стартовая позиция (конец специальных символов).

```
position = len(listWord) + len(listSyllables) + len(listSpecial) + 118
```

# Создание символов-ловушек на все оставшиеся не занятые элементы памяти.

```
traps = tuple([Key[x] for x in range(position, Limit)])
```

# Удалить ненужные элементы.

```
del listWord, listSpecial, position
```

# Переключатель режимов шифрования.

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

# Сообщение для шифрования/расшифрования.

```
startMessage = input("Write the message: ").upper()
```

# Функция возвращающая триады чисел.

```
def regular(text):
```

```
    return findall(r"[0-9]{3}", text)
```

# Главная функция.

```
def encryptDecrypt(mode, message, final = "", string = ""):
```

# Если переключатель равен шифрованию, тогда разделить сообщение по словам.

```
if cryptMode == 'E':
```

```
    secondText = findall(r"[^\s]+", message)
```

# Удалить ненужные элементы.

```
del message
```

# Внедрение в код специальных символов.

```
for indexWord in range(len(secondText)):
```

```
    if secondText[indexWord] in keysSpecial:
```

```
        secondText[indexWord] = keysSpecial[secondText[indexWord]]
```

# Кодирование слов на числа.

```
for indexWord in range(len(secondText)):
```

```
    if secondText[indexWord] in keysCode:
```

```
        secondText[indexWord] = keysCode[secondText[indexWord]]
```

# Замена слогов на числа.

```
for indexWord in range(len(secondText)):
```

```
    for syllable in keysSyllables:
```

```
        if syllable in secondText[indexWord]:
```

```
            secondText[indexWord] =
```

```
secondText[indexWord].replace(syllable,keysSyllables[syllable])
```

# Разделение всех слов на символы.

```
for indexWord in range(len(secondText)):
```

```
    secondText[indexWord] = list(secondText[indexWord])
```

# Добавление в конец каждого списка пробел.

```
for indexWord in range(len(secondText)):
```

```
    secondText[indexWord].append(' ')
```

# Шифрование всех символов на числа при помощи омофоноф.

```
for indexWord in range(len(secondText)):
```

```
    for indexSymbol in range(len(secondText[indexWord])):
```

```
        symbol = secondText[indexWord][indexSymbol]
```

```
        if symbol in keysCrypt:
```

```
            length = len(keysCrypt[symbol])
```

```
secondText[indexWord][indexSymbol] = keysCrypt[symbol]  
[randint(0, length - 1)]
```

# Соединение всех чисел в одну строку.

```
for word in secondText:  
    string += "".join(word)
```

# Разделение чисел по триадам.

```
finalList = list(regular(string))
```

# Внедрение в сообщение ложных символов.

```
for indexList in range(len(finalList)):  
    randSwitch = randint(0,2); randPosition = randint(0,len(finalList))  
    if not randSwitch: finalList.insert(randPosition,choice(traps))
```

# Занесение в переменную final зашифрованный текст.

```
for word in finalList:  
    final += "".join(word)
```

# Вернуть зашифрованное сообщение.

```
return ".".join(regular(final))
```

# Если переключатель равен расшифрованию, тогда сделать перебор сообщения через функцию, возвращающая триады чисел.

```
else:  
    for symbolText in regular(message):
```

# Перебор всех специальных символов.

```
for element in keysSpecial:  
    if symbolText == keysSpecial[element]: final += element
```

# Перебор всех кодов.

```
for word in keysCode:  
    if symbolText == keysCode[word]: final += word
```

# Перебор всех символов и слогов шифра.

```
for syllable in keysSyllables:  
    if symbolText == keysSyllables[syllable]: final += syllable
```

# Перебор всех омофонов.

```
for symbol in keysCrypt:  
    if symbolText in keysCrypt[symbol]: final += symbol
```

# Разделить слова между собой в список.

```
listWord = findall(r"[\s]+", final)
```

# Перебор списка слов.

```
for _ in range(len(listWord)):
```

# Если специальный символ будет являться словом в списке, тогда сделать действия.

```
for element in keysSpecial:
```

```
    if element in listWord:
```

# Если найдено слово '<-', тогда удалить предыдущее слово и этот символ.

```
if element == '<-':
```

```
    del listWord[listWord.index(element) - 1]
```

```
    listWord.remove(element)
```

# Если найдено слово '->', тогда удалить последующее слово и этот символ.

```
elif element == '->':
```

```
    del listWord[listWord.index(element) + 1]
```

```
    listWord.remove(element)
```

# Если найдено слово '<+', тогда продублировать предыдущее слово заменив этот символ дублированным словом.

```
elif element == '<+':
```

```
    listWord[listWord.index(element)] = listWord[listWord.index(element) - 1]
```

# Если найдено слово '+>', тогда продублировать последующее слово заменив этот символ дублированным словом.

```
elif element == '+>':
```

```
    listWord[listWord.index(element)] = listWord[listWord.index(element) + 1]
```

```
else: pass
```

# Вернуть расшифрованное сообщение.

```
return " ".join(listWord)
```

# Вывод результата.

```
print("Final message:", encryptDecrypt(cryptMode, startMessage))
```

### 037. RSA шифрование (модуль).

(RSA является асимметричным методом шифрования. В данном примере я не расписывал скрипт с нуля, а установил готовый модуль rsa.)

```
// main.py
```

```
Start_[]
```

```
from rsa import newkeys
```

```
pub, priv = newkeys(1024)
```

```
print(str(pub) + "\n\n" + str(priv))
```

```
fileEncrypt = "encryptRSA.py"
```

```
fileDecrypt = "decryptRSA.py"
```

```
with open(fileEncrypt, "w") as encryptScript:
```

```
    encryptScript.write("""
```

```
from rsa import encrypt, PublicKey
```

```
message = input("Write the message: ").encode("utf8")
```

```
fileName = "encryptMessage.txt"
```

```
encryptMessage = encrypt(message, str(pub))
```

```
with open(fileName, "wb") as file:
```

```
    file.write(encryptMessage)
```

```
    print("Encrypt message:\n{message}\n[+] File: '{name}' successfully  
saved!".format(message = str(encryptMessage), name = fileName))
```

```
""")
```

```
    print("[+] File: '{name}' successfully saved!".format(name = fileEncrypt))
```

```
with open(fileDecrypt, "w") as decryptScript:
```

```
    decryptScript.write("""
```

```
from rsa import decrypt, PrivateKey
```

```
fileName = input("Write the filename: ")
```

```
with open(fileName, "rb") as file:
```

```
    decryptMessage = decrypt(file.read(), str(priv))
```

```
    print("Decrypted message:", str(decryptMessage.decode("utf8")))
```

```
""")
```

```
    print("[+] File: '{name}' successfully saved!".format(name = fileDecrypt))
```

```
]_End
```

**// terminal**

**Start\_1**

\$ python main.py

PublicKey(11771105794757592845646068472184849673904233810164636440170937  
94284982013380942638204590120306097937179449331728392551603116715566944  
80039811535960917598398562389338987778280064829139819522521772044574277  
41136679863841512439390464182934066451466047958923845070829576308634826  
6970724862588375182544212130104851, 65537)

PrivateKey(1177110579475759284564606847218484967390423381016463644017093  
79428498201338094263820459012030609793717944933172839255160311671556694  
48003981153596091759839856238933898777828006482913981952252177204457427  
74113667986384151243939046418293406645146604795892384507082957630863482  
66970724862588375182544212130104851, 65537,  
14422689401697281009282989735820123423631078245208360256736291206659452  
77850522511677809278126549343966153185800233789977116319941798792620484  
05235490911123269709217612610807517347146060130464443497456812718443107  
52679428764593835461429891497872537100737917243749225085096967756594546  
26541770110084822970073,  
46616629005215852188334700013363750449465185302750313331009524641445141  
13770394561856730062023434361862218206533265343459798363260211684442953  
5204470188733828475791,  
25250873016666530376601557713475906103023184986294875200452371794500410  
82721690833580449382223608585578004085530873315454398540590680971398777  
661)

[+] File: 'encryptRSA.py' successfully saved!

[+] File: 'decryptRSA.py' successfully saved!

\$ python encryptRSA.py

Write the message: helloworld

Encrypt message:

b'2gu\xcf\n\x18\x01\xa3\xba\xc3L\xc3lVzu,z7\xa0\xf7\x13\x92k2\x8fU<\xa8\xb6\x16\  
x9fS\xd8-\xca\x0cakJ\*\x81>\x0c\x9c\x86\xc2^\x18\xb9r\xce0\x82o\x0f\xdf;m\x15G\  
xcd\xcf\xda\xa54\xab\xe1T\x83\n\\LG\xb4\xd1\x08^Y\xdf\\x94\x10\xca\x81\x93<\xad\  
xf2H\xf1\xcdf0v\xbe\x9c(\xc6P\xac\xc9\x87\xfcOm\xbd\xa9\x83\x1f\xde=\x95<kY\  
x10\xa1\xd35\xb1D\xcd\xdf3.\x96\xc4'

[+] File: 'encryptMessage.txt' successfully saved!

\$ python decryptRSA.py

Write the filename: encryptMessage.txt

Decrypted message: helloworld

**]**\_End

# Импорт функции из модуля rsa.

**from** rsa **import** newkeys

# Создание публичного и приватного ключей.

pub, priv = newkeys(1024)

print(str(pub) + "\n\n" + str(priv))

# Имена файлов для шифрования/расшифрования.

fileEncrypt = "encryptRSA.py"

fileDecrypt = "decryptRSA.py"

# Создание файла для шифрования сообщений.

**with** open(fileEncrypt, "w") **as** encryptScript:  
 encryptScript.write("""

# Импорт функций из модуля rsa.

**from** rsa **import** encrypt, PublicKey

# Сообщение для шифрования.

message = input("Write the message: ").encode("utf8")

# Файл с зашифрованным сообщением.

fileName = "encryptMessage.txt"

# Шифрование сообщения.

encryptMessage = encrypt(message, ""+str(pub)+"")

# Создание файла с зашифрованным сообщением.

**with** open(fileName, "wb") **as** file:

file.write(encryptMessage)

print("Encrypt message:\n{message}\n[+] File: '{name}' successfully  
saved!".format(message = str(encryptMessage), name = fileName))  
 """)

print("[+] File: '{name}' successfully saved!".format(name = fileEncrypt))

# Создание файла для расшифрования сообщений.

**with** open(fileDecrypt, "w") **as** decryptScript:

decryptScript.write("""



```
# Импорт функций из модуля rsa.  
from rsa import decrypt, PrivateKey
```

```
# Имя файла с зашифрованным текстом.  
fileName = input("Write the filename: ")
```

```
# Открытие файла на чтение и его расшифровка.
```

```
with open(fileName, "rb") as file:  
    decryptMessage = decrypt(file.read(), str(priv))  
    print("Decrypted message:", str(decryptMessage.decode("utf8")))  
"  
    print("[+] File: '{name}' successfully saved!".format(name = fileDecrypt))
```

## 038. Аффинный шифр.

(Аффинный шифр основан на шифре Цезаря. Особенность этого метода шифрования заключена в модульной арифметике.)

// main.py

Start\_

```
print("Possible Key[a]: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.")
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
mainKey = input("Write the keys: ").split()
```

```
for key in mainKey:
```

```
    try: int(key)
```

```
    except:
```

```
        print("Error: key is not in numbers")
```

```
        raise SystemExit
```

```
if len(mainKey) != 2:
```

```
    print("Error: quality keys must be 2"); raise SystemExit
```

```
def encryptDecrypt(mode, message, key, final = ""):
```

```
    for symbol in message:
```

```
        if mode == 'E':
```

```
            final += chr((int(key[0]) * ord(symbol) + int(key[1]) - 13)%26 +
```

```
ord('A'))
```

```
        else:
```

```
            final += chr(pow(int(key[0]),11) * ((ord(symbol) + 26 - int(key[1]) -
```

```
13))%26 + ord('A'))
```

```
    return final
```

```
print("Final message:",encryptDecrypt(cryptMode, startMessage, mainKey))
```

End

// terminal

Start\_

```
$ python main.py
```

```
Possible Key[a]: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.
```

```
[E]ncrypt|[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Write the keys: 11 8
```

Final message: HAZZGQGNZP

\$ python main.py

Possible Key[a]: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.

[E]ncrypt|[D]ecrypt: d

Write the message: HAZZGQGNZP

Write the keys: 11 8

Final message: HELLOWORLD

**】\_End**

# Возможные комбинации первого ключа.

```
print("Possible Key[a]: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.")
```

# Переключатель режимов шифрования.

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

# Сообщение для шифрования/расшифрования.

```
startMessage = input("Write the message: ").upper()
```

# Ввод ключей.

```
mainKey = input("Write the keys: ").split()
```

Если один из ключей не является целым числом, тогда вывести ошибку.

```
for key in mainKey:
```

```
    try: int(key)
```

```
    except:
```

```
        print("Error: key is not int numbers")
```

```
        raise SystemExit
```

# Если ключей больше или меньше двух – вывести ошибку.

```
if len(mainKey) != 2:
```

```
    print("Error: quality keys must be 2"); raise SystemExit
```

# Главная функция.

```
def encryptDecrypt(mode, message, key, final = ""):
```

# Посимвольный перебор символов сообщения.

```
for symbol in message:
```

# Если переключатель равен 'E', тогда зашифровать сообщение при помощи двух ключей.

**if** mode == 'E':

**final** += **chr**((**int**(key[0]) \* **ord**(symbol) + **int**(key[1]) - 13)%26 + **ord**('A'))

# Если переключатель равен 'D', тогда расшифровать сообщение с двумя ключами.

**else**:

**final** += **chr**(**pow**(**int**(key[0]),11) \* ((**ord**(symbol) + 26 - **int**(key[1]) - 13)%26 + **ord**('A'))

# Вернуть сообщение.

**return** **final**

# Вывод сообщения.

**print**("Final message:",**encryptDecrypt**(cryptMode, startMessage, mainKey))

### 039. Шифр Хилла [2x2].

(Шифр Хилла основан на линейной алгебре и модульной арифметике. В данном примере описана матрица 2x2.)

// main.py

Start\_

```
from re import findall
```

```
matrixLength = 2
```

```
MatrixKey = [[15,4],[11,3]]
```

```
det = MatrixKey[0][0]*MatrixKey[1][1] - MatrixKey[0][1]*MatrixKey[1][0]
```

```
if det != 1:
```

```
    print("Error: determinant != 1")
```

```
    raise SystemExit
```

```
iMatrixKey = [
```

```
    [MatrixKey[1][1],-MatrixKey[0][1]],
```

```
    [-MatrixKey[1][0],MatrixKey[0][0]]
```

```
]
```

```
alpha = tuple("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ").upper()
```

```
for symbol in startMessage:
```

```
    if symbol not in [chr(x) for x in range(65,91)]:
```

```
        startMessage = startMessage.replace(symbol,"")
```

```
while len(startMessage) % matrixLength != 0: startMessage += 'Z'
```

```
def regular(text):
```

```
    return findall(r"[A-Z]{2}", text)
```

```
def encryptDecrypt(message, matrix, summ = 0, final = ""):
```

```
    for double in range(len(message)):
```

```
        for string in range(matrixLength):
```

```
            for column in range(matrixLength):
```

```
                summ += matrix[string][column] *
```

```
alpha.index(message[double][column])
```

```
        final += alpha[(summ)%26]; summ = 0
```

```
    return final
```

```
if cryptMode == 'E': finalMessage = encryptDecrypt(regular(startMessage),  
MatrixKey)
```

```
else: finalMessage = encryptDecrypt(regular(startMessage), iMatrixKey)
print("Final message:", finalMessage)
]_End
```

**// terminal**

**Start\_**

```
$ python main.py
```

```
[E]ncrypt[D]ecrypt: e
```

```
Write the message: helloworld
```

```
Final message: RLBYMMSXVA
```

```
$ python main.py
```

```
[E]ncrypt[D]ecrypt: d
```

```
Write the message: RLBYMMSXVA
```

```
Final message: HELLOWORLD
```

**]\_End**

```
# Импорт функции из модуля re.
```

```
from re import findall
```

```
# Создание матрицы 2x2.
```

```
matrixLength = 2
```

```
MatrixKey = [[15,4],[11,3]]
```

```
# Нахождение определителя матрицы и если он не равен единице, тогда вывести ошибку.
```

```
det = MatrixKey[0][0]*MatrixKey[1][1] - MatrixKey[0][1]*MatrixKey[1][0]
```

```
if det != 1:
```

```
    print("Error: determinant != 1")
```

```
    raise SystemExit
```

```
# Создание обратной матрицы.
```

```
iMatrixKey = [
```

```
    [MatrixKey[1][1],-MatrixKey[0][1]],
```

```
    [-MatrixKey[1][0],MatrixKey[0][0]]
```

```
]
```

```
# Создание кортежа с текстом.
```

```
alpha = tuple("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

```

# Переключатель режимов шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E', 'D']:
    print("Error: mode is not found")
    raise SystemExit

# Сообщение для шифрования/расшифрования.
startMessage = input("Write the message: ").upper()

# Перебор всех символов сообщения и если символ не найден в диапазоне A-Z,
тогда удалить этот символ.
for symbol in startMessage:
    if symbol not in [chr(x) for x in range(65,91)]:
        startMessage = startMessage.replace(symbol, "")

# Пока длина сообщения не будет делиться на длину матрицы без остатка –
добавлять символ 'Z'.
while len(startMessage) % matrixLength != 0: startMessage += 'Z'

# Функция возвращающая пары символов.
def regular(text):
    return findall(r"[A-Z]{2}", text)

# Главная функция.
def encryptDecrypt(message, matrix, summ = 0, final = ""):

# Шифрование/расшифрование пар символов по матрице и алфавиту.
for double in range(len(message)):
    for string in range(matrixLength):
        for column in range(matrixLength):
            summ += matrix[string][column] * alpha.index(message[double]
[column])
            final += alpha[(summ)%26]; summ = 0

# Вернуть сообщение.
return final

# Если переключатель равен шифрованию, тогда подставить обычную матрицу.
if cryptMode == 'E': finalMessage = encryptDecrypt(regular(startMessage),
MatrixKey)

```

# Если переключатель равен расшифрованию, тогда подставить обратную матрицу.  
**else: finalMessage = encryptDecrypt(regular(startMessage), iMatrixKey)**

# Вывод сообщения.

**print("Final message:", finalMessage)**



## 040. Шифр Хилла [3x3].

(Шифр Хилла основан на линейной алгебре и модульной арифметике. В данном примере описана матрица 3x3.)

// main.py

Start\_

```
from re import findall
```

```
alpha = tuple("ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890")
```

```
MatrixLength = 3; MatrixMod = len(alpha)
```

```
MatrixSquare = MatrixLength * MatrixLength
```

```
def checkErrors(key):
```

```
    if len(key) != MatrixSquare: return "Error: len(key) != %d"%MatrixSquare
```

```
    elif not getDeter(sliceto(key)): return "Error: det(Key) = 0"
```

```
    elif not getDeter(sliceto(key)) % MatrixMod: return "Error: det(Key) mod
```

```
len(alpha) = 0"
```

```
    else: return None
```

```
def regular(text):
```

```
    template = r".{%d}"%MatrixLength
```

```
    return findall(template, text)
```

```
def encode(matrix):
```

```
    for x in range(len(matrix)):
```

```
        for y in range(MatrixLength):
```

```
            matrix[x][y] = alpha.index(matrix[x][y])
```

```
    return matrix
```

```
def decode(matrixM, matrixK, message = ""):
```

```
    matrixF = []
```

```
    for z in range(len(matrixM)):
```

```
        temp = [0 for _ in range(MatrixLength)]
```

```
        for x in range(MatrixLength):
```

```
            for y in range(MatrixLength):
```

```
                temp[x] += matrixK[x][y] * matrixM[z][y]
```

```
            temp[x] = alpha[temp[x] % MatrixMod]
```

```
        matrixF.append(temp)
```

```
    for string in matrixF:
```

```
        message += "".join(string)
```

```
    return message
```

```
def sliceto(text):
```

```
    matrix = []
```

```
    for three in regular(text):
```

```

        matrix.append(list(three))
    return encode(matrix)
def getIDeter(det):
    for num in range(MatrixMod):
        if num * det % MatrixMod == 1:
            return num
def algebratic(x, y, det):
    matrix = sliceto(mainKey)
    matrix.remove(matrix[x])
    for z in range(2):
        matrix[z].remove(matrix[z][y])
    det2x2 = matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
    return (pow(-1, x + y) * det2x2 * getIDeter(det)) % MatrixMod
def getDeter(matrix):
    return \
    (matrix[0][0] * matrix[1][1] * matrix[2][2]) + \
    (matrix[0][1] * matrix[1][2] * matrix[2][0]) + \
    (matrix[1][0] * matrix[2][1] * matrix[0][2]) - \
    (matrix[0][2] * matrix[1][1] * matrix[2][0]) - \
    (matrix[0][1] * matrix[1][0] * matrix[2][2]) - \
    (matrix[1][2] * matrix[2][1] * matrix[0][0])
def getAlgr(det, index = 0):
    algrs = [0 for _ in range(MatrixSquare)]
    for string in range(MatrixLength):
        for column in range(MatrixLength):
            algrs[index] = algebratic(string, column, det)
            index += 1
    return algrs
def getIMatr(algr):
    return [
        [algr[0],algr[3],algr[6]],
        [algr[1],algr[4],algr[7]],
        [algr[2],algr[5],algr[8]]
    ]
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not Found")
    raise SystemExit
startMessage = input("Write the message: ").upper()

```

```

mainKey = input("Write the key: ").upper()
if checkErrors(mainKey):
    print(checkErrors(mainKey))
    raise SystemExit
for symbol in startMessage:
    if symbol not in alpha:
        startMessage = startMessage.replace(symbol,"")
while len(startMessage) % MatrixLength != 0:
    startMessage += startMessage[-1]
def encryptDecrypt(mode, message, key):
    MatrixMessage, MatrixKey = sliceto(message), sliceto(key)
    if mode == 'E':
        final = decode(MatrixMessage, MatrixKey)
    else:
        algbr = getAlgbr(getDeter(MatrixKey))
        final = decode(MatrixMessage, getIMatr(algbr))
    return final
print("Final message:", encryptDecrypt(cryptMode, startMessage, mainKey))
]_End

```

**// terminal**

**Start\_**

```

$ python main.py
[E]ncrypt[D]ecrypt: e
Write the message: helloworld
Write the key: qwertyuio
Final message: WP4RTLNLPP6P

```

```

$ python main.py
[E]ncrypt[D]ecrypt: d
Write the message: WP4RTLNLPP6P
Write the key: qwertyuio
Final message: HELLOWORLDDD

```

**]\_End**

# Импортирование функции findall из модуля регулярных выражений.

**from re import findall**

# Алфавит символов, которые будут использованы в шифре. Рекомендуется делать длину алфавита простым числом.

```
alpha = tuple("ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890")
```

# Длина матрицы и модуль матрицы по алфавиту.

```
MatrixLength = 3; MatrixMod = len(alpha)
```

# Квадрат длина матрицы.

```
MatrixSquare = MatrixLength * MatrixLength
```

# Функция на наличие ошибок.

```
def checkErrors(key):
```

# Если длина ключа не равна квадрату длины матрицы, тогда вывести ошибку.

```
if len(key) != MatrixSquare: return "Error: len(key) != %d" % MatrixSquare
```

# Если определитель матрицы равен нулю, тогда вывести ошибку.

```
elif not getDeter(sliceto(key)): return "Error: det(Key) = 0"
```

# Если квадрат длины матрицы нацело делится на определитель матрицы, тогда вывести ошибку.

```
elif not getDeter(sliceto(key)) % MatrixMod: return "Error: det(Key) mod len(alpha) = 0"
```

# Иначе ошибки нет.

```
else: return None
```

# Функция вычерпывающая из текста по несколько символов и возвращающая список.

```
def regular(text):  
    template = r".{%d}" % MatrixLength  
    return findall(template, text)
```

# Функция кодирования символов матрицы в числа.

```
def encode(matrix):  
    for x in range(len(matrix)):  
        for y in range(MatrixLength):  
            matrix[x][y] = alpha.index(matrix[x][y])  
    return matrix
```

```

# Функция шифрования / расшифрования сообщения и декодирования матрицы.
def decode(matrixM, matrixK, message = ""):
    matrixF = []

    # Перемножение матрицы-ключа с векторами-сообщениями.
    for z in range(len(matrixM)):
        temp = [0 for _ in range(MatrixLength)]
        for x in range(MatrixLength):
            for y in range(MatrixLength):
                temp[x] += matrixK[x][y] * matrixM[z][y]
            temp[x] = alpha[temp[x] % MatrixMod]
        matrixF.append(temp)

    # Создание строки из матрицы.
    for string in matrixF:
        message += " ".join(string)

    # Вернуть итоговое сообщение.
    return message

# Создание матрицы из строки.
def sliceto(text):
    matrix = []
    for three in regular(text):
        matrix.append(list(three))
    return encode(matrix)

# Получение обратного определителя матрицы по формуле  $n \cdot d \% m == 1$ .
def getIDeter(det):
    for num in range(MatrixMod):
        if num * det % MatrixMod == 1:
            return num

# Нахождение алгебраических дополнений матрицы.
def algebratic(x, y, det):

# Создание матрицы на основе ключа.
matrix = sliceto(mainKey)

# Удаление строки из матрицы.
matrix.remove(matrix[x])

```

# Удаление столбца матрицы.

```
for z in range(2):  
    matrix[z].remove(matrix[z][y])
```

# Получение определителя матрицы 2x2.

```
det2x2 = matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
```

# Вернуть определитель матрицы по формуле  $(-1^{x+y}) * d_{2x2} * d^{-1} \% m$

```
return (pow(-1, x + y) * det2x2 * getIDeter(det)) % MatrixMod
```

# Получение определителя матрицы.

```
def getDeter(matrix):  
    return \  
    (matrix[0][0] * matrix[1][1] * matrix[2][2]) + \  
    (matrix[0][1] * matrix[1][2] * matrix[2][0]) + \  
    (matrix[1][0] * matrix[2][1] * matrix[0][2]) - \  
    (matrix[0][2] * matrix[1][1] * matrix[2][0]) - \  
    (matrix[0][1] * matrix[1][0] * matrix[2][2]) - \  
    (matrix[1][2] * matrix[2][1] * matrix[0][0])
```

# Получение алгебраических дополнений.

```
def getAlgbr(det, index = 0):  
    algbrs = [0 for _ in range(MatrixSquare)]  
    for string in range(MatrixLength):  
        for column in range(MatrixLength):  
            algbrs[index] = algebraic(string, column, det)  
            index += 1  
    return algbrs
```

# Получение обратной матрицы.

```
def getIMatr(algbr):  
    return [  
        [algbr[0],algbr[3],algbr[6]],  
        [algbr[1],algbr[4],algbr[7]],  
        [algbr[2],algbr[5],algbr[8]]  
    ]
```

# Переключатель режимов шифрования.

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()  
if cryptMode not in ['E','D']:  
    print("Error: mode is not Found")  
    raise SystemExit
```

```

# Сообщение для шифрования / расшифрования.
startMessage = input("Write the message: ").upper()

# Ключ.
mainKey = input("Write the key: ").upper()

# Проверка на наличие ошибок.
if checkErrors(mainKey):
    print(checkErrors(mainKey))
    raise SystemExit

# Перебор всех символов в сообщении и если символ не был найден в алфавите,
тогда удалить его.
for symbol in startMessage:
    if symbol not in alpha:
        startMessage = startMessage.replace(symbol, '')

# Пока длина сообщения не делится нацело на длину матрицы, прибавлять к
сообщению последний символ этого же сообщения.
while len(startMessage) % MatrixLength != 0:
    startMessage += startMessage[-1]

# Основная функция шифрования / расшифрования.
def encryptDecrypt(mode, message, key):

# Преобразование сообщения и ключа в матрицы.
MatrixMessage, MatrixKey = sliceto(message), sliceto(key)

# Если переключатель равен 'E', тогда зашифровать сообщение при помощи
функции decode.
if mode == 'E':
    final = decode(MatrixMessage, MatrixKey)

# Иначе, если переключатель равен 'D', тогда расшифровать сообщение.
else:

# Нахождение алгебраических дополнений ключа-матрицы.
algbr = getAlgbr(getDeter(MatrixKey))

# Нахождение обратной матрицы и расшифрование сообщения.
final = decode(MatrixMessage, getIMatr(algbr))

```

# Вернуть сообщение.

**return final**

# Вывод сообщения.

**print("Final message:", encryptDecrypt(cryptMode, startMessage, mainKey))**



## 041. Протокол Диффи-Хеллмана.

(Протокол Диффи-Хеллмана позволяет получить общие ключи двум сторонам при незащищённом (прослушивающем) соединении. Третья же сторона не получает итоговый общий ключ.)

**// main.py**

**Start\_**[

p = 3001; g = 3

if g\*\*(p-1)%p != 1: raise SystemExit

a = 41;                    b = 12

A = g\*\*a%p;            B = g\*\*b%p

kA = B\*\*a%p;        kB = A\*\*b%p

print(kA, kB)

**End\_**

**// terminal**

**Start\_**[

\$ python main.py

2899 2899

**End\_**

# Генерация случайного простого числа p и генератора g.

p = 3001; g = 3

# Если 'g<sup>p-1</sup> mod p' не равно единице, тогда вывести ошибку.

if g\*\*(p-1)%p != 1: raise SystemExit

# Создание приватных ключей.

a = 41;                    b = 12

# Создание публичных ключей.

A = g\*\*a%p;        B = g\*\*b%p

# Создание общего ключа.

kA = B\*\*a%p;    kB = A\*\*b%p

# Вывод результата.

print(kA, kB)

## 042. RSA шифрование (механизм работы).

(RSA является асимметричным методом шифрования. В данном примере будет расписан механизм работы RSA шифрования.)

// main.py

Start\_

```
from random import choice
def getPrime(minN, maxN, flag = False, primes = []):
    for num in range(minN, maxN):
        for get in range(2, num):
            if num % get == 0:
                flag = True
                break
        if not flag:
            primes.append(num)
        flag = False
    return primes
def getPubExp(minN, maxN, Fn, pubExp = []):
    for e in range(minN, maxN):
        d = int((1 + 2 * Fn) / e)
        if d * e == 1 + 2 * Fn:
            pubExp.append(e)
    return pubExp
def getNumK(minN, maxN, Fn, e, numK = []):
    for k in range(minN, maxN):
        d = int((1 + k * Fn) / e)
        if d * e == 1 + k * Fn:
            numK.append(k)
    return numK
def getPrivExp(e, n, Fn, k):
    d = int((1 + k * Fn) / e)
    if d * e != 1 + k * Fn:
        raise SystemExit
    return d
def generateKeys(minP, maxP, maxN):
    primes = getPrime(minP, maxP)
    p, q = choice(primes), choice(primes)
    if p == q: return generateKeys(minP, maxP, maxN)
    n, Fn = p*q, (p-1)*(q-1)
```

```

try:
    pubExp = getPubExp(2, maxN, Fn)[0]
    numK = getNumK(2, maxN, Fn, pubExp)[0]
    privExp = getPrivExp(pubExp, n, Fn, numK)
except: return generateKeys(minP, maxP, maxN)
if pubExp > privExp: return generateKeys(minP, maxP, maxN)
return ([pubExp,n], [privExp,n])
def encryptDecrypt(message, key):
    return powerWithModule(message,key[0],key[1])
def powerWithModule(message, power, mod, result = 1):
    for _ in range(power):
        result = result * message % mod
    return result
while True:
    try:
        cryptMode = input("[G]enerate_[C]ipher: ").upper()
    except KeyboardInterrupt:
        print(); raise SystemExit
    if cryptMode == 'G':
        pub, priv = generateKeys(50,500,25)
        print("Public_key: [%d.%d]\nPrivate_key: [%d.%d]\n"%
              (pub[0], pub[1], priv[0], priv[1]))
    elif cryptMode == 'C':
        try:
            startMessage = int(input("Write the number: "))
            rsaKey = [int(k) for k in input("Write the key: ").split(".")]
        except (KeyboardInterrupt, ValueError):
            print(); raise SystemExit
        print("Final message: %d\n"%(encryptDecrypt(startMessage,
rsaKey)))
    else:
        print("Error: mode is not found")
        raise SystemExit

]_End

```

**// terminal**

**Start\_**

\$ python main.py

[G]enerate\_[C]ipher: g  
Public\_key: [3.69631]  
Private\_key: [46043.69631]

[G]enerate\_[C]ipher: c  
Write the number: 555  
Write the key: 3.69631  
Final message: 9770

[G]enerate\_[C]ipher: c  
Write the number: 9770  
Write the key: 46043.69631  
Final message: 555

[G]enerate\_[C]ipher: ^C  
**]-\_End**

# Импорт функции choice из модуля random.  
**from random import choice**

# Функция вычисления простых чисел.  
**def getPrime(minN, maxN, flag = False, primes = []):**

# Перебор чисел в дипозоне от minN до maxN.  
**for num in range(minN, maxN):**

# Перебор чисел в диапазоне от 2 до num.  
**for get in range(2,num):**

# Если num делится без остатка на get, тогда присвоить к flag значение True и прервать цикл вложенный цикл.  
**if num % get == 0:**  
    **flag = True**  
    **break**

# Если flag равняется значению False, тогда к списку primes добавить число num.  
**if not flag:**  
    **primes.append(num)**

# Присвоить к переменной flag дефолтное значение False.

**flag = False**

# Вернуть получившийся список.

**return primes**

# Функция подбора открытой экспоненты.

**def getPubExp(minN, maxN, Fn, pubExp = []):**

# Перебор чисел в дипозоне от minN до maxN.

**for e in range(minN, maxN):**

#  $d = (1 + 2 * f(n)) / e$ .

**d = int((1 + 2 \* Fn) / e)**

# Если  $d * e = 1 + 2 * f(n)$ , тогда добавить в список pubExp экспоненту e.

**if d \* e == 1 + 2 \* Fn:**

**pubExp.append(e)**

# Вернуть получившийся список.

**return pubExp**

# Функция подбора натурального числа k.

**def getNumK(minN, maxN, Fn, e, numK = []):**

# Перебор чисел в дипозоне от minN до maxN.

**for k in range(minN, maxN):**

#  $d = (1 + k * f(n)) / e$ .

**d = int((1 + k \* Fn) / e)**

# Если  $d * e = 1 + k * f(n)$ , тогда добавить в список numK экспоненту k.

**if d \* e == 1 + k \* Fn:**

**numK.append(k)**

# Вернуть получившийся список.

**return numK**

# Функция получения приватной экспоненты.

**def getPrivExp(e, n, Fn, k):**

```

#  $d = (1 + k * f(n)) / e$ .
d = int((1 + k * Fn) / e)

# Если  $d * e \neq 1 + k * f(n)$ , тогда вывести ошибку.
if d * e != 1 + k * Fn:
    raise SystemExit

# Вернуть секретную экспоненту d.
return d

# Функция генерации ключей.
def generateKeys(minP, maxP, maxN):

    # Получение простых чисел.
    primes = getPrime(minP, maxP)

    # Выбор двух случайных простых чисел.
    p, q = choice(primes), choice(primes)

    # Если  $p == q$ , тогда заного запустить функцию generateKeys.
    if p == q: return generateKeys(minP, maxP, maxN)

    # Получение произведения двух множителей и получение функции Эйлера.
    n, Fn = p*q, (p-1)*(q-1)

    # Проверка на наличие ошибок.
    try:

        # Получение открытой экспоненты.
        pubExp = getPubExp(2, maxN, Fn)[0]

        # Получение натурального числа k.
        numK = getNumK(2, maxN, Fn, pubExp)[0]

        # Получение секретной экспоненты.
        privExp = getPrivExp(pubExp, n, Fn, numK)

        # Если вывелась какая-либо ошибка, тогда заного запустить функцию
        generateKeys.
        except: return generateKeys(minP, maxP, maxN)

```

```

# Если публичная экспонента больше приватной, тогда заново генерировать ключи.
if pubExp > privExp: return generateKeys(minP, maxP, maxN)

# Вернуть кортеж двух списков. Первый список – публичный ключ, второй список
– приватный ключ.
return ([pubExp,n], [privExp,n])

# Функция шифрования / расшифрования.
def encryptDecrypt(message, key):

# Вызвать функцию powerWithModule и вернуть её результат.
return powerWithModule(message,key[0],key[1])

# Функция ускоряющая шифрование/расшифрование сообщения.
def powerWithModule(message, power, mod, result = 1):
    for _ in range(power):
        result = result * message % mod
    return result

# Бесконечный цикл и проверка на наличие ошибок.
while True:
    try:

# Ввод опции.
cryptMode = input("[G]enerate_[C]ipher: ").upper()

# Если пользователь попытался закрыть программу при помощи комбинации
клавиш подобия Ctrl+C, тогда не выводить визуальную ошибку и закрыть
программу.
except KeyboardInterrupt:
    print(); raise SystemExit

# Если выбор опции равен символу 'G', тогда сгенерировать публичный и
приватный ключи, а также вывести их на экран.
if cryptMode == 'G':
    pub, priv = generateKeys(50,500,25)
    print("Public_key: [%d.%d]\nPrivate_key: [%d.%d]\n"%
        (pub[0], pub[1], priv[0], priv[1]))

```

# Иначе, если выбор опции равен символу 'C', тогда проверить на наличие ошибок.

```
elif cryptMode == 'C':  
    try:
```

# Ввод числа-сообщения.

```
startMessage = int(input("Write the number: "))
```

# Ввод публичного/приватного ключа.

```
rsaKey = [int(k) for k in input("Write the key: ").split(".")]
```

# Если пользователь попытался закрыть программу при помощи комбинации клавиш подобия Ctrl+C или произошла ошибка ввода сообщения/ключа, тогда не выводить визуальную ошибку и закрыть программу.

```
except (KeyboardInterrupt, ValueError):  
    print(); raise SystemExit  
    print("Final message: %d\n"%(encryptDecrypt(startMessage, rsaKey)))
```

# Иначе, если опция не была найдена, тогда вывести ошибку.

```
else:  
    print("Error: mode is not found")  
    raise SystemExit
```



### 043. Цифровая подпись.

(Цифровая подпись позволяет сопоставить отправленное сообщение с автором. В данном случае взят принцип RSA. Также в данном случае использован пример без шифрования открытого сообщения и подписи.)

// main.py

Start\_1

```
from random import choice
def getPrime(minN, maxN, flag = False, primes = []):
    for num in range(minN, maxN):
        for get in range(2, num):
            if num % get == 0:
                flag = True
                break
        if not flag:
            primes.append(num)
        flag = False
    return primes
def getPubExp(minN, maxN, Fn, pubExp = []):
    for e in range(minN, maxN):
        d = int((1 + 2 * Fn) / e)
        if d * e == 1 + 2 * Fn:
            pubExp.append(e)
    return pubExp
def getNumK(minN, maxN, Fn, e, numK = []):
    for k in range(minN, maxN):
        d = int((1 + k * Fn) / e)
        if d * e == 1 + k * Fn:
            numK.append(k)
    return numK
def getPrivExp(e, n, Fn, k):
    d = int((1 + k * Fn) / e)
    if d * e != 1 + k * Fn:
        raise SystemExit
    return d
def generateKeys(minP, maxP, maxN):
    primes = getPrime(minP, maxP)
    p, q = choice(primes), choice(primes)
    if p == q: return generateKeys(minP, maxP, maxN)
```

```

n, Fn = p*q, (p-1)*(q-1)
try:
    pubExp = getPubExp(2, maxN, Fn)[0]
    numK = getNumK(2, maxN, Fn, pubExp)[0]
    privExp = getPrivExp(pubExp, n, Fn, numK)
except: return generateKeys(minP, maxP, maxN)
if pubExp > privExp: return generateKeys(minP, maxP, maxN)
return ([pubExp,n], [privExp,n])
def encryptDecrypt(message, key):
    return powerWithModule(message,key[0],key[1])
def powerWithModule(message, power, mod, result = 1):
    for _ in range(power):
        result = result * message % mod
    return result
def getKeys(name, pub, priv):
    return "%s's keys:
- Public_key: [%d.%d]
- Private_key: [%d.%d]\n"%\
        (name, pub[0], pub[1], priv[0], priv[1])
pubA, privA = generateKeys(125,250,50)
print(getKeys("Alice", pubA, privA))
mAlice = 111
sAlice = encryptDecrypt(mAlice, privA)
_mAlice = encryptDecrypt(sAlice, pubA)
print(" m = %s : _m = %s"%(mAlice, _mAlice))
if mAlice == _mAlice: print("Signature is True")
]_End

```

**// terminal**

**Start\_**

\$ python main.py

Alice's keys:

- Public\_key: [3.23449]

- Private\_key: [15427.23449]

m = 111 : \_m = 111

Signature is True

**]\_End**

# Импорт функции choice из модуля random.

```
from random import choice
```

# Функция вычисления простых чисел.

```
def getPrime(minN, maxN, flag = False, primes = []):
```

# Перебор чисел в дипозоне от minN до maxN.

```
for num in range(minN, maxN):
```

# Перебор чисел в диапазоне от 2 до num.

```
for get in range(2,num):
```

# Если num делится без остатка на get, тогда присвоить к flag значение True и прервать цикл вложенный цикл.

```
if num % get == 0:
```

```
    flag = True
```

```
    break
```

# Если flag равняется значению False, тогда к списку primes добавить число num.

```
if not flag:
```

```
    primes.append(num)
```

# Присвоить к переменной flag дефолтное значение False.

```
flag = False
```

# Вернуть получившийся список.

```
return primes
```

# Функция подбора открытой экспоненты.

```
def getPubExp(minN, maxN, Fn, pubExp = []):
```

# Перебор чисел в дипозоне от minN до maxN.

```
for e in range(minN, maxN):
```

#  $d = (1 + 2 * f(n)) / e$ .

```
d = int((1 + 2 * Fn) / e)
```

# Если  $d * e = 1 + 2 * f(n)$ , тогда добавить в список pubExp экспоненту e.

```
if d * e == 1 + 2 * Fn:
```

```
    pubExp.append(e)
```

# Вернуть получившийся список.

**return** pubExp

# Функция подбора натурального числа k.

**def** getNumK(minN, maxN, Fn, e, numK = []):

# Перебор чисел в дипозоне от minN до maxN.

**for** k **in** range(minN, maxN):

#  $d = (1 + k * f(n)) / e$ .

**d** = **int**((1 + k \* Fn) / e)

# Если  $d * e = 1 + k * f(n)$ , тогда добавить в список numK экспоненту k.

**if** d \* e == 1 + k \* Fn:

    numK.append(k)

# Вернуть получившийся список.

**return** numK

# Функция получения приватной экспоненты.

**def** getPrivExp(e, n, Fn, k):

#  $d = (1 + k * f(n)) / e$ .

**d** = **int**((1 + k \* Fn) / e)

# Если  $d * e \neq 1 + k * f(n)$ , тогда вывести ошибку.

**if** d \* e != 1 + k \* Fn:

**raise** SystemExit

# Вернуть секретную экспоненту d.

**return** d

# Функция генерации ключей.

**def** generateKeys(minP, maxP, maxN):

# Получение простых чисел.

primes = getPrime(minP,maxP)

# Выбор двух случайных простых чисел.

p, q = choice(primes), choice(primes)

```

# Если p == q, тогда заного запустить функцию generateKeys.
if p == q: return generateKeys(minP, maxP, maxN)

# Получение произведения двух множителей и получение функции Эйлера.
n, Fn = p*q, (p-1)*(q-1)

# Проверка на наличие ошибок.
try:

# Получение открытой экспоненты.
pubExp = getPubExp(2, maxN, Fn)[0]

# Получение натурального числа k.
numK = getNumK(2, maxN, Fn, pubExp)[0]

# Получение секретной экспоненты.
privExp = getPrivExp(pubExp, n, Fn, numK)

# Если вывелась какая-либо ошибка, тогда заного запустить функцию
generateKeys.
except: return generateKeys(minP, maxP, maxN)

# Если публичная экспонента больше приватной, тогда заного генерировать ключи.
if pubExp > privExp: return generateKeys(minP, maxP, maxN)

# Вернуть кортеж двух списков. Первый список – публичный ключ, второй список
– приватный ключ.
return ([pubExp,n], [privExp,n])

# Функция шифрования / расшифрования.
def encryptDecrypt(message, key):

# Вызвать функцию powerWithModule и вернуть её результат.
return powerWithModule(message,key[0],key[1])

# Функция ускоряющая шифрование/расшифрование сообщения.
def powerWithModule(message, power, mod, result = 1):
    for _ in range(power):
        result = result * message % mod
    return result

```

# Функция вывода пользователя и его ключей.

```
def getKeys(name, pub, priv):  
    return '''%s's keys:  
- Public_key: [%d.%d]  
- Private_key: [%d.%d]\n'''%\n        (name, pub[0], pub[1], priv[0], priv[1])
```

# Получение публичного и приватного ключей.

```
pubA, privA = generateKeys(125,250,50)
```

# Вывод полученных ключей.

```
print(getKeys("Alice", pubA, privA))
```

# Открытое число-сообщение.

```
MAlice = 111
```

# Создание цифровой подписи.

```
sAlice = encryptDecrypt(mAlice, privA)
```

# Создание прообраза сообщения.

```
_mAlice = encryptDecrypt(sAlice, pubA)
```

# Вывод открытого сообщения и его прообраза.

```
print(" m = %s : _m = %s"%(mAlice, _mAlice))
```

# Если открытое сообщение равно прообразу, тогда автор является подлинным.

```
if mAlice == _mAlice: print("Signature is True")
```

## 044. RSA шифрование (цифровая подпись).

(В данном случае представлено RSA шифрование с использованием цифровой подписи.)

// main.py

Start\_

```
from random import choice
def getPrime(minN, maxN, flag = False, primes = []):
    for num in range(minN, maxN):
        for get in range(2,num):
            if num % get == 0:
                flag = True
                break
        if not flag:
            primes.append(num)
        flag = False
    return primes
def getPubExp(minN, maxN, Fn, pubExp = []):
    for e in range(minN, maxN):
        d = int((1 + 2 * Fn) / e)
        if d * e == 1 + 2 * Fn:
            pubExp.append(e)
    return pubExp
def getNumK(minN, maxN, Fn, e, numK = []):
    for k in range(minN, maxN):
        d = int((1 + k * Fn) / e)
        if d * e == 1 + k * Fn:
            numK.append(k)
    return numK
def getPrivExp(e, n, Fn, k):
    d = int((1 + k * Fn) / e)
    if d * e != 1 + k * Fn:
        raise SystemExit
    return d
def generateKeys(minP, maxP, maxN):
    primes = getPrime(minP,maxP)
    p, q = choice(primes), choice(primes)
    if p == q: return generateKeys(minP, maxP, maxN)
    n, Fn = p*q, (p-1)*(q-1)
```

```

try:
    pubExp = getPubExp(2, maxN, Fn)[0]
    numK = getNumK(2, maxN, Fn, pubExp)[0]
    privExp = getPrivExp(pubExp, n, Fn, numK)
except: return generateKeys(minP, maxP, maxN)
if pubExp > privExp: return generateKeys(minP, maxP, maxN)
return ([pubExp,n], [privExp,n])
def encryptDecrypt(message, key):
    return powerWithModule(message,key[0],key[1])
def powerWithModule(message, power, mod, result = 1):
    for _ in range(power):
        result = result * message % mod
    return result
def getKeys(name, pub, priv):
    return "%s's keys:
- Public_key: [%d.%d]
- Private_key: [%d.%d]\n"%\
        (name, pub[0], pub[1], priv[0], priv[1])
pubA, privA = generateKeys(125,250,50)
pubB, privB = generateKeys(250,500,50)
print(getKeys("Alice", pubA, privA))
print(getKeys("Bob", pubB, privB))
mAlice = 111
sAlice = encryptDecrypt(mAlice, privA)
pubB = [pubB[0], pubB[1]]
CmAlice = encryptDecrypt(mAlice, pubB)
CsAlice = encryptDecrypt(sAlice, pubB)
CsmAlice = [CmAlice, CsAlice]
DmBob = encryptDecrypt(CmAlice, privB)
DsBob = encryptDecrypt(CsAlice, privB)
_mAlice = encryptDecrypt(DsBob, pubA)
print(" m = %s : _m = %s"%(DmBob, _mAlice))
if _mAlice == DmBob: print("Signature is True")
]_End

```

**// terminal**  
**Start\_**



```
$ python main.py
```

Alice's keys:

- Public\_key: [5.24511]

- Private\_key: [9677.24511]

Bob's keys:

- Public\_key: [5.133901]

- Private\_key: [53261.133901]

```
m = 111 : _m = 111
```

Signature is True

**End**

```
# Импорт функции choice из модуля random.
```

```
from random import choice
```

```
# Функция вычисления простых чисел.
```

```
def getPrime(minN, maxN, flag = False, primes = []):
```

```
# Перебор чисел в дипозоне от minN до maxN.
```

```
for num in range(minN, maxN):
```

```
# Перебор чисел в диапазоне от 2 до num.
```

```
for get in range(2,num):
```

```
# Если num делится без остатка на get, тогда присвоить к flag значение True и прервать цикл вложенный цикл.
```

```
if num % get == 0:
```

```
    flag = True
```

```
    break
```

```
# Если flag равняется значению False, тогда к списку primes добавить число num.
```

```
if not flag:
```

```
    primes.append(num)
```

```
# Присвоить к переменной flag дефолтное значение False.
```

```
flag = False
```

```
# Вернуть получившийся список.
```

```
return primes
```

# Функция подбора открытой экспоненты.

```
def getPubExp(minN, maxN, Fn, pubExp = []):
```

# Перебор чисел в дипозоне от minN до maxN.

```
for e in range(minN, maxN):
```

#  $d = (1 + 2 * f(n)) / e$ .

```
d = int((1 + 2 * Fn) / e)
```

# Если  $d * e = 1 + 2 * f(n)$ , тогда добавить в список pubExp экспоненту e.

```
if d * e == 1 + 2 * Fn:
    pubExp.append(e)
```

# Вернуть получившийся список.

```
return pubExp
```

# Функция подбора натурального числа k.

```
def getNumK(minN, maxN, Fn, e, numK = []):
```

# Перебор чисел в дипозоне от minN до maxN.

```
for k in range(minN, maxN):
```

#  $d = (1 + k * f(n)) / e$ .

```
d = int((1 + k * Fn) / e)
```

# Если  $d * e = 1 + k * f(n)$ , тогда добавить в список numK экспоненту k.

```
if d * e == 1 + k * Fn:
    numK.append(k)
```

# Вернуть получившийся список.

```
return numK
```

# Функция получения приватной экспоненты.

```
def getPrivExp(e, n, Fn, k):
```

#  $d = (1 + k * f(n)) / e$ .

```
d = int((1 + k * Fn) / e)
```

# Если  $d * e \neq 1 + k * f(n)$ , тогда вывести ошибку.

```
if d * e != 1 + k * Fn:
    raise SystemExit
```

# Вернуть секретную экспоненту d.

**return d**

# Функция генерации ключей.

**def generateKeys(minP, maxP, maxN):**

# Получение простых чисел.

**primes = getPrime(minP,maxP)**

# Выбор двух случайных простых чисел.

**p, q = choice(primes), choice(primes)**

# Если  $p == q$ , тогда заного запустить функцию generateKeys.

**if p == q: return generateKeys(minP, maxP, maxN)**

# Получение произведения двух множителей и получение функции Эйлера.

**n, Fn = p\*q, (p-1)\*(q-1)**

# Проверка на наличие ошибок.

**try:**

# Получение открытой экспоненты.

**pubExp = getPubExp(2, maxN, Fn)[0]**

# Получение натурального числа k.

**numK = getNumK(2, maxN, Fn, pubExp)[0]**

# Получение секретной экспоненты.

**privExp = getPrivExp(pubExp, n, Fn, numK)**

# Если вывелась какая-либо ошибка, тогда заного запустить функцию generateKeys.

**except: return generateKeys(minP, maxP, maxN)**

# Если открытая экспонента больше приватной, тогда заного генерировать ключи.

**if pubExp > privExp: return generateKeys(minP, maxP, maxN)**

# Вернуть кортеж двух списков. Первый список – публичный ключ, второй список – приватный ключ.

**return ([pubExp,n], [privExp,n])**

```

# Функция шифрования / расшифрования.
def encryptDecrypt(message, key):

# Вызвать функцию powerWithModule и вернуть её результат.
return powerWithModule(message, key[0], key[1])

# Функция ускоряющая шифрование/расшифрование сообщения.
def powerWithModule(message, power, mod, result = 1):
    for _ in range(power):
        result = result * message % mod
    return result

# Функция вывода пользователя и его ключей.
def getKeys(name, pub, priv):
    return '''%s's keys:
- Public_key: [%d.%d]
- Private_key: [%d.%d]\n''' % \
        (name, pub[0], pub[1], priv[0], priv[1])

# Алиса и Боб создают свои публичные и приватные ключи.
pubA, privA = generateKeys(125, 250, 50)
pubB, privB = generateKeys(250, 500, 50)

# Вывод публичных и приватных ключей Алисы и Боба.
print(getKeys("Alice", pubA, privA))
print(getKeys("Bob", pubB, privB))

# Открытое сообщение Алисы.
mAlice = 111

# Создание подписи со стороны Алисы.
sAlice = encryptDecrypt(mAlice, privA)

# Боб отправляет Алисе свой публичный ключ.
pubB = [pubB[0], pubB[1]]

# Алиса зашифровывает при помощи публичного ключа Боба своё сообщение и
цифровую подпись.
CmAlice = encryptDecrypt(mAlice, pubB)
CsAlice = encryptDecrypt(sAlice, pubB)

```

# Алиса отправляет зашифрованные сообщение и подпись Бобу.

**CsmAlice = [CmAlice, CsAlice]**

# Боб расшифровывает сообщение и подпись Алисы.

**DmBob = encryptDecrypt(CmAlice, privB)**

**DsBob = encryptDecrypt(CsAlice, privB)**

# Боб создаёт прообраз сообщения.

**\_mAlice = encryptDecrypt(DsBob, pubA)**

**print(" m = %s : \_m = %s"%(DmBob, \_mAlice))**

# Если прообраз равен расшифрованному сообщению, тогда автор является подлинным.

**if \_mAlice == DmBob: print("Signature is True")**

## 045. Шифр с лавинным эффектом.

(Лавинный эффект позволяет координально изменять зашифрованное сообщение даже при незначительном редактировании открытого сообщения.)

// main.py

Start\_

```
m = 32; n = 127; length = n - m
```

```
dictN = {x-m+1 : chr(x) for x in range(m,n)}
```

```
dictC = {chr(x) : x-m+1 for x in range(m,n)}
```

```
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
```

```
if cryptMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
startMessage = input("Write the message: ")
```

```
mainKey = int(input("Write the key: "))
```

```
for symbol in startMessage:
```

```
    if symbol not in [chr(x) for x in range(m,n)]:
```

```
        startMessage = startMessage.replace(symbol, " ")
```

```
def encrypt(message, key, result = ""):
```

```
    for index, symbol in enumerate(message):
```

```
        pos = len(message) - index
```

```
        result += dictN[(dictC[symbol]+(pos*key)+key)%length+1]
```

```
        key = sumAll(message[:len(message)-pos+1])%256
```

```
    return result
```

```
def decrypt(message, key, result = ""):
```

```
    for index, symbol in enumerate(message):
```

```
        pos = len(message) - index
```

```
        result += dictN[(dictC[symbol]-(pos*key)-key-2)%length+1]
```

```
        key = sumAll(result)%256
```

```
    return result
```

```
def sumAll(message, summ = 0):
```

```
    for symbol in message: summ += dictC[symbol]
```

```
    return summ
```

```
def toString(listS, final = ""):
```

```
    return "".join(listS)
```

```
def encryptDecrypt(mode, message, key):
```

```
    if mode == 'E':
```

```
        enc_message = list(encrypt(message, key))
```

```
        enc_message.reverse()
```

```

        return encrypt(toString(enc_message), key)
    else:
        dec_message = list(decrypt(message, key))
        dec_message.reverse()
        return decrypt(toString(dec_message), key)
print("Final message:", encryptDecrypt(cryptMode, startMessage, mainKey))
]_End

```

**// terminal**

**Start\_1**

```

$ python main.py
[E]ncrypt|[D]ecrypt: e
Write the message: Hello World!
Write the key: 123
Final message: %Ogy,}wo7"{h

```

```

$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the message: %Ogy,}wo7"{h
Write the key: 123
Final message: Hello World!

```

```

$ python main.py
[E]ncrypt|[D]ecrypt: e
Write the message: Hello world!
Write the key: 123
Final message: eX[L'r8I7]`{

```

```

$ python main.py
[E]ncrypt|[D]ecrypt: d
Write the message: eX[L'r8I7]`{
Write the key: 123
Final message: Hello world!

```

**]\_End**

# m – начало символов по ASCII таблице, n – конец символов по ASCII таблице,  
length – количество используемых символов в шифре.

**m = 32; n = 127; length = n – m**

```

# Словарь dictN вида – число:символ, словарь dictC вида – символ:число.
dictN = {x-m+1 : chr(x) for x in range(m,n)}
dictC = {chr(x) : x-m+1 for x in range(m,n)}

# Переключатель режимов шифрования.
cryptMode = input("[E]ncrypt|[D]ecrypt: ").upper()
if cryptMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit

# Ввод сообщения для шифрования/расшифрования.
startMessage = input("Write the message: ")

# Ввод ключа.
mainKey = int(input("Write the key: "))

# Перебор всех символов в сообщении и если символ не был найден в диапазоне
символов от m до n по ASCII таблице, тогда удалить этот символ.
for symbol in startMessage:
    if symbol not in [chr(x) for x in range(m,n)]:
        startMessage = startMessage.replace(symbol, '')

# Функция шифрования.
def encrypt(message, key, result = ""):

# Посимвольный перебор открытого сообщения.
for index, symbol in enumerate(message):

# Вычисление позиции.
pos = len(message) - index

# Шифрование символа.
result += dictN[(dictC[symbol]+(pos*key)+key)%length+1]

# Присвоение к переменной key нового значения в зависимости от суммы кодов.
key = sumAll(message[:len(message)-pos+1])%256

# Вернуть зашифрованное сообщение.
return result

# Функция расшифрования.
def decrypt(message, key, result = ""):

```



# Посимвольный перебор зашифрованного сообщения.

**for** index, symbol **in** enumerate(message):

# Вычисление позиции.

pos = len(message) - index

# Расшифрование символа.

result += dictN[(dictC[symbol]-(pos\*key)-key-2)%length+1]

# Присвоение к переменной key нового значения в зависимости от суммы кодов.

key = sumAll(result)%256

# Вернуть расшифрованное сообщение.

**return** result

# Вычисление суммы кодов.

**def** sumAll(message, summ = 0):

**for** symbol **in** message: summ += dictC[symbol]

**return** summ

# Преобразование списка в строку.

**def** toString(listS, final = ""):

**return** "".join(listS)

# Основная функция шифрования/расшифрования.

**def** encryptDecrypt(mode, message, key):

# Если переключатель равен 'E', тогда зашифровать сообщение.

**if** mode == 'E':

    enc\_message = list(encrypt(message, key))

# Реверснуть сообщение, чтобы лавинный эффект распространялся в обе стороны от изменения символов открытого сообщения, а не только в одну сторону.

enc\_message.reverse()

# Вернуть зашифрованное сообщения.

**return** encrypt(toString(enc\_message), key)

# Иначе, если переключатель равен 'D', тогда расшифровать сообщение.

**else:**

    dec\_message = list(decrypt(message, key))

# Реверснуть сообщение, чтобы лавинный эффект распространялся в обе стороны от изменения символов открытого сообщения, а не только в одну сторону.

**dec\_message.reverse()**

# Вернуть расшифрованное сообщения.

**return decrypt(toString(dec\_message), key)**

# Вывод сообщения.

**print("Final message:", encryptDecrypt(cryptMode, startMessage, mainKey))**

## 046. Шифр транспонированной матрицы.

(Данный метод шифрования основан на транспонированной матрице.

T – транспонирование, A – открытое сообщение, E – зашифрованное сообщение.

Определение:  $T(A) = E$ ,  $T(E) = A$ ,  $T(T(A)) = A$ )

// main.py

Start\_1

```
from random import randint
message = input("Write the message: ")
length = len(message)
if not length:
    print("Error: message is not found")
    raise SystemExit
try: matrixLength = int(input("Matrix length: "))
except ValueError:
    print("Error: only int number")
    raise SystemExit
matrixSquare = matrixLength ** 2
divF = length / matrixSquare
divI = length // matrixSquare
QUAN = divI if divF <= divI else divI + 1
def sliceMessage(message, index = 0, slices = []):
    for _ in range(QUAN):
        slices.append(message[index:index+matrixSquare])
        index += matrixSquare
    return slices
def appendSymbol(slices):
    while len(slices[-1]) % matrixSquare != 0:
        slices[-1] += chr(randint(65,90))
    return slices
def getMatrix(slices):
    for matrix in range(QUAN):
        text, slices[matrix], index = slices[matrix], [], 0
        for _ in range(matrixLength):
            slices[matrix].append(list(text[index:index+matrixLength]))
            index += matrixLength
    return slices
def getTransMatrix(matrixes, trans_matrix = []):
    for matrix in range(QUAN):
```

```

        temp_matrix = []
        for _ in range(matrixLength):
            temp_matrix.append([0 for _ in range(matrixLength)])
        for string in range(matrixLength):
            for column in range(matrixLength):
                temp_matrix[string][column] = matrixes[matrix][column]
[string]
        trans_matrix.append(temp_matrix)
    return trans_matrix
def matrixMessage(matrixes, final = ""):
    for matrix in range(QUAN):
        for string in range(matrixLength):
            for column in range(matrixLength):
                final += matrixes[matrix][string][column]
    return final
matrix = getMatrix(appendSymbol(sliceMessage(message)))
encrypt = matrixMessage(getTransMatrix(matrix))
print("Final message:", encrypt)
]_End

```

**// terminal**

**Start\_**

\$ python main.py

Write the message: HELLOWORLD

Matrix length: 3

Final message: HLOEORLWLDPTLEQXUN

\$ python main.py

Write the message: HLOEORLWLDPTLEQXUN

Matrix length: 3

Final message: HELLOWORLWLDXPEUTQN

**]\_End**

# Импортирование функции randint из модуля random.

**from random import randint**

# Сообщение для шифрования/расшифрования.

**message = input("Write the message: ")**

# Длина сообщения.

**length = len(message)**

# Если сообщения не найдено, тогда вывести ошибку.

**if not length:**

**print("Error: message is not found")**

**raise SystemExit**

# Ввод длины матрицы. Если введённый/введённые символы не числа типа Int, тогда вывести ошибку.

**try: matrixLength = int(input("Matrix length: "))**

**except ValueError:**

**print("Error: only int number")**

**raise SystemExit**

# Получение квадрата матрицы.

**matrixSquare = matrixLength \*\* 2**

# Получение переменной, где делится длина на квадрат матрицы с дробной частью и получение переменной, где делится длина на квадрат матрицы без дробной части.

**divF = length / matrixSquare**

**divI = length // matrixSquare**

# Вычисление количества матриц.

**QUAN = divI if divF <= divI else divI + 1**

# Функция для разделения сообщения по частям в зависимости от длины квадрата матрицы.

**def sliceMessage(message, index = 0, slices = []):**

**for \_ in range(QUAN):**

**slices.append(message[index:index+matrixSquare])**

**index += matrixSquare**

**return slices**

# Заполнение последнего элемента случайными символами, пока длина сообщения не будет кратна длине квадрата матрицы.

**def appendSymbol(slices):**

**while len(slices[-1]) % matrixSquare != 0:**

**slices[-1] += chr(randint(65,90))**

**return slices**

# Получение матриц из разделённого сообщения.

```
def getMatrix(slices):
```

# Перебор списков хранящих срезы сообщения.

```
for matrix in range(QUAN):
```

# Создание переменных, которые будут обнуляться или обновляться при следующей итерации цикла.

```
text, slices[matrix], index = slices[matrix], [], 0
```

# Создание ячеек матрицы в зависимости от указанной длины матрицы.

```
for _ in range(matrixLength):
```

```
    slices[matrix].append(list(text[index:index+matrixLength]))
```

```
    index += matrixLength
```

# Вернуть получившуюся матрицу.

```
return slices
```

# Получение транспонированной матрицы.

```
def getTransMatrix(matrixes, trans_matrix = []):
```

#Перебор списков хранящих матрицы сообщения.

```
for matrix in range(QUAN):
```

# Создание матрицы, которая обнуляется при каждой новой итерации цикла.

```
temp_matrix = []
```

# Заполнение temp\_matrix ячейками с нулями. Длина temp\_matrix зависит от указанной длины матрицы.

```
for _ in range(matrixLength):
```

```
    temp_matrix.append([0 for _ in range(matrixLength)])
```

# Создание транспонированной матрицы.

```
for string in range(matrixLength):
```

```
    for column in range(matrixLength):
```

```
        temp_matrix[string][column] = matrixes[matrix][column][string]
```

# Добавить в финальную транспонированную матрицу temp\_matrix.

```
trans_matrix.append(temp_matrix)
```

# Вернуть транспонированную матрицу.

```
return trans_matrix
```

# Преобразование матрицы в сообщение.

```
def matrixMessage(matrixes, final = ""):
```

# Перебор количества матриц.

```
for matrix in range(QUAN):
```

# Перебор количества строк в одной матрице.

```
for string in range(matrixLength):
```

# Перебор количества столбцов в одной матрице.

```
for column in range(matrixLength):
```

# К переменной final прибавить ячейку матрицы.

```
final += matrixes[matrix][string][column]
```

# Вернуть сообщение.

```
return final
```

# Создание матрицы из сообщения.

```
matrix = getMatrix(appendSymbol(sliceMessage(message)))
```

# Шифрование матрицы под средством транспонирования и получение сообщения из матрицы.

```
encrypt = matrixMessage(getTransMatrix(matrix))
```

# Вывод результата.

```
print("Final message:", encrypt)
```

# Криптоанализ

## 101. BruteForce шифра Цезаря.

(Всего существует 25 вариантов дешифровки шифра Цезаря.)

// main.py

Start\_

```
cryptMessage = input("Write the message: ").upper()
```

```
print("All possible variants of decrypt:")
```

```
for key in range(26):
```

```
    if key < 10: print("[ 0%d ]"%(key), end = ' ')
```

```
    else: print("[ %d ]"%(key), end = ' ')
```

```
    for symbol in cryptMessage:
```

```
        print(chr((ord(symbol)-key-13)%26+ord('A')), end = " ")
```

```
    print()
```

End

// terminal

Start\_

```
$ python main.py
```

```
Write the message: PMTTWEWZTL
```

```
All possible variants of decrypt:
```

```
[ 00 ] PMTTWEWZTL
```

```
[ 01 ] OLSSVDVYSK
```

```
[ 02 ] NKRRUCUXRJ
```

```
[ 03 ] MJQQTBTWQI
```

```
[ 04 ] LIPPSASVPH
```

```
[ 05 ] KHOORZRUOG
```

```
[ 06 ] JGNNQYQTNF
```

```
[ 07 ] IFMMPXPSME
```

```
[ 08 ] HELLOWORLD
```

```
[ 09 ] GDKKNVNQKC
```

```
[ 10 ] FCJJMUMPJB
```

```
[ 11 ] EBIILTLOIA
```

```
[ 12 ] DAHHKSKNHZ
```

```
[ 13 ] CZGGJRJMGY
```

```
[ 14 ] BYFFIQILFX
```

```
[ 15 ] AXEEHPHKEW
```



```
[ 16 ] ZWDDGOGJDV
[ 17 ] YVCCFNFIGU
[ 18 ] XUBBEMEBHT
[ 19 ] WTAADLDGAS
[ 20 ] VSZZCKCFZR
[ 21 ] URYYBJBEYQ
[ 22 ] TQXXAIADXP
[ 23 ] SPWWZHZCWO
[ 24 ] ROVVYGYBVN
[ 25 ] QNUUXFXAUM
]_End
```

# Сообщение для дешифровки.

```
cryptMessage = input("Write the message: ").upper()
```

# Вывод получившихся вариантов дешифровки.

```
print("All possible variants of decrypt:")
```

# Перебор всех ключей от 0 до 25 включительно.

```
for key in range(26):
```

# Если ключ меньше 10 – выводить индекс первый с нулём.

```
if key < 10: print("[ 0%d ]"%(key), end = ' ')
```

# Иначе выводить индекс ключа без нуля.

```
else: print("[ %d ]"%(key), end = ' ')
```

# Посимвольный перебор зашифрованного сообщения и подстановка ключа.

```
for symbol in cryptMessage:
```

```
    print(chr((ord(symbol)-key-13)%26+ord('A')), end = '')
```

```
print()
```

## 102. Частотный криптоанализ.

(Частотный криптоанализ помогает дешифровать сообщения, которые были зашифрованы при помощи простого шифра замены.)

// main.py

Start\_1

```
print("""
\t\t A = 8.17% \t N = 6.75%
\t\t B = 1.49% \t O = 7.51%
\t\t C = 2.78% \t P = 1.93%
\t\t D = 4.25% \t Q = 0.10%
\t\t E = 12.7% \t R = 5.99%
\t\t F = 2.23% \t S = 6.33%
\t\t G = 2.02% \t T = 9.06%
\t\t H = 6.09% \t U = 2.76%
\t\t I = 6.97% \t V = 0.98%
\t\t J = 0.15% \t W = 2.36%
\t\t K = 0.77% \t X = 0.15%
\t\t L = 4.03% \t Y = 1.97%
\t\t M = 2.41% \t Z = 0.05%
""")
name = input("File-name: ")
text = ""
try:
    with open(name, "r") as file:
        original = file.read()
except FileNotFoundError:
    print("File is not found!")
else:
    for symbol in original:
        if symbol != " ":
            text += symbol
dictionary = set(text)
def check(words, char):
    k = 0
    for symbol in words:
        if symbol == char: k += 1
    return k
var = 0
```

```

print("[*] Result: ")
for symbol in dictionary:
    stat = 100 * check(text,symbol) / len(text)
    if var%2 == 0:
        print("\t\t{0} - {1}%\t".format(symbol, round(stat,2)), end = "")
    else:
        print("{0} - {1}%".format(symbol, round(stat,2)))
    var += 1
print()
]_End

```

**// book.txt**

**Start\_**

```

:*.}(!{[?[]*%?(^.:)$:}>([%-?:.})(?&!@%[[%>/;(>([$:[*]/%.:]>?!+%!?$(?[*%;>!(/%[*
%>%:^\;/%?$!/==*(/+(&*[[:+!-%+%[:?&]%[(%$/;%!-(?&:^\.:]>^!+%@)]/(?#
%.:]>}>[*}($%!/[*%:#%!?(/[*%*]+@=%!/[*%;>:]$%/[/!(=$:[*@%!>+./!].@!>-(?^
%>(:>^!>[:*(/:?..]>@>: !$+!(?${[*} (= ^) == .!;;%!>.:]> /*! == :}%/[*% = ; } (==* := $+%];!
^=:![*](=/[*%];;?..]>/:]? $=%// $%%%; $:[*>($%>@%(?&}>!#-%$(!+!}>:[*=%//@:![*
%:^[!==@]($(?&!?$:^&:: $=.;>($%[*%?(^*%[*>({%!?$(@%#!/[!]}!.[*%}>:/[!]/[*(/
+.=:{%}!/+.$%#!.

```

**]\_End**

**// terminal**

**Start\_**

\$ python main.py

```

A = 8.17%  N = 6.75%
B = 1.49%  O = 7.51%
C = 2.78%  P = 1.93%
D = 4.25%  Q = 0.10%
E = 12.7%  R = 5.99%
F = 2.23%  S = 6.33%
G = 2.02%  T = 9.06%
H = 6.09%  U = 2.76%
I = 6.97%  V = 0.98%
J = 0.15%  W = 2.36%
K = 0.77%  X = 0.15%
L = 4.03%  Y = 1.97%
M = 2.41%  Z = 0.05%

```

File-name: book.txt

[\*] Result:

.	- 3.28%	]	- 3.93%
^	- 2.62%	!	- 8.08%
+	- 2.62%	>	- 5.9%
\$	- 4.8%	-	- 1.09%
#	- 1.31%	}	- 3.49%
(	- 7.42%	@	- 2.18%
:	- 8.95%	[	- 7.42%
/	- 6.11%	?	- 4.8%
{	- 0.44%	;	- 2.62%
&	- 1.53%	%	- 10.48%
=	- 4.8%	*	- 6.11%

**]**\_End

# Вывод частоты встречаемости символов английского алфавита.

```
print("""
\t\t A = 8.17% \t N = 6.75%
\t\t B = 1.49% \t O = 7.51%
\t\t C = 2.78% \t P = 1.93%
\t\t D = 4.25% \t Q = 0.10%
\t\t E = 12.7% \t R = 5.99%
\t\t F = 2.23% \t S = 6.33%
\t\t G = 2.02% \t T = 9.06%
\t\t H = 6.09% \t U = 2.76%
\t\t I = 6.97% \t V = 0.98%
\t\t J = 0.15% \t W = 2.36%
\t\t K = 0.77% \t X = 0.15%
\t\t L = 4.03% \t Y = 1.97%
\t\t M = 2.41% \t Z = 0.05%
""")
```

# Имя файла для чтения и создание переменной text.

```
name = input("File-name: "); text = ""
```

# Попытка открыть файл, если файл не найден – вывести ошибку. Иначе переписать все символы в переменную text за исключением пробелов.

**try:**

```
    with open(name,"r") as file:
        original = file.read()
```

**except FileNotFoundError:**

```
        print("File is not found!")
else:
    for symbol in original:
        if symbol != " ":
            text += symbol
```

```
# Создание множества в котором нет повторяющихся символов.
dictionary = set(text)
```

```
# Функция для подсчёта количества символов.
```

```
def check(words, char):
    k = 0
    for symbol in words:
        if symbol == char: k += 1
    return k
```

```
# Создание переменной var.
```

```
print("[*] Result: "); var = 0
```

```
# Посимвольный перебор множества dictionary.
```

```
for symbol in dictionary:
```

```
# Вычисление статистика встречаемости символа в тексте.
```

```
stat = 100 * check(text,symbol) / len(text)
```

```
# Если переменная var делится на два без остатка, тогда выводить сообщение без символа новой строки.
```

```
if var%2 == 0:
```

```
    print("\t\t{0} – {1}%\t".format(symbol, round(stat,2)), end = "")
```

```
# Если же переменная var не делится на два без остатка, тогда выводить сообщение с символом новой строки.
```

```
else:
```

```
    print("{0} – {1}% ".format(symbol, round(stat,2)))
```

```
var += 1
```

```
print()
```

## 103. BruteForce криптографической хеш-функции по словарю.

(Зашифрованные сообщения при помощи криптографических хеш-функций в основном являются паролями. Словарь, который я использую в этом скрипте:

<http://scrapmaker.com/download/data/wordlists/dictionaries/rockyou.txt> )

// main.py

Start\_

```
#!/usr/bin/python3
```

```
from sys import argv
```

```
from hashlib import sha256, sha512, md5
```

```
# chmod +x main.py
```

```
# ./main.py sha256 hash.txt rockyou.txt
```

```
line = "-----"
```

```
try:
```

```
    hashAlgr, fileHash, fileDict = argv[1], argv[2], argv[3]
```

```
except IndexError:
```

```
    print("Error: not enough arguments")
```

```
    raise SystemExit
```

```
with open(fileHash) as file:
```

```
    hashFunc = file.read()
```

```
    hashFunc = hashFunc.replace('\n', '')
```

```
def generator(string):
```

```
    for word in string:
```

```
        passwd = word.replace('\n', '')
```

```
        if encrypt(passwd) == hashFunc:
```

```
            yield line + "\n[True]: " + passwd
```

```
            return
```

```
        else:
```

```
            yield "[False]: " + passwd
```

```
def encrypt(string):
```

```
    passwd = string.encode()
```

```
    if hashAlgr == "md5":
```

```
        signature = md5(passwd).hexdigest()
```

```
    elif hashAlgr == "sha256":
```

```
        signature = sha256(passwd).hexdigest()
```

```
    elif hashAlgr == "sha512":
```

```
        signature = sha512(passwd).hexdigest()
```

```
    else: print("Error: not found algorithm."); raise SystemExit
```

```
        return signature
print(line)
with open(fileDict, errors = "ignore") as dictionary:
    for password in generator(dictionary):
        print(password)
print(line)
]_End
```

**// hash.txt**

**Start\_**

481f6cc0511143ccdd7e2d1b1b94faf0a700a8b49cd13922a70b5ae28acaa8c5

**]\_End**

**// terminal**

**Start\_**

\$ chmod +x main.py

\$ ./main.py sha256 hash.txt rockyou.txt

```
-----
[False]: 123456
[False]: 12345
[False]: 123456789
[False]: password
[False]: iloveyou
[False]: princess
[False]: 1234567
[False]: rockyou
[False]: 12345678
[False]: abc123
[False]: nicole
[False]: daniel
[False]: babygirl
[False]: monkey
[False]: lovely
[False]: jessica
```

```
-----
[True]: 654321
-----
```

**]\_End**

```
# Возможность использовать chmod +x.  
#!/usr/bin/python3
```

```
# Импортирование функции из sys и функций из hashlib.
```

```
from sys import argv  
from hashlib import sha256, sha512, md5
```

```
# Примеры использования программы.
```

```
# chmod +x main.py
```

```
# ./main.py sha256 hash.txt rockyou.txt
```

```
# Линия-разделитель.
```

```
line = "-----"
```

```
# Аргументы через терминал, если же каких-то аргументов не хватает – вывести ошибку.
```

```
try:
```

```
    hashAlgr, fileHash, fileDict = argv[1], argv[2], argv[3]
```

```
except IndexError:
```

```
    print("Error: not enough arguments")
```

```
    raise SystemExit
```

```
# Чтение файла с хешем и удаление символа новой строки.
```

```
with open(fileHash) as file:
```

```
    hashFunc = file.read()
```

```
    hashFunc = hashFunc.replace('\n', '')
```

```
# Создание функции-генератора.
```

```
def generator(string):
```

```
# Перебор всех слов в словаре.
```

```
for word in string:
```

```
# Удаление в слове символа новой строки.
```

```
    passwd = word.replace('\n', '')
```

```
# Сравнение слова с криптографической хеш-функцией. Если криптографическая хеш-функция равна слову, тогда вывести True и закрыть программу.
```

```
if encrypt(passwd) == hashFunc:
```

```
    yield line + "\n[True]: " + passwd
```

```
    return
```



# Иначе вывести False и продолжать поиск слова.

**else:**

**yield "[False]: "+passwd**

# Функция перевода слова в криптографическую хеш-функцию для последующего сравнения.

**def encrypt(string):**

# Перевод слова в кодировку.

**passwd = string.encode()**

# Если алгоритм равен md5, то использовать функцию md5.

**if hashAlgr == "md5":**

**signature = md5(passwd).hexdigest()**

# Если алгоритм равен sha256, то использовать функцию sha256.

**elif hashAlgr == "sha256":**

**signature = sha256(passwd).hexdigest()**

# Если алгоритм равен sha512, то использовать функцию sha512.

**elif hashAlgr == "sha512":**

**signature = sha512(passwd).hexdigest()**

# Если алгоритм не был найден – вывести ошибку.

**else: print("Error: not found algorithm."); raise SystemExit**

# Вернуть криптографическую хеш-функцию.

**return signature**

# Открыть словарь на чтение и с игнорированием ошибок.

**print(line)**

**with open(fileDict, errors = "ignore") as dictionary:**

# Перебор каждого слова в словаре через генератор.

**for password in generator(dictionary):**

**print(password)**

**print(line)**

## 104. BruteForce запароленного zip-архива по словарю.

(Поддерживаются только архивы формата zip. Словарь, который я использую в этом скрипте: <http://scrapmaker.com/download/data/wordlists/dictionaries/rockyou.txt> )

```
// main.py
```

```
Start_["
```

```
#!/usr/bin/python3
```

```
from zipfile import ZipFile
```

```
from sys import argv
```

```
from os import mkdir
```

```
# chmod +x main.py
```

```
# ./main.py archive.zip rockyou.txt
```

```
try:
```

```
    archiveFile, dictionaryFile = argv[1], argv[2]
```

```
except IndexError:
```

```
    print("Error: not enough arguments")
```

```
    raise SystemExit
```

```
line = "-----"
```

```
def generator(string):
```

```
    for word in string:
```

```
        passwd = word.replace('\n', '')
```

```
        archive.setpassword(passwd.encode())
```

```
        try:
```

```
            archive.extractall(directory)
```

```
        except:
```

```
            yield "[False]: " + passwd
```

```
        else:
```

```
            yield line + "\n[True]: " + passwd; return
```

```
directory = "ExtractArchive"
```

```
try: mkdir(directory)
```

```
except FileExistsError: pass
```

```
print(line)
```

```
with open(dictionaryFile, errors = 'ignore') as dictionary:
```

```
    with ZipFile(archiveFile) as archive:
```

```
        print(line)
```

```
        for password in generator(dictionary):
```

```
            print(password)
```

```
print(line)
```

```
]_End
```

**// terminal**

**Start\_**

\$ chmod +x main.py

\$ ./main.py archive.zip rockyou.txt

-----  
[False]: 123456  
[False]: 12345  
[False]: 123456789  
[False]: password  
[False]: iloveyou  
[False]: princess  
[False]: 1234567  
[False]: rockyou  
[False]: 12345678  
[False]: abc123  
[False]: nicole  
[False]: daniel  
[False]: babygirl  
[False]: monkey  
[False]: lovely  
[False]: jessica  
[False]: 654321  
[False]: michael  
[False]: ashley

-----  
[True]: qwerty

**End**

# Возможность использовать chmod +x.

**#!/usr/bin/python3**

# Импортирование функций из модулей zipfile, sys, os.

**from zipfile import ZipFile**

**from sys import argv**

**from os import mkdir**

# Примеры использования программы.

# **chmod +x main.py**

# **./main.py archive.zip rockyou.txt**

# Ввод аргументов с терминала.

```
try:  
    archiveFile, dictionaryFile = argv[1], argv[2]
```

# Если же не поступило какого-то аргумента, тогда вывести ошибку.

```
except IndexError:  
    print("Error: not enough arguments")  
    raise SystemExit
```

# Линия-разделитель.

```
line = "-----"
```

# Создание функции-генератор.

```
def generator(string):
```

# Перебор слов в словаре.

```
for word in string:
```

# Удаление символа новой строки в слове.

```
passwd = word.replace('\n', '')
```

# Установка пароля под архив, который пытаемся открыть.

```
archive.setpassword(passwd.encode())
```

# Попытка разархивировать архив.

```
try:  
    archive.extractall(directory)
```

Если пароль не верный – вывести False и продолжить искать пароль.

```
except:  
    yield "[False]: " + passwd
```

Если же пароль верный – вывести True и закрыть программу.

```
else:  
    yield line + "\n[True]: " + passwd; return
```

# Директория, в которой будут помещены разархивированные файлы. Если директория уже создана – ничего не делать.

```
directory = "ExtractArchive"
```

```
try: mkdir(directory)
```

```
except FileExistsError: pass
```

# Открытие словаря на уровне чтения с игнорирование ошибок.

```
print(line)
```

```
with open(dictionaryFile, errors = 'ignore') as dictionary:
```

# Открытие zip-архива на чтение.

```
with ZipFile(archiveFile) as archive:
```

```
    print(line)
```

# Перебор каждого слова в словаре через генератор.

```
for password in generator(dictionary):
```

```
    print(password)
```

```
print(line)
```

# Стеганография

## 201. Сообщение в файле на уровне байтов.

(Данный скрипт позволяет заносить сообщение в конец файла любого формата на уровне байтов. Осторожно: Файл перезаписывается!)

**// main.py**

**Start\_**

```
#!/usr/bin/python3
```

```
from sys import argv
```

```
# chmod +x main.py
```

```
# ./main.py file.jpg
```

```
try:
```

```
    nameFile = argv[1]
```

```
    textFile = argv[2:]
```

```
except IndexError:
```

```
    print("Error: not enough arguments")
```

```
    raise SystemExit
```

```
with open(nameFile, 'ab') as file:
```

```
    file.write("".join(textFile).encode("utf-8"))
```

```
    print("[+] File successfully overwritten!")
```

**End**

**// terminal**

**Start\_**

```
$ chmod +x main.py
```

```
$ ./main.py picture.jpeg helloworld!
```

```
[+] File: picture.jpeg successfully overwritten!
```

**End**

```
# Возможность использовать chmod +x.
```

```
#!/usr/bin/python3
```

```
# Импорт аргументов из модуля sys.
```

```
from sys import argv
```

# Пример использования программы.

# **chmod +x main.py**

# **./main.py file.jpg**

# Попытка указать аргументы в виде переменных.

**try:**

**nameFile = argv[1]**

**textFile = argv[2:]**

# Если произошла ошибка с аргументами, тогда закрыть программу.

**except IndexError:**

**print("Error: Arguments!"); raise SystemExit**

# Открыть файл на уровне байт и добавить в файл введённый текст.

**with open(nameFile, 'ab') as file:**

**file.write("".join(textFile).encode("utf-8"))**

**print("[+] File successfully overwritten!")**

## 202. Чтение байтов в файле.

(Побайтовое чтение файла на поиск скрытых сообщений.)

// **main.py**

**Start\_**[

```
#!/usr/bin/python3
```

```
from sys import argv
```

```
# chmod +x main.py
```

```
# ./main.py file.jpg
```

```
try:
```

```
    nameFile = argv[1]
```

```
except IndexError:
```

```
    print("Error: not enough arguments")
```

```
    raise SystemExit
```

```
try:
```

```
    with open(nameFile, "rb") as file:
```

```
        byte = file.read(1)
```

```
        counter = 0
```

```
        while byte:
```

```
            print(byte)
```

```
            byte = file.read(1)
```

```
            counter += 1
```

```
except FileNotFoundError:
```

```
    print("[x] File: '{name}' is not defined!".format(name = nameFile))
```

```
else:
```

```
    print("Number of bytes in the '{name}': {number}".format(name = nameFile,
```

```
number = counter))
```

**End\_**

// **terminal**

**Start\_**[

```
$ chmod +x main.py
```

```
$ ./main.py picture.jpeg
```

```
...
```

```
...
```

```
...
```

```
b'K'
```

```
b'\xfa'
```

```
b'c'
```



b'\x18'  
b'\xae'  
b'R'  
b'\xbe'  
b'\\'  
b'p'  
b'\x07'  
b'\x1e'  
b'!'  
b'q'  
b'\xae'  
b'\x95'  
b'5'  
b'\x02'  
b'\xbf'  
b'\xff'  
b'\xd9'  
b'h'  
b'e'  
b'l'  
b'l'  
b'o'  
b'w'  
b'o'  
b'r'  
b'l'  
b'd'  
b'!'

Number of bytes in the 'picture.jpeg': 1620896

**l\_End**

```
# Возможность использовать chmod +x.  
#!/usr/bin/python3
```

```
# Импорт аргументов из модуля sys.  
from sys import argv
```

```
# Пример использования программы.  
# chmod +x main.py  
# ./main.py file.jpg
```

# Попытка внести аргументы в переменную.

**try:**

**nameFile = argv[1]**

# Если вышла ошибка с аргументами, тогда закрыть программу.

**except IndexError:**

**print("Error: arguments"); raise SystemExit**

# Попробовать открыть файл и выводить побайтово всё содержимое файла.

**try:**

**with open(nameFile, "rb") as file:**

**byte = file.read(1)**

**counter = 0**

**while byte:**

**print(byte)**

**byte = file.read(1)**

**counter += 1**

# Если файл не был найден, тогда написать об этом.

**except FileNotFoundError:**

**print("[x] File: '{name}' is not defined!".format(name = nameFile))**

# Иначе, если всё прошло успешно – написать количество байт в файле.

**else:**

**print("Number of bytes in the '{name}': {number}".format(name = nameFile, number = counter))**

## 203. Занесение архива в файл.

(При помощи данного скрипта можно заносить архив в любой файл любого формата. В итоге полученный файл можно будет использовать как архив с функцией разархивации. Сам файл не теряет своих прежних функций.

Осторожно: Файл перезаписывается!)

**// main.py**

**Start\_**[

```
#!/usr/bin/python3
```

```
from sys import argv
```

```
# chmod +x main.py
```

```
# ./main.py file.jpg
```

```
try:
```

```
    nameFile, archiveFile = argv[1], argv[2]
```

```
except IndexError:
```

```
    print("Error: not enough arguments")
```

```
    raise SystemExit
```

```
try:
```

```
    with open(nameFile, "rb") as file:
```

```
        readFile = file.read()
```

```
    with open(archiveFile, "rb") as archive:
```

```
        readArchive = archive.read()
```

```
except FileNotFoundError:
```

```
    print("Error: file is not found")
```

```
    raise SystemExit
```

```
with open(nameFile, "wb") as file:
```

```
    file.write(readFile)
```

```
    file.write(readArchive)
```

```
    print("[+] File: successfully overwritten!")
```

**]\_End**

**// terminal**

**Start\_**[

```
$ chmod +x main.py
```

```
$ ./main.py picture.jpeg archive.zip
```

```
[+] File: successfully overwritten!
```

```
$ unzip picture.jpeg
```

**]\_End**

```
# Возможность использовать chmod +x.  
#!/usr/bin/python3
```

```
# Импорт функции из модуля sys.  
from sys import argv
```

```
# Пример использования программы.  
# chmod +x main.py  
# ./main.py file.jpg
```

```
# Указание аргументов в терминале.  
try:
```

```
    nameFile, archiveFile = argv[1], argv[2]
```

```
# Если не хватает какого-то аргумента, тогда вывести ошибку.
```

```
except IndexError:  
    print("Error: not enough arguments")  
    raise SystemExit
```

```
# Попытка открыть файл и архив.
```

```
try:  
    with open(nameFile, "rb") as file:  
        readFile = file.read()  
    with open(archiveFile, "rb") as archive:  
        readArchive = archive.read()
```

```
# Если же какой-то файл не был найден – вывести ошибку.
```

```
except FileNotFoundError:  
    print("Error: file is not found")  
    raise SystemExit
```

```
# Перезаписать файл и внести в него сначала себя, потом архив.
```

```
with open(nameFile, "wb") as file:  
    file.write(readFile)  
    file.write(readArchive)  
    print("[+] File: successfully overwritten!")
```

# Шифры не описанные в виде программ

## 301. Масонский шифр.

(Является простым шифром подстановки. Особенность шифра заключена в визуализации шифрования символов.)

// **Encrypt**

**Start\_**[

*Создание ключей:*

- 1) В основном в этом шифре криптограф сначала чертит:
  - две решётки в виде игры “крестики-нолики”;
  - два Икса;
- 2) Вторая решётка отличается от первой добавлением точек в каждую ячейку. То же самое делается и с Иксами.
- 3) В каждую ячейку полей криптограф заносит по одному символу (не обязательно в порядке алфавита).
- 4) В итоге мы получаем следующие ключи:

<b>A</b>	<b>B</b>	<b>C</b>
<b>D</b>	<b>E</b>	<b>F</b>
<b>G</b>	<b>H</b>	<b>I</b>

<b>J</b>	<b>K</b>	<b>L</b>
<b>M</b>	<b>N</b>	<b>O</b>
<b>P</b>	<b>Q</b>	<b>R</b>

<b>S</b>
<b>T</b> <b>U</b>
<b>V</b>

<b>W</b>
<b>X</b> <b>Y</b>
<b>Z</b>

Шифрование:

- 1) Каждый полученный символ ключа является формой решётки/Икса.
- 2) Допустим мы зашифруем сообщение “WORLD”:

WORLD

**End\_**

## 302. Пляшущие человечки.





































(Данный шифр был представлен в рассказе «Пляшущие человечки», вошедший в сборник из 13 рассказов «Возвращение Шерлока Холмса». «Пляшущие человечки» являются обычным шифром замены.)

// **Encrypt**

**Start\_**

Создание ключей:

- 1) Пишется какой-либо алфавит, а также, если существует необходимость, то и последовательность чисел.
- 2) К каждому символу присваивается отдельный рисунок человечка, который должен отличаться от других, то-есть не должно быть одинаковых человечков в разных символах.

												
A	B	C	D	E	F	G	H	I	J	K	L	M
												
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
												
1	2	3	4	5	6	7	8	9	0			

Шифрование:

- 1) Каждый человечек равен определённому символу.
- 2) Допустим мы зашифруем сообщение “WORLD”:



**\_End**

# Кодирование

## 401. Азбука Морзе.

(Азбука Морзе предназначена для передачи информации в основном по телеграфу. Цель была в передаче какого-либо сообщения при помощи сигналов по каналам электросвязи, где тире является длинным сигналом, а точка коротким.)

// image.png

Start\_

A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —		
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —

End\_

```

// main.py
Start_[]
codeMode = input("[E]ncode|[D]ecode: ").upper()
if codeMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ").upper()
codes = {
    'A':'.-.', 'N':'.-.', '1':'.----',
    'B':'-...', 'O':'.---', '2':'.---',
    'C':'.-.-.', 'P':'.---', '3':'.---',
    'D':'.-..', 'Q':'.---', '4':'.----',
    'E':'.', 'R':'.-.', '5':'.-----',
    'F':'.---', 'S':'.---', '6':'.-----',
    'G':'.---', 'T':'.-', '7':'.---',
    'H':'.-----', 'U':'.---', '8':'.---',
    'I':'.--', 'V':'.---', '9':'.---',
    'J':'.---', 'W':'.---', '0':'.-----',
    'K':'.-.-', 'X':'.-.-',
    'L':'.-..', 'Y':'.---',
    'M':'.--', 'Z':'.---',
}
def encodeDecode(mode, message, final = ""):
    if mode == 'E':
        for symbol in message:
            if symbol not in codes:
                message = message.replace(symbol, "")
        for symbol in message:
            final += codes[symbol] + ' '
    else:
        for code in message.split():
            for symbol in codes:
                if code == codes[symbol]:
                    final += symbol
    return final
print("Final message:",encodeDecode(codeMode, startMessage))
]_End

```



**// terminal**

**Start\_**

```
$ python main.py
```

```
[E]ncode|[D]ecode: e
```

```
Write the message: hello world
```

```
Final message: .... . .-.. .-. --- .-- --- .- .-. ..
```

```
$ python main.py
```

```
[E]ncode|[D]ecode: d
```

```
Write the message: .... . .-.. .-. --- .-- --- .- .-. ..
```

```
Final message: HELLOWORLD
```

**End**

```
# Переключатель режимов кодирования.
```

```
codeMode = input("[E]ncode|[D]ecode: ").upper()
```

```
# Если переключатель не равен 'E' или 'D', тогда вывести ошибку.
```

```
if codeMode not in ['E','D']:
```

```
    print("Error: mode is not found")
```

```
    raise SystemExit
```

```
# Сообщение для кодирования/декодирования.
```

```
startMessage = input("Write the message: ").upper()
```

```
# Коды Азбуки Морзе
```

```
codes = {
```

```
    'A':'.-.',      'N':'.-',      '1':'.----',
```

```
    'B':'-...',     'O':'---',      '2': '..---',
```

```
    'C':'.-.-.',    'P':'.--.',     '3': '...--',
```

```
    'D':'-..',      'Q':'-.-.',     '4': '....-',
```

```
    'E':'.',        'R':'.-.',     '5': '.....',
```

```
    'F':'.-.-.',    'S':'....',    '6': '-....',
```

```
    'G':'-.-.',     'T':'-',       '7': '--...',
```

```
    'H':'. ....',   'U':'.-.-.',   '8': '---..',
```

```
    'I':'. .',      'V':'. ...',   '9': '----.',
```

```
    'J':'.---',     'W':'.-.-.',   '0': '-----',
```

```
    'K':'.-.-.',    'X':'.-.-.',
```

```
    'L':'.-.-.',    'Y':'.-.-.',
```

```
    'M':'-.-.',     'Z':'.-.-.',
```

```
}
```

# Основная функция для кодирования/декодирования.

```
def encodeDecode(mode, message, final = ""):
```

# Если переключатель равен 'E', тогда закодировать сообщение.

```
if mode == 'E':
```

# Посимвольный перебор сообщения и если символ не находится в ключах кода, тогда удалить этот символ из сообщения.

```
for symbol in message:
```

```
    if symbol not in codes:
```

```
        message = message.replace(symbol, "")
```

# Посимвольный перебор сообщения и добавление к переменной final значение ключа из кодов.

```
for symbol in message:
```

```
    final += codes[symbol] + ' '
```

# Иначе, если переключатель равен 'D', тогда сделать перебор кодов в сообщении по разделителю-пробел.

```
else:
```

```
    for code in message.split():
```

# Перебор ключей в кодах.

```
for symbol in codes:
```

# Если код в сообщении равен значению ключа в кодах, тогда к переменной final добавить сам ключ.

```
if code == codes[symbol]:
```

```
    final += symbol
```

# Вернуть сообщение.

```
return final
```

# Вывод результата.

```
print("Final message:", encodeDecode(codeMode, startMessage))
```

## 402. ASCII кодировка.

(ASCII кодировка является таблицей, в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды.)

// image.png

Start\_

код	символ	код	символ	код	символ	код	символ	код	символ	код	символ
32	Пробел	48	.	64	@	80	P	96	'	112	p
33	!	49	0	65	A	81	Q	97	a	113	q
34	"	50	1	66	B	82	R	98	b	114	r
35	#	51	2	67	C	83	S	99	c	115	s
36	\$	52	3	68	D	84	T	100	d	116	t
37	%	53	4	69	E	85	U	101	e	117	u
38	&	54	5	70	F	86	V	102	f	118	v
39	'	55	6	71	G	87	W	103	g	119	w
40	(	56	7	72	H	88	X	104	h	120	x
41	)	57	8	73	I	89	Y	105	i	121	y
42	*	58	9	74	J	90	Z	106	j	122	z
43	+	59	:	75	K	91	[	107	k	123	{
44	,	60	;	76	L	92	\	108	l	124	
45	-	61	<	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

End\_

```

// main.py
Start_
codeMode = input("[E]ncode|[D]ecode: ").upper()
if codeMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ")
def encodeDecode(mode, message, final = ""):
    if mode == 'E':
        for symbol in message:
            final += str(ord(symbol)) + ' '
    else:
        for code in message.split():
            final += chr(int(code))
    return final
print("Final message:",encodeDecode(codeMode, startMessage))
]_End

```

// terminal

```

Start_
$ python main.py
[E]ncode|[D]ecode: e
Write the message: helloworld
Final message: 104 101 108 108 111 119 111 114 108 100

$ python main.py
[E]ncode|[D]ecode: d
Write the message: 104 101 108 108 111 119 111 114 108 100
Final message: helloworld
]_End

```

```

# Переключатель режимов кодирования.
codeMode = input("[E]ncode|[D]ecode: ").upper()

# Если переключатель не равен 'E' или 'D', тогда вывести ошибку.
if codeMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit

```

```
# Сообщение для кодирования/декодирования.
startMessage = input("Write the message: ").upper()

# Основная функция кодирования/декодирования.
def encodeDecode(mode, message, final = ""):

    # Если переключатель равен 'E', тогда сделать посимвольный перебор сообщения.
    if mode == 'E':
        for symbol in message:

            # К переменной final прибавлять число символа по таблице ASCII.
            final += str(ord(symbol)) + ' '

    # Иначе, если переключатель равен 'D', тогда сделать перебор кодов в сообщении
    # по разделителю-пробел.
    else:
        for code in message.split():

            # К переменной final прибавлять символ числа по таблице ASCII.
            final += chr(int(code))

    # Вернуть сообщение.
    return final

# Вывод результата.
print("Final message:", encodeDecode(codeMode, startMessage))
```

### 403. Двоичный код.

(Двоичный код состоит только из 0 и 1, но тем не менее двоичным кодом можно описать многое.)

// image.png

Start\_

Binary	Character	Binary	Character	Binary	Character	Binary	Character
00000000	NUL	00100000	SP	01000000	@	01100000	`
00000001	SOH	00100001	!	01000001	A	01100001	a
00000010	STX	00100010	"	01000010	B	01100010	b
00000011	ETX	00100011	#	01000011	C	01100011	c
00000100	EOT	00100100	\$	01000100	D	01100100	d
00000101	ENQ	00100101	%	01000101	E	01100101	e
00000110	ACK	00100110	&	01000110	F	01100110	f
00000111	BEL	00100111	'	01000111	G	01100111	g
00001000	BS	00101000	(	01001000	H	01101000	h
00001001	HT	00101001	)	01001001	I	01101001	i
00001010	LF	00101010	*	01001010	J	01101010	j
00001011	VT	00101011	+	01001011	K	01101011	k
00001100	FF	00101100	,	01001100	L	01101100	l
00001101	CR	00101101	-	01001101	M	01101101	m
00001110	SO	00101110	.	01001110	N	01101110	n
00001111	SI	00101111	/	01001111	O	01101111	o
00010000	DLE	00110000	0	01010000	P	01110000	p
00010001	DC1	00110001	1	01010001	Q	01110001	q
00010010	DC2	00110010	2	01010010	R	01110010	r
00010011	DC3	00110011	3	01010011	S	01110011	s
00010100	DC4	00110100	4	01010100	T	01110100	t
00010101	NAK	00110101	5	01010101	U	01110101	u
00010110	SYN	00110110	6	01010110	V	01110110	v
00010111	ETB	00110111	7	01010111	W	01110111	w
00011000	CAN	00111000	8	01011000	X	01111000	x
00011001	EM	00111001	9	01011001	Y	01111001	y
00011010	SUB	00111010	:	01011010	Z	01111010	z
00011011	ESC	00111011	;	01011011	[	01111011	{
00011100	FS	00111100	<	01011100	\	01111100	
00011101	GS	00111101	=	01011101	]	01111101	}
00011110	RS	00111110	>	01011110	^	01111110	~
00011111	US	00111111	?	01011111	_	01111111	DEL

\_End

**// main.py**

**Start\_**

```
codeMode = input("[E]ncode|[D]ecode: ").upper()
if codeMode not in ['E','D']:
    print("Error: mode is not found")
    raise SystemExit
startMessage = input("Write the message: ")
def encodeDecode(mode, message, final = ""):
    if mode == 'E':
        for symbol in message:
            final += "{0:08b} ".format(ord(symbol))
    else:
        for code in message.split():
            final += chr(int(code,2))
    return final
print("Final message:",encodeDecode(codeMode, startMessage))
End
```

**// terminal**

**Start\_**

\$ python main.py

[E]ncode|[D]ecode: e

Write the message: hello world

Final message: 01101000 01100101 01101100 01101100 01101111 00100000 01110111  
01101111 01110010 01101100 01100100

\$ python main.py

[E]ncode|[D]ecode: d

Write the message: 01101000 01100101 01101100 01101100 01101111 00100000  
01110111 01101111 01110010 01101100 01100100

Final message: hello world

**End**

# Переключатель режимов кодирования.

**codeMode = input("[E]ncode|[D]ecode: ").upper()**

# Если переключатель не равен 'E' или 'D', тогда вывести ошибку.

```
if codeMode not in ['E','D']:  
    print("Error: mode is not found")  
    raise SystemExit
```

# Сообщение для кодирования/декодирования.

```
startMessage = input("Write the message: ").upper()
```

# Основная функция кодирования/декодирования.

```
def encodeDecode(mode, message, final = ""):
```

# Если переключатель равен 'E', тогда сделать посимвольный перебор сообщения.

```
if mode == 'E':  
    for symbol in message:
```

# К переменной final прибавить двоичный код по ASCII таблице.

```
final += "{0:08b} ".format(ord(symbol))
```

# Иначе, если переключатель равен 'D', тогда сделать перебор кодов в сообщении по разделителю-пробел.

```
else:  
    for code in message.split():
```

# К переменной final прибавить значение двоичного кода по ASCII таблице.

```
final += chr(int(code,2))
```

# Вернуть сообщение.

```
return final
```

# Вывод результата.

```
print("Final message:",encodeDecode(codeMode, startMessage))
```



# Дополнительные скрипты

## 501. Шифровальщик файлов.

(Шифровальщик основан на AES шифровании, но зашифровывает он не один файл, а целую директорию, которая будет являться корневой директорией.

Осторожно: Файлы перезаписываются!)

// **main.py**

**Start\_**[

```
direct = input("Write the root directory: ")
```

```
password = input("Write the password: ")
```

```
with open("encrypt.py", "w") as encryptFile:
```

```
    encryptFile.write("""
```

```
import os
```

```
from sys import argv
```

```
from pyAesCrypt import encryptFile
```

```
def encrypt(file):
```

```
    print("-----")
```

```
    password = ""+str(password)+"
```

```
    bufferSize = 128*1024
```

```
    encryptFile(file, file+".crp", password, bufferSize)
```

```
    print("[encrypted] '{name}.crp".format(name = file))
```

```
    os.remove(file)
```

```
def walk(dir):
```

```
    for name in os.listdir(dir):
```

```
        path = os.path.join(dir, name)
```

```
        if os.path.isfile(path): encrypt(path)
```

```
        else: walk(path)
```

```
walk(""+str(direct)+"")
```

```
print("-----")
```

```
os.remove(argv[0])
```

```
""")
```

```
    print("[+] File 'encrypt.py' successfully saved!")
```

```
with open("decrypt.py", "w") as decryptFile:
```

```
    decryptFile.write("""
```

```
import os
```

```
from sys import argv
```

```

from pyAesCrypt import decryptFile
def decrypt(file):
    print("-----")
    password = ""+str(password)+"
    bufferSize = 128*1024
    decryptFile(file, os.path.splitext(file)[0], password, bufferSize)
    print("[decrypted] '{name}'".format(name = os.path.splitext(file)[0]))
    os.remove(file)
def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)
        if os.path.isfile(path):
            try: decrypt(path)
            except: pass
        else: walk(path)
walk(""+str(direct)+"")
print("-----")
os.remove(argv[0])
")
    print("[+] File 'decrypt.py' successfully saved!")
]_End

```

**// terminal**

**Start\_**

\$ python main.py

Write the root directory: /home/user/Templates/Python/TEST/

Write the password: helloworld

[+] File 'encrypt.py' successfully saved!

[+] File 'decrypt.py' successfully saved!

\$ python encrypt.py

-----  
[encrypted] '/home/user/Templates/Python/TEST/DIR/some.rb.crp'  
-----

[encrypted] '/home/user/Templates/Python/TEST/DIR/picture.jpeg.crp'  
-----

[encrypted] '/home/user/Templates/Python/TEST/crypter.py.crp'  
-----

[encrypted] '/home/user/Templates/Python/TEST/SOMETHING/main.py.crp'

-----  
[encrypted] '/home/user/Templates/Python/TEST/SOMETHING/file.txt.crp'  
-----

[encrypted] '/home/user/Templates/Python/TEST/decrypt.py.crp'  
-----

[encrypted] '/home/user/Templates/Python/TEST/encrypt.py.crp'  
-----

\$ python decrypt.py

-----  
[decrypted] '/home/user/Templates/Python/TEST/crypter.py'  
-----

[decrypted] '/home/user/Templates/Python/TEST/DIR/some.rb'  
-----

[decrypted] '/home/user/Templates/Python/TEST/DIR/picture.jpeg'  
-----

[decrypted] '/home/user/Templates/Python/TEST/encrypt.py'  
-----

[decrypted] '/home/user/Templates/Python/TEST/SOMETHING/file.txt'  
-----

[decrypted] '/home/user/Templates/Python/TEST/SOMETHING/main.py'  
-----

[decrypted] '/home/user/Templates/Python/TEST/decrypt.py'  
-----

**l\_End**

# Корневая директория с которой будет начинаться шифрование файлов.

**direct = input("Write the root directory: ")**

# Установка пароля для всех файлов.

**password = input("Write the password: ")**

# Создание python-скрипта, который будет зашифровывать файлы.

**with open("encrypt.py", "w") as encryptFile:**

**encryptFile.write("""**

# Импортирование модуля os и функций argv и encryptFile.

**import os**

**from sys import argv**

**from pyAesCrypt import encryptFile**

```

# Функция encrypt, которая отвечает за шифрование отдельного файла.
def encrypt(file):

# Перенос пароля из билдера в скрипт шифрования.
print("-----")
password = ''' +str(password) +'''

# Установка размера буфера для шифрования.
bufferSize = 128*1024

# Шифрование файла.
encryptFile(file, file+".crp", password, bufferSize)

# Вывести результат и удалить копию незашифрованного файла.
print("[encrypted] '{name}.crp'".format(name = file))
os.remove(file)

# Функция walk, которая отвечает за хождение по директориям и выборку файлов.
def walk(dir):

# Перебор всех файлов и директорий в выбранной директории.
for name in os.listdir(dir):

# Полный путь к файлу или директории.
path = os.path.join(dir, name)

# Если объект является файлом, тогда использовать функцию encrypt.
if os.path.isfile(path): encrypt(path)

# Если же объект является директорией, тогда перейти в эту директорию.
else: walk(path)

# Указание корневой директории из билдера.
walk('''+str(direct)+'')
print("-----")

# Удаление скриптового файла шифрования после выполнения всех функций.
os.remove(argv[0])
'''

# Вывод результата создания скрипта.
print("[+] File 'encrypt.py' successfully saved!")

```

# Создание python-скрипта, который будет расшифровывать файлы.

```
with open("decrypt.py","w") as decryptFile:  
    decryptFile.write("""
```

# Импортирование модуля os и функций argv и decryptFile.

```
import os  
from sys import argv  
from pyAesCrypt import decryptFile
```

# Функция decrypt, которая отвечает за расшифрование отдельного файла.

```
def decrypt(file):
```

# Перенос пароля из билдера в скрипт шифрования.

```
print("-----")  
password = ""+str(password)+"
```

# Установка размера буфера для шифрования.

```
bufferSize = 128*1024
```

# Расшифрование файла.

```
decryptFile(file, os.path.splitext(file)[0], password, bufferSize)
```

# Вывести результат и удалить копию зашифрованного файла.

```
print("[decrypted] '{name}'.format(name = os.path.splitext(file)[0]))  
os.remove(file)
```

# Функция walk, которая отвечает за хождение по директориям и выборку файлов.

```
def walk(dir):
```

# Перебор всех файлов и директорий в выбранной директории.

```
for name in os.listdir(dir):
```

# Полный путь к файлу или директории.

```
path = os.path.join(dir, name)
```

# Если объект является файлом, тогда попробовать расшифровать файл. Если же файл не зашифрован, тогда пропустить.

```
if os.path.isfile(path):  
    try: decrypt(path)  
    except: pass
```

# Если же объект является директорией, тогда перейти в эту директорию.

**else:** **walk**(path)

# Указание корневой директории из билдера.

**walk**(""+**str**(direct)+"")

**print**("-----")

# Удаление скриптового файла расшифрования после выполнения всех функций.

**os.remove**(argv[0])

''')

# Вывод результата создания скрипта.

**print**("["+**File** 'decrypt.py' successfully saved!")

## 502. Скрипт для создания .onion сайта в сети Tor.

(Скрипт создан для операционной системы Linux. Скрипт можно использовать не только для создания .onion сайта, но и для его последующего включения в сеть, т.к. скрипт проверяет наличие предыдущих директорий и файлов не давая их удалять.)

// main.py

Start\_1

```
#!/usr/bin/python3
```

```
# $ chmod +x main.py
```

```
# $ sudo ./main.py
```

```
from os import system, listdir, mkdir, chdir, getcwd
```

```
from time import sleep
```

```
'''
```

```
dist = ["apt-get install", "pacman -S"]
```

```
prog = ["tor"]
```

```
for distribution in dist:
```

```
    for program in prog:
```

```
        system("{dist} {prog}".format(dist = distribution, prog = program))
```

```
'''
```

```
www = [False, "/var/www/"]
```

```
onion = [False, "/var/www/onion/"]
```

```
main_files = "/var/lib/tor/onion/"
```

```
html_file = [False, "/var/www/onion/index.html"]
```

```
host_file = "/var/lib/tor/onion/hostname"
```

```
key_file = "/var/lib/tor/onion/private_key"
```

```
readme = [False, "README"]
```

```
torrc = [False, "/etc/tor/torrc"]
```

```
string1 = "HiddenServiceDir /var/lib/tor/onion"
```

```
string2 = "HiddenServicePort 80 127.0.0.1:80"
```

```
if "www" in listdir("/var/"):
    www[0] = True
```

```
if www[0] == False:
```

```
    mkdir(www[1])
```

```
    print("Directory '{dir}' created".format(dir = www[1]))
```

```
else:
```

```
    if "onion" in listdir(www[1]):
```

```
        onion[0] = True
```

```
if onion[0] == False:
```

```
    mkdir(onion[1])
```

```

        print("Directory '{dir}' created".format(dir = onion[1]))
if "index.html" in listdir(onion[1]):
    html_file[0] = True
if html_file[0] == False:
    with open(html_file[1], "w") as html:
        html.write("""
<!DOCTYPE html>
<html>
    <head>
        <title>Script_by_#%&!</title>
        <meta charset="utf-8">
    </head>
    <body>
        <p>Hello World!</p>
    </body>
</html> """)
        print("File '{file}' created".format(file = html_file[1]))
with open(torrc[1], "r") as tor:
    for string in tor:
        if string == string1 or string == string2:
            torrc[0] = True
            break
if torrc[0] == False:
    with open(torrc[1], "a") as tor:
        tor.write(string1 + "\n" + string2)
        print("Strings appended in the '{file}' file:".format(file = torrc[1]))
        print("{}'\n'{}'".format(string1, string2))
system("systemctl start tor.service")
system("systemctl restart tor.service")
sleep(1)
with open(host_file, "r") as host:
    hostname = host.read()
    print("File '{file}' read".format(file = host_file))
with open(key_file, "r") as key:
    private_key = key.read()
    print("File '{file}' read".format(file = key_file))
if readme[1] in listdir(getcwd()):
    readme[0] = True

```



```

if www[0] == False or onion[0] == False or html_file[0] == False or readme[0] ==
False:
    with open(readme[1], "w") as info:
        info.write("""
Tor configuration: """+torrc[1]+""":
- """+string1+""
- """+string2+"""\n
HTML file: """+html_file[1]+"""\n
Main files: """+main_files+""
- hostname ["""+host_file+"""]:
"""+hostname+""
- private_key ["""+key_file+"""]:
"""+private_key+""
Use this command for run tor service:
[for one time, after rebooting use this command again]
- systemctl start tor.service
[for always time]
- systemctl enable tor.service\n
Use this command for activate port 80:
[use in the directory: """+onion[1]+"""]
- python3 -m http.server 80
""")

    if readme[0] == True:
        print("File '{file}' updated".format(file = readme[1]))
    else:
        print("File '{file}' created".format(file = readme[1]))

chdir(onion[1])
system("python3 -m http.server 80")
]_End

```

## // terminal

### Start\_

```

$ chmod +x main.py
$ sudo ./main.py
Directory '/var/www/' created
Directory '/var/www/onion/' created
File '/var/www/onion/index.html' created
Strings appended in the '/etc/tor/torrc' file:
'HiddenServiceDir /var/lib/tor/onion'

```

```
'HiddenServicePort 80 127.0.0.1:80'  
File '/var/lib/tor/onion/hostname' read  
File '/var/lib/tor/onion/private_key' read  
File 'README' created  
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

**l\_End**

## // README

### Start\_1

Tor configuration: /etc/tor/torrc:

- HiddenServiceDir /var/lib/tor/onion
- HiddenServicePort 80 127.0.0.1:80

HTML file: /var/www/onion/index.html

Main files: /var/lib/tor/onion/

- hostname [/var/lib/tor/onion/hostname]:

jua25maeyrn4mthj.onion

- private\_key [/var/lib/tor/onion/private\_key]:

-----BEGIN RSA PRIVATE KEY-----

```
MIICXAIBAAKBgQD1Pg05PdBRbIjtACSnck2sTLWbqlLNiVgt+GcwFH5lDpeinIEM  
Bmn7dKIOEynMEz+7xCL0vltohgygD0Wvxp3AAtvheu9yzllXLoG2Ipmog5LlUjLZ  
MA+DKgAd0wd5BAXEFOeFVleI2ieSaXf//ER5nkulfVISFB+  
+YqmH90OoRQIDAQAB  
AoGAAqRSfCVgUyA6MWNpAazHjW2eHzksfy5VltkwM2JlIi2QEJ5i/YAsMBtHi6NF  
Nf+XFiy8u6o5Tdzz0d2YQJaBKIT5WPon4XLH05/cbGDTXHsQVrC2BudWpXIcevdN  
aNK2eVyeBcabYk8vz90sosq4NBzd/  
0e75hXR3qBE35VsTkECQQD7GqDLRM8ib47W  
DWDLBVJmlc8B+I89/9f1/5mY8AGUjlVIRIk2X4wpumMtqHcsqAHf69/CcKOLl6Qo  
C1zf0EqNAkEA+gYqYnuk4deBY1C2EVQLQyC2NGjXl5Kl2ho7yNKGntw11iSEz0b  
W  
xWnnFHfLw6WX/W1/  
e1wO6Ear8kcyPHoCmQJAP1qFjSOMOzz4RctUS4TJOHa9ptqm  
kVb2jutxPmP3khqjK7uW/u/2diS/lyp0/wBYkL17VByFNtgIo83SHen4lQJAHqo+  
7JfJqcFqsymRCxMJxpPuhQMO3jJIUTXCe2EGzdkoaTlVaK7BjLjudJ40yaw3tgeG  
CTVDRs3ULQT6blxwkQJBAMiY3hih3FB34NCtMzcVQt1gFtax7U8Uu7goo6k/  
3ADO  
wCgUM7XUDPtQ8B/Lz0w20JMxkO/D0WgvQWL2hRZE9MA=  
-----END RSA PRIVATE KEY-----
```

Use this command for run tor service:

[for one time, after rebooting use this command again]

- `systemctl start tor.service`

[for always time]

- `systemctl enable tor.service`

Use this command for activate port 80:

[use in the directory: `/var/www/onion/`]

- `python3 -m http.server 80`

## **]\_End**

```
# Даёт возможность chmod +x.
```

```
#!/usr/bin/python3
```

```
# Инструкция по запуску скрипта.
```

```
# $ chmod +x main.py
```

```
# $ sudo ./main.py
```

```
# Импортирование функций из модуля os и time.
```

```
from os import system, listdir, mkdir, chdir, getcwd
```

```
from time import sleep
```

```
# Раскомментировать, если нужно установить tor. (можно также дополнить список  
и дописать туда допустим nginx).
```

```
'''
```

```
dist = ["apt-get install", "pacman -S"]
```

```
prog = ["tor"]
```

```
for distribution in dist:
```

```
    for program in prog:
```

```
        system("{dist} {prog}".format(dist = distribution, prog = program))
```

```
'''
```

```
# Директории, файлы и строки, которые подлежат проверке или созданию.
```

```
www = [False, "/var/www/"]
```

```
onion = [False, "/var/www/onion/"]
```

```
main_files = "/var/lib/tor/onion/"
```

```
html_file = [False, "/var/www/onion/index.html"]
```

```
host_file = "/var/lib/tor/onion/hostname"
```

```
key_file = "/var/lib/tor/onion/private_key"
```

```
readme = [False, "README"]
```

```
torrc = [False, "/etc/tor/torrc"]
```

```
string1 = "HiddenServiceDir /var/lib/tor/onion"
```

```
string2 = "HiddenServicePort 80 127.0.0.1:80"
```

```
# Если директория 'www' уже находится в директории '/var/', тогда присвоить значение True.
```

```
if "www" in listdir("/var/"):
    www[0] = True
```

```
# Если www[0] всё-таки равно значению False (нет в директории '/var/'), тогда создать директорию 'www'.
```

```
if www[0] == False:
    mkdir(www[1])
    print("Directory '{dir}' created".format(dir = www[1]))
```

```
# Если же www[0] равно значению True, тогда проверить наличие директории 'onion' в директории '/var/www/'. Если директория найдена – присвоить True.
```

```
else:
    if "onion" in listdir(www[1]):
        onion[0] = True
```

```
# Если onion[0] всё-таки равно значению False (нет в директории '/var/www/'), тогда создать директорию 'onion'.
```

```
if onion[0] == False:
    mkdir(onion[1])
    print("Directory '{dir}' created".format(dir = onion[1]))
```

```
# Если в директории '/var/www/onion/' находится файл 'index.html', тогда присвоить значение True.
```

```
if "index.html" in listdir(onion[1]):
    html_file[0] = True
```

```
# Если всё-таки html_file[0] равен значению False, тогда создать html файл в директории '/var/www/onion/'.
```

```
if html_file[0] == False:
    with open(html_file[1], "w") as html:
        html.write(''
<!DOCTYPE html>
<html>
    <head>
        <title>Script_by_#%&!/</title>
        <meta charset="utf-8">
    </head>
    <body>
```

```

        <p>Hello World!</p>
    </body>
</html> '''
    print("File '{file}' created".format(file = html_file[1]))

# Открыть конфигурационный файл tor'а на чтение.
with open(torrc[1], "r") as tor:

# Перебор всех строк в файле 'torrc'.
for string in tor:

# Если строка равна строке1 или строке2, тогда присвоить значение True и
остановить цикл.
if string == string1 or string == string2:
    torrc[0] = True
    break

# Если же значение torrc[0] равно False, тогда открыть конфигурационный файл
tor'а на дополнение.
if torrc[0] == False:
    with open(torrc[1], "a") as tor:

# Внести в конец файла две строки: string1, string2.
tor.write(string1 + "\n" + string2)
print("Strings appended in the '{file}' file:".format(file = torrc[1]))
print('{0}\n{1}'.format(string1, string2))

# Запустить сервисы tor'а и подождать секунду.
system("systemctl start tor.service")
system("systemctl restart tor.service")
sleep(1)

# Открыть 'hostname' файл на чтение и скопировать всё в переменную hostname.
with open(host_file, "r") as host:
    hostname = host.read()
    print("File '{file}' read".format(file = host_file))

# Открыть 'private_key' файл на чтение и скопировать всё в переменную
private_key.
with open(key_file, "r") as key:
    private_key = key.read()
    print("File '{file}' read".format(file = key_file))

```

# Если файл 'README.txt' находится в директории запуска скрипта, тогда присвоить значение True.

```
if readme[1] in listdir(getcwd()):  
    readme[0] = True
```

# Если директория 'www' создавалась только при запуске скрипта или же директория 'onion' создавалась только при запуске скрипта, или же 'index.html' был создан только при запуске скрипта или же 'README' был создан только при запуске скрипта.

```
if www[0] == False or onion[0] == False or html_file[0] == False or readme[0] == False:
```

# Создать/перезаписать файл 'README'.

```
with open(readme[1], "w") as info:  
    info.write("""
```

```
Tor configuration: '''+torrc[1]+'':
```

```
- '''+string1+''
```

```
- '''+string2+''\n
```

```
HTML file: '''+html_file[1]+''\n
```

```
Main files: '''+main_files+''
```

```
- hostname ['''+host_file+'']:
```

```
'''+hostname+''
```

```
- private_key ['''+key_file+'']:
```

```
'''+private_key+''
```

```
Use this command for run tor service:
```

```
[for one time, after rebooting use this command again]
```

```
- systemctl start tor.service
```

```
[for always time]
```

```
- systemctl enable tor.service\n
```

```
Use this command for activate port 80:
```

```
[use in the directory: '''+onion[1]+'']
```

```
- python3 -m http.server 80
```

```
""")
```

# Если файл 'README' до этого был уже создан, тогда написать, что файл обновлён, иначе написать, что он создан.

```
if readme[0] == True:
```

```
    print("File '{file}' updated".format(file = readme[1]))
```

```
else:
```

```
    print("File '{file}' created".format(file = readme[1]))
```

# Перейти в директорию '/var/www/onion/' и от туда запустить python сервер.

**chdir(onion[1])**

**system("python3 -m http.server 80")**

# Дополнительная информация

*Криптография является одной из старейших наук и до сих пор в ней находится очень много нераскрытых, а в некоторых случаях пугающих тайн, которые порождают человеческий интерес*

Дело «Тамам Шуд»: [https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%BE\\_%C2%AB%D0%A2%D0%B0%D0%BC%D0%B0%D0%BC%D0%A8%D1%83%D0%B4%C2%BB](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%BE_%C2%AB%D0%A2%D0%B0%D0%BC%D0%B0%D0%BC%D0%A8%D1%83%D0%B4%C2%BB)

Дело YOGTZE: [https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%BE\\_YOGTZE](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%BE_YOGTZE)

G-AGWH Star Dust: [https://ru.wikipedia.org/wiki/G-AGWH\\_Star\\_Dust#STENDEC](https://ru.wikipedia.org/wiki/G-AGWH_Star_Dust#STENDEC)

Исчезновение Фредерика Валентича:

[https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%87%D0%B5%D0%B7%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%B8%D0%B5%D0%A4%D1%80%D0%B5%D0%B4%D0%B5%D1%80%D0%B8%D0%BA%D0%B0\\_%D0%92%D0%B0%D0%BB%D0%B5%D0%BD%D1%82%D0%B8%D1%87%D0%B0](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%87%D0%B5%D0%B7%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%B8%D0%B5%D0%A4%D1%80%D0%B5%D0%B4%D0%B5%D1%80%D0%B8%D0%BA%D0%B0_%D0%92%D0%B0%D0%BB%D0%B5%D0%BD%D1%82%D0%B8%D1%87%D0%B0)

Зодиак: [https://ru.wikipedia.org/wiki/%D0%97%D0%BE%D0%B4%D0%B8%D0%B0%D0%BA\\_\(%D1%83%D0%B1%D0%B8%D0%B9%D1%86%D0%B0\)](https://ru.wikipedia.org/wiki/%D0%97%D0%BE%D0%B4%D0%B8%D0%B0%D0%BA_(%D1%83%D0%B1%D0%B8%D0%B9%D1%86%D0%B0))

Записки Рикки Маккормика:

[https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BF%D0%B8%D1%81%D0%BA%D0%B8\\_%D0%A0%D0%B8%D0%BA%D0%BA%D0%B8\\_%D0%9C%D0%B0%D0%BA%D0%BA%D0%BE%D1%80%D0%BC%D0%B8%D0%BA%D0%B0](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BF%D0%B8%D1%81%D0%BA%D0%B8_%D0%A0%D0%B8%D0%BA%D0%BA%D0%B8_%D0%9C%D0%B0%D0%BA%D0%BA%D0%BE%D1%80%D0%BC%D0%B8%D0%BA%D0%B0)

Криптограммы Бейла: [https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B\\_%D0%91%D0%B5%D0%B9%D0%BB%D0%B0](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B_%D0%91%D0%B5%D0%B9%D0%BB%D0%B0)



## *Книги связанные с криптографией*

“Книга шифров. Тайная история шифров и их расшифровки”: [https://mega.nz/#!FlczADAY!jTR9vaYWumN22laXQSm\\_hjA7CjyFz3KV1RZ8gvqcUT4](https://mega.nz/#!FlczADAY!jTR9vaYWumN22laXQSm_hjA7CjyFz3KV1RZ8gvqcUT4)

“Протоколы, алгоритмы и исходные тексты на языке C”:  
[https://mega.nz/#!c09Ena6R!OVes3cf7pCTFkH3DaoId2fD0QLojgS7Oa7NJ\\_ictGo4](https://mega.nz/#!c09Ena6R!OVes3cf7pCTFkH3DaoId2fD0QLojgS7Oa7NJ_ictGo4)

“Таинственные страницы. Занимательная криптография”:  
[https://mega.nz/#!YlFzgB5C!axg3d3U1l4ix6Qgo4h1g3HOvrZec20ZSabU\\_1BPND\\_g](https://mega.nz/#!YlFzgB5C!axg3d3U1l4ix6Qgo4h1g3HOvrZec20ZSabU_1BPND_g)

**GitHub профиль автора:** 104 116 116 112 115 58 47 47 103 105 116 104 117 98 46 99  
111 109 47 78 117 109 98 101 114 53 55 49

**YouTube канал автора:** <https://www.youtube.com/channel/UCwFnt>