Санкт-Петербургский государственный инженерно-экономический университет.

Кафедра вычислительных систем и программирования

Лабораторная работа «Шифрование»

Составитель: Федоров Д.Ю.

http://pycode.ru/edu/

Оглавление

1. Предварительные сведения

В штаб разведки попали десять зашифрованных сообщений, содержащих важные сведения о расположении космических спутников противника. Командующий штабом поручил вам дешифровать сообщения и в срочном порядке доложить ему все дешифрованные координаты.

Известно, что для шифрования использовался метод Цезаря и английский алфавит. В штабе нашлась реализация алгоритма шифрования.

Ваша задача: изменить программу шифрования таким образом, чтобы она перебирала все возможные смещения (ключи) в методе Цезаря и с новой строки выводила получившиеся дешифрованные строки.

Далее представлен интерфейс работы с программой шифрования / дешифрования.

После запуска программы следуем инструкциям.

>>> Вы хотите зашифровать или дешифровать сообщение?

e

>>> Введите сообщение:

engec

>>> Укажите ключ (1-26)*:*

3

>>> Полученный текст:

hqjhf

>>>

>>> Вы хотите зашифровать или дешифровать сообщение?

d

>>> Введите сообщение:

hqjhf

>>> Укажите ключ (1-26):

3

>>> Полученный текст:

engec

>>>

Так как ключ нам не известен, а сообщения могут быть зашифрованы разными ключами, то напишем программу перебора всех возможных чисел со следующим интерфейсом:

>>> Вы хотите зашифровать или дешифровать или подобрать сообщение?

b

>>> Введите сообщение:

hqjhf

>>> Полученный текст:

(1, 'gpige')

(2, 'fohfd')

(3, 'engec')

(4, 'dmfdb')

(5, 'cleca')

(6, 'bkdbz')

(7, 'ajcay')

(8, 'zibzx')

(9, 'yhayw')

И.т.д.

2. Теоретические сведения

2.1. Шифр Цезаря

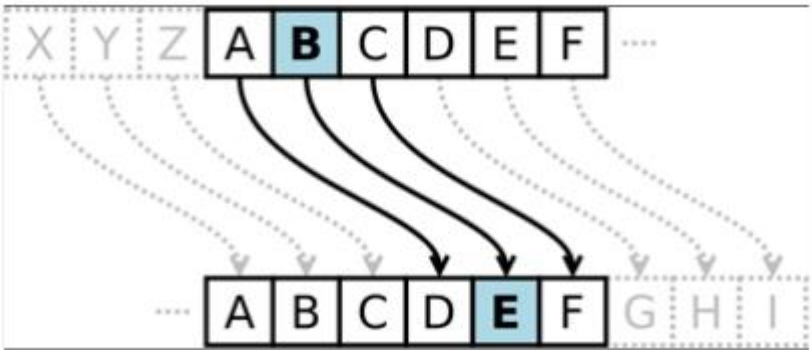

Рисунок. Смещение на три позиции, «B» становится «E»

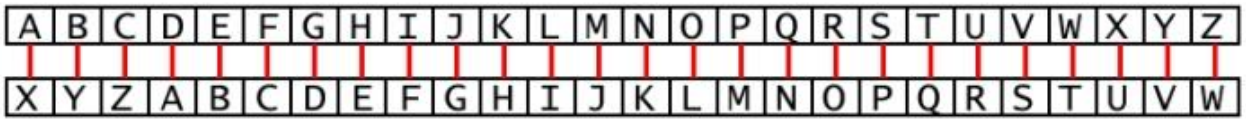

Рисунок. Весь алфавит со смещением на три позиции

2.2. ASCII или использование чисел для букв

Как реализовать смещение букв в программе? Это можно сделать, представив каждую букву в виде числа (называемого порядковым числительным, от англ. *ordinal*) и затем сложением или вычитанием из этого числа сформировать новое число (и новую букву). ASCII (произносится как “ask-ee” и является американским стандартным кодом для обмена сообщениями) – это код, который сопоставляет каждый символ с числом от 32 до 127. Числа меньше 32 “непечатные”, поэтому мы не будем их использовать. Заглавные буквы от “A” до “Z” в ASCII имеют номера от 65 до 90. Строчные буквы от “a” до “z” имеют номера от 97 до 122. Числам от “0” до “9” соответствуют ASCII номера от 48 до 57.

32	(space)	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	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		

Таблица ASCII символов.

И так, если мы хотим сместить “A” на три позиции, то сначала преобразуем его в число (65). Затем к полученному числу прибавим 3. После этого преобразуем число 65+3=68 обратно в букву (“D”). Воспользуемся стандартными функциями языка Python *chr()* и *ord()* для преобразований между буквами и числами.

2.3. Функции chr( ) и ord( )

Функция *chr()* (производная от “char”, сокращенное “character”, “символ”) принимает в качестве параметра целое ASCII число и возвращает строку, состоящую из единственного символа. Функция *ord()* (сокращенное от “ordinal”) в качестве параметра принимает строку, состоящую из одного символа, и возвращает целое ASCII значение для этого символа.

Выполните следующие команды в интерактивном режиме:

```
>>> chr(65)
'A'
>>> ord('A')
65
>>> chr(65+8)
'I'
>>> chr(52)
'4'
>>> chr(ord('F'))
'F'
>>> ord(chr(68))
68
>>>
```

Функции *chr()* и *ord()* мы будем использовать в нашей программе.

2.4. Исходный текст программы шифрования / дешифрования

```
# Caesar Cipher
MAX_KEY_SIZE = 26
def getMode():
    while True:
        print('Do you wish to encrypt or decrypt a message?')
        mode = raw_input().lower()
        if mode in 'encrypt e decrypt d'.split():
```

```
            return mode
        else:
            print('Enter either "encrypt" or "e" or "decrypt" or "d".')


def getMessage():
    print('Enter your message:')
    return raw_input()


def getKey():
    key = 0
    while True:
        print('Enter the key number (1-%s)' % (MAX_KEY_SIZE))
        key = int(raw_input())
        if (key >= 1 and key <= MAX_KEY_SIZE):
            return key


def getTranslatedMessage(mode, message, key):
    if mode[0] == 'd':
        key = -key
    translated = ''

    for symbol in message:
        if symbol.isalpha():
            num = ord(symbol)
            num += key

            if symbol.isupper():
                if num > ord('Z'):
                    num -= 26
```

```
                elif num < ord('A'):
                    num += 26
            elif symbol.islower():
                if num > ord('z'):
                    num -= 26
                elif num < ord('a'):
                    num += 26

            translated += chr(num)
        else:
            translated += symbol
    return translated

mode = getMode()
message = getMessage()
key = getKey()

print('Your translated text is:')
print(getTranslatedMessage(mode, message, key))
```

2.5. Пояснения к исходному тексту программы

```
# Caesar Cipher
MAX_KEY_SIZE = 26
```

Первая строка – это обычный комментарий, который пропускается интерпретатором.

MAX_KEY_SIZE – это переменная, которая содержит число 26. *MAX_KEY_SIZE* служит напоминанием, что в программе используется ключ в интервале от 1 до 26.

```
def getMode():
    while True:
        print('Do you wish to encrypt or decrypt a message?')
        mode = raw_input().lower()
        if mode in 'encrypt e decrypt d'.split():
            return mode
        else:
            print('Enter either "encrypt" or "e" or "decrypt" or "d".')
```

Функция *getMode()* (с помощью бесконечного цикла) позволяет пользователю задать режим работы программы: шифрование (encrypt e) или дешифрование (decrypt d). Возвращаемое *raw_input()* (метод *lower()* возвращает строку в нижнем регистре) значение сохраняется в *mode*. Условное выражение проверяет, содержит ли введенная строка *mode* один из элементов списка [‘encrypt’, ‘e’, ‘decrypt’, ‘d’], полученного в результате выполнения *‘encrypt e decrypt d’.split()*. Можно использовать список, но для упрощения ввода чаще всего используют метод *split()*.

Выполните в интерактивном режиме интерпретатора Python:

```
>>> ‘encrypt e decrypt d’.split()
['encrypt', 'e', 'decrypt', 'd']
```

split() может принимать входной параметр – разделитель, по умолчанию разделителем является пробел.

```
>>> ‘encrypt <> e <> decrypt <> d’.split(‘<>’)
```

['encrypt ', ' e ', ' decrypt ', ' d']

Следующий участок кода:

```
def getMessage():
    print('Enter your message:')
    return raw_input()
```

Функция *getMessage()* получает от пользователя строку для шифрования или дешифрования и использует эту строку в качестве возвращаемого значения.

```
def getKey():
    key = 0
    while True:
        print('Enter the key number (1-%s)' % (MAX_KEY_SIZE))
        key = int(raw_input())
        if (key >= 1 and key <= MAX_KEY_SIZE):
            return key
```

Функция *getKey()* позволяет указать ключ, который будет использоваться для шифрования или дешифрования сообщения. Цикл *while* повторяется до тех пор, пока пользователь не введет действительный ключ, т.е. попадающий в интервал от 1 до *MAX_KEY_SIZE*.

Оператор % предназначен для работы со строками, он играет роль функции *sprint* в языке *C*.

Выполните в интерактивном режиме интерпретатора Python:

>>> name = “Ivan”

```
>>> “My name is %s!” % name
‘My name is Ivan!’
>>>
```

В примере строка “Ivan” подключается к указанной строке, замещая спецификатор %s.

Функция *raw_input()* возвращает введенную пользователем строку, которую необходимо преобразовать в число (с помощью *int()*), чтобы далее работать с числовыми значениями.

Функция *getKey()* возвращает значение указанного пользователем ключа.

```
def getTranslatedMessage(mode, message, key):
    if mode[0] == 'd':
        key = -key
    translated = ''
```

getTranslatedMessage() – функция, которая занимается шифрованием и дешифрованием в программе. На вход функции поступают три параметра:

mode – устанавливает режим шифрования или дешифрования,

message – открытый текст (или зашифрованный текст), который надо зашифровать (или дешифровать),

key – ключ, который используется в процессе шифрования.

Первая строка функции определяет режим шифрования или дешифрования. Если первой буквой переменной *mode* является ‘d’, то устанавливается режим дешифрования. Единственное отличие между двумя режимами состоит в знаке ключа. Если в качестве ключа было выбрано целое число 22, тогда для режима дешифрования оно устанавливается как -22.

translated – строка, которая будет содержать конечный результат: шифротекст (если мы использовали режим шифрования) или открытый текст (если использовался режим дешифрования). Мы будем присоединять строки к этой переменной, формируя результат. Переменная должна быть определена с помощью некоторой строки, прежде чем мы сможем использовать ее для присоединения строк.

isalpha() – строковый метод, который возвращает *True*, если строка состоит из букв нижнего или верхнего регистра от *A* до *Z*. Если строка содержит любые нестроковые символы, тогда *isalpha()* вернет *False*.

Выполните следующие выражения в интерактивном режиме:

```
>>> 'Hello'.isalpha()
True
>>> 'Forty two'.isalpha()
False
>>> 'Fortytwo'.isalpha()
True
>>> '42'.isalpha()
False
>>> ''.isalpha()
False
>>>
```

Далее мы воспользуемся методом *isalpha()*.

```
for symbol in message:
  if symbol.isalpha():
      num = ord(symbol)
      num += key
```

В цикле мы пробегаем по всем буквам (символам) строки сообщения. Строки обрабатываются как списки, содержащие строки, состоящие из единичных символов. Например, для строки ‘Hello’ список будет иметь вид [‘H’, ‘e’, ‘l’, ‘l’, ‘o’]. На каждой итерации во время выполнения цикла, *symbol* будет иметь значение буквы из *message*.

Шифруются и дешифруются только буквы из *message*, поэтому необходима проверка всех символов. Числа, знаки, знаки препинания и остальные символы остаются в первоначальном виде. Переменная *num* содержит числовое значение, соответствующее букве, хранящейся в переменной *symbol*. Затем значение из *num* “смещается” на значение из *key*, это выражение эквивалентно *num = num + key*.

Далее, в программе используются строковые методы *isupper()* и *islower()*. Они очень похожи на методы *isdigit()* и *isalpha()*. *isupper()* возвращает *True*, если вызывающая его строка содержит, по крайней мере, одну заглавную букву и не содержит букв в нижнем регистре. *islower()* возвращает *True*, если строка содержит, по крайней мере, одну букву в нижнем регистре и не содержит заглавных букв. В противном случае эти методы возвращают *False*. Наличие небуквенных символов, таких как числа и пробелы, не влияет на результат.

Попробуйте ввести следующие выражения в интерактивном режиме интерпретатора Python.

```
>>> 'HELLO'.isupper()

True

>>> 'hello'.isupper()

False

>>> 'hello'.islower()

True

>>> 'Hello'.islower()
```

```
False
>>> 'LOOK OUT BEHIND YOU!'.isupper()
True
>>> '42'.isupper()
False
>>> '42'.islower()
False
>>> ''.isupper()
False
>>> ''.islower()
False
>>>
```

Продолжим изучать код программы.

```
if symbol.isupper():
    if num > ord('Z'):
        num -= 26
    elif num < ord('A'):
        num += 26
```

Этот код проверяет, является ли *symbol* буквой в верхнем регистре. Если это так, то нужно позаботиться о двух особых случаях. Что произойдет, если *symbol* является буквой ‘Z’ и был задан ключ равный 4? В этом случае значением *num* будет символ ‘^’ (числовое значение символа ‘^’ равно 94). Но ‘^’ не является буквой, поэтому в этом случае нам необходимо начинать отсчет с начала алфавита.

Мы проверяем выход за границы числового значения, соответствующего последней буквы алфавита (“Z”) и если такая проверка заканчивается успешно, то уменьшаем числовое значение на 26, по количеству букв в английском алфавите. После этого значение переменной *num* станет равным 68, что соответствует в ASCII значению ‘D’.

```
        elif symbol.islower():
            if num > ord('z'):
                num -= 26
            elif num < ord('a'):
                num += 26
```

Если *symbol* – буква в нижнем регистре, то программа выполняет код, схожий с выполняемым ранее. Единственное отличие в использовании *ord(‘z’)* и *ord(‘a’*) вместо *ord(‘Z’)* и *ord(‘A’)*.

В режиме дешифрования мы можем получить случай, когда *num* станет меньше, чем наименьшее возможное значение (в ASCII число 65, соответствующее букве ‘A’). В этом случае мы добавляем к *num* 26 и, тем самым, учитываем смещение.

```
            translated += chr(num)
        else:
            translated += symbol
```

Переменная *translated* служит “накопителем” результирующей строки. Если *symbol* содержит букву верхнего или нижнего регистров, то после преобразования она добавляется к результирующей строке *translated*. В случае, когда *symbol* содержит символ, отличный от буквы, он добавляется к *translated,* не подвергаясь преобразованию, в исходном виде.

```
    return translated
```

В последней строке функции *getTranslatedMessage()* возвращается результирующая строка *translated*.

```
mode = getMode()

message = getMessage()

key = getKey()


print('Your translated text is:')

print(getTranslatedMessage(mode, message, key))
```

Это основная часть программы. Мы вызываем каждую из трех ранее определенных функций, чтобы получить значения *mode*, *message* и *key*. Затем эти три значения передаются в качестве входных параметров в функцию *getTranslatedMessage()*, которая возвращает результат (преобразованный текст), выводимый на экран пользователя.

Что надо изменить в алгоритме, чтобы программа перебирала все ключи и выводила соответствующие им дешифрованные сообщения?

Подсказки:

для реализации программы можно воспользоваться, например, циклом *for* в связке с функцией *range, range* генерирует индексы в цикле *for*.

Попробуйте ввести следующие выражения в интерактивном режиме интерпретатора Python.

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(1, 5)
[1, 2, 3, 4]
>>> for i in range(3):
…       print i
0
1
2
>>>
```

3. Литература для дополнительного чтения

1. Марк Лутц. Изучаем Python
2. Al Sweigart. Invent Your Own Computer Games with Python, 2nd Edition.