Eenigma

Данная программа работает по принципу алгоритма энигмы.
Алгоритм взят из статьи: https://habr.com/ru/post/217331/

Программа(main.py) имеет консольный интерфейс, а настройка роторов и коммутатора происходит в файле settings.py.
Для кодирования используются только латинские символы (как и в оригинальной энигме).
Создать новый ротор или коммутатор можно при помощи файла rotor_generator.py (запускать через idle).

Описание работы функций

```
def generate_rotor_layer():
    rotor_layer = [0 for i in range(26)]
    used = []
    for i in range(26):
        err = 1
        while err == 1:
            r_num = r.randint(0,25)
            if not(r_num in used):
                err = 0
                rotor_layer[i] = r_num
                used.append(r_num)
    return rotor_layer
```

Данная функция генерирует рандомный уровень для ротора таким образом, чтобы каждые 2 буквы латинского алфавита образовали пару.

```
def generate_rotor():
    layers = [generate_rotor_layer() for i in range(13)]
    layers_second_part = [[0 for i in range(26)] for j in range(13)]
    for i in range(len(layers)):
        for j in range(len(layers[i])):
            layers_second_part[i][layers[i][j]] = j

    for layer in layers_second_part:
        layers.append(layer)

    return layers
```

Данная функция сначала генерирует 13 уровней, поскольку в роторе 2 уровня соединены между собой, а затем создает обратные им уровни и создает ротор, состоящий из 26 уровней (26 букв, между ними создаются пары).

Пример ротора в программе:

```
rotor1 = [[22, 18, 7, 3, 8, 11, 24, 12, 9, 21, 0, 2, 4, 1, 14, 13, 10, 25, 6, 5, 16, 19, 23, 17, 20, 15],
[25, 20, 10, 24, 1, 11, 7, 0, 16, 18, 6, 12, 22, 14, 23, 19, 13, 5, 3, 15, 9, 4, 21, 2, 17, 8],
[1, 9, 19, 13, 22, 11, 7, 0, 14, 5, 6, 25, 16, 23, 10, 21, 3, 17, 4, 20, 2, 18, 8, 12, 15, 24],
[20, 24, 22, 9, 25, 17, 0, 19, 4, 5, 16, 10, 13, 12, 8, 2, 21, 7, 15, 18, 3, 23, 11, 1, 6, 14],
[8, 22, 7, 19, 9, 13, 20, 1, 0, 12, 24, 15, 6, 17, 4, 2, 21, 25, 11, 14, 10, 18, 16, 23, 5, 3],
[10, 23, 1, 13, 8, 18, 0, 21, 17, 25, 11, 14, 4, 16, 19, 6, 5, 9, 24, 2, 3, 20, 22, 7, 12, 15],
[8, 17, 13, 22, 23, 18, 10, 4, 11, 0, 24, 16, 5, 15, 21, 25, 7, 6, 3, 2, 19, 1, 14, 20, 12, 9],
[17, 18, 15, 20, 13, 0, 11, 6, 10, 2, 1, 19, 7, 3, 8, 9, 21, 24, 5, 16, 25, 12, 23, 4, 22, 14],
[13, 1, 8, 24, 20, 10, 15, 4, 22, 14, 11, 25, 0, 18, 7, 23, 19, 12, 6, 5, 21, 17, 3, 9, 16, 2],
[24, 10, 2, 19, 9, 15, 16, 22, 4, 0, 14, 8, 23, 13, 12, 6, 17, 25, 3, 20, 5, 1, 21, 18, 7, 11],
[19, 3, 1, 2, 4, 21, 7, 11, 25, 20, 15, 12, 13, 16, 24, 18, 23, 9, 5, 10, 0, 14, 6, 8, 17, 22],
[25, 24, 2, 13, 0, 15, 4, 11, 3, 1, 7, 5, 12, 23, 21, 10, 14, 6, 17, 19, 8, 22, 16, 20, 9, 18],
[5, 15, 20, 17, 10, 0, 11, 19, 23, 8, 3, 4, 1, 13, 7, 2, 25, 22, 14, 12, 9, 21, 16, 24, 18, 6],
[10, 13, 11, 3, 12, 19, 18, 2, 4, 8, 16, 5, 7, 15, 14, 25, 20, 23, 1, 21, 24, 9, 0, 22, 6, 17],
[7, 4, 23, 18, 21, 17, 10, 6, 25, 20, 2, 5, 11, 16, 13, 19, 8, 24, 9, 15, 1, 22, 12, 14, 3, 0],
[7, 0, 20, 16, 18, 9, 10, 6, 22, 1, 14, 5, 23, 3, 8, 24, 12, 17, 21, 2, 19, 15, 4, 13, 25, 11],
[6, 23, 15, 20, 8, 9, 24, 17, 14, 3, 11, 22, 13, 12, 25, 18, 10, 5, 19, 7, 0, 16, 2, 21, 1, 4],
[8, 7, 15, 25, 14, 24, 12, 2, 0, 4, 20, 18, 9, 5, 19, 11, 22, 13, 21, 3, 6, 16, 1, 23, 10, 17],
[6, 2, 19, 20, 12, 16, 15, 23, 4, 17, 0, 10, 24, 3, 11, 25, 13, 8, 5, 14, 21, 7, 22, 1, 18, 9],
[9, 21, 19, 18, 7, 12, 17, 16, 0, 25, 6, 8, 24, 2, 22, 13, 11, 1, 5, 20, 23, 14, 3, 4, 10, 15],
[5, 10, 9, 13, 23, 18, 7, 12, 14, 15, 8, 6, 21, 4, 25, 2, 19, 0, 1, 11, 3, 16, 24, 22, 17, 20],
[12, 1, 25, 22, 7, 19, 18, 14, 2, 23, 5, 10, 17, 0, 9, 6, 24, 21, 13, 16, 4, 20, 8, 15, 3, 11],
[9, 21, 2, 18, 8, 20, 15, 24, 11, 4, 1, 25, 14, 13, 10, 5, 6, 16, 23, 3, 19, 22, 7, 12, 0, 17],
[20, 2, 3, 1, 4, 18, 22, 6, 23, 17, 19, 7, 11, 12, 21, 10, 13, 24, 15, 0, 9, 5, 25, 16, 14, 8],
[4, 9, 2, 8, 6, 11, 17, 10, 20, 24, 15, 7, 12, 3, 16, 5, 22, 18, 25, 19, 23, 14, 21, 13, 1, 0],
[5, 12, 15, 10, 11, 0, 25, 14, 9, 20, 4, 6, 19, 13, 18, 1, 22, 3, 24, 7, 2, 21, 17, 8, 23, 16]]
```

```python
def generate_commutator():
    commutator = dict()
    used = []
    for i in range(10):
        err1 = 1
        while err1 == 1:
            r_num1 = r.randint(0,25)
            if not(r_num1 in used):
                err1 = 0
                err2 = 1
                used.append(r_num1)
                while err2 == 1:
                    r_num2 = r.randint(0,25)
                    if not(r_num2 in used):
                        err2 = 0
                        used.append(r_num2)
                        commutator[r_num1] = r_num2
                        commutator[r_num2] = r_num1
    return commutator
```

Коммутатор представляет из себя 10 связей между 20 символами.
Функция находит рандомный неиспользованный символ и строит связь с новым рандомным неиспользованным символом.
В конечном итоге функция возвращает словарь, состоящий из связей(пример: 3: 25; 3 – символ d, 25 – символ z) и обратных связей(25: 3)

Пример коммутатора в программе:

```
commutator = {19: 13, 13: 19, 25: 0, 0: 25, 6: 7, 7: 6, 8: 24, 24: 8, 22: 17, 17: 22, 14: 1, 1: 14, 9: 5, 5: 9, 3: 18, 18: 3, 20: 21, 21: 20, 23: 11, 11: 23}
```

Для создания необходимого элемента необходимо раскомментировать определенную часть кода:

```
#для вывода рандомного коммутатора
"""
print(generate_commutator())
"""

#для вывода рандомного ротора
"""
rotor = generate_rotor()
for elem in rotor:
    print(elem, end="")
    print(", ")
"""
```

```
def Eencode(s, settt, rotors, commutator):
    rotor1 = rotors[0]
    rotor2 = rotors[1]
    rotor3 = rotors[2]

    sett = settt[0:]
    out = ""
    for el in s:
        s_num = ord(el) - 97

        s_num = rotor1[sett[0]][s_num]
        s_num = rotor2[sett[1]][s_num]
        s_num = rotor3[sett[2]][s_num]

        if s_num in commutator:
            s_num = commutator[s_num]
        out += chr(s_num+97)

        if sett[0] != 25:
            sett[0] += 1
        else:
            sett[0] = 0
            if sett[1] != 25:
                sett[1] += 1
            else:
                sett[1] = 0
                if sett[2] != 25:
                    sett[2] += 1
                else:
                    sett[2] = 0

    return out
```

Функция кодирования последовательности отправляет каждый символ на роторы, затем на коммутатор.
Перед кодированием нового символа функция передвигает положения роторов.
Функция возвращает зашифрованную последовательность.

```
def Edecode(s, settt, rotors, commutator):
    rotor1 = rotors[0]
    rotor2 = rotors[1]
    rotor3 = rotors[2]

    sett = settt[0:]
    out = ''
    for el in s:
        s_num = ord(el) - 97

        if s_num in commutator:
            s_num = commutator[s_num]

        s_num = rotor3[(sett[2]+13)%26][s_num]
        s_num = rotor2[(sett[1]+13)%26][s_num]
        s_num = rotor1[(sett[0]+13)%26][s_num]

        out += chr(s_num+97)

        if sett[0] != 25:
            sett[0] += 1
        else:
            sett[0] = 0
            if sett[1] != 25:
                sett[1] += 1
            else:
                sett[1] = 0
                if sett[2] != 25:
                    sett[2] += 1
                else:
                    sett[2] = 0

    return out
```

Функция отправляет символы в обратном порядке относительно кодирования.
Сначала на коммутатор, затем на роторы, на роторах символы отправляются на обратные уровни (запись (sett[i]+13) % 26).
Перед кодированием нового символа функция передвигает положения роторов.
Функция возвращает исходную(расшифрованную) строку.

В работе были использованы библиотеки: unittest, random.