

`__import__()` 函数用于动态加载类和函数。  
如果一个模块经常变化就可以使用 `__import__()` 来动态载入。

# The Vigenere cipher 维吉尼亚密码

Eric Martin, CSE, UNSW

COMP9021 Principles of Programming, trimester 1, 2019

```
In [1]: from string import printable
        from collections import Counter
        from operator import itemgetter
        from itertools import product
        from re import split
```

可打印的  
考虑一下用大写字母写的信息。凯撒代码在两者之间固定了一个自然数  $n$  1和25，并将所有字母  $n$  的位置向右移动，在需要时进行换行，保持所有其他字符(如空格)不变。例如，消息 ALEA JACTA EST 变为，水平的向 右边移动

凯撒的代码

Consider a message all written in capital letters. The *Caesar code* fixes a natural number  $n$  between 1 and 25 and shifts all letters  $n$  positions to the right, wrapping around when needed, leaving all other characters such as spaces unchanged. For instance, the message ALEA JACTA EST becomes

- with  $n = 1$ : BMFB KBDUB FTU
- with  $n = 2$ : CNGC LCEVC GUV
- ...
- with  $n = 10$ : KVOK TKMDK OCD
- ...
- with  $n = 25$ : ZKDZ IZBSZ DRS

The *Vigenere cipher* offers a generalisation as follows. Choose a word written all in uppercase letters as the key, for instance, RAW. Since RAW consists of 3 letters, write the letters that make up the message over 3 columns:

ALE  
AJA  
CTA  
EST

Vigenere密码提供了一个概括如下。选择一个全用大写字母写的单词例如，作为键RAW。由于RAW由3个字母组成，所以将组成消息的字母写在3列上也就是说，给的键有几位，那么原来的句子就按几位 写出来

As R, A and W are the 18th, 1st and 23rd letters of the alphabet, respectively, all letters in the first column are shifted 18 positions to the right, all letters in the second column are shifted 1 position to the right, and all letters in the third column are shifted 23 positions to the right, wrapping around when needed:

SMB  
SKX  
UUX  
WTQ

按列移动， R A W 这三个字母在字母表中的位置数是多少，那么 1 2 3 列分别向后移动多少位

密文

This results in the encrypted message: SMBS KXUUX WTQ 维吉尼亚密

Let us not distinguish between (capital) letters and other characters, but rather let the Vigenere cipher encrypt all characters in a message with a key that itself can be any sequence of characters. The `string` module defines a string consisting of all printable ASCII characters; it could be a good candidate for which characters to allow both in message and key:

让我们不要区分(大写)字母和其他字符，而是让Vigenere密码使用密钥加密消息中的所有字符，密钥本身可以是任何字符序列。字符串模块定义一个由所有可打印ASCII字符组成的字符串;它可能是一个很好的候选人，字符允许在消息和关键字:

In [1]: `printable` 打印

Out[1]: `'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'`

`'\r'` is the carriage return character: 回车字符 `{ } ".format("hello", "world")` # 不设置指定位置, 按默认顺序  
`'hello world'`

In [2]: `print('00000\r11')` `>>> "{0} {1}".format("hello", "world")` # 设置指定位置  
`'hello world'`

11000 `>>> "{1} {0} {1}".format("hello", "world")` # 设置指定位置  
`'world hello world'`

ASCII characters can be represented as `\x` followed by their ASCII code in the form of two hexadecimal digits: ASCII 字符可以表示为 `\x`, 后跟两个十六进制数字形式的 ASCII 代码

In [3]: `ord('c'), f"{ord('c'):x}", '\x63'`  
# *f-strings cannot embed backslashes*, 字符串不能嵌入反斜杠,  
# *so we instead use the format string method*. 格式字符串的方法  
`ord('\n'), '{:x}'.format(ord('\n')), '\x0a'`

Out[3]: (99, '63', 'c') `ord()` 函数是 `chr()` 函数 (对于8位的ASCII字符串) 或 `unichr()` 函数 (对于Unicode对象) 的配对函数, 它  
以一个字符 (长度为1的字符串) 作为参数, 返回对应的 ASCII 数值, 或者 Unicode 数值, 如果所给的 Unicode 字符超出了你的 Python 定义范围, 则会引发一个 `TypeError` 的异常. `ord(c)` c -- 字符 `ord('a')`  
Out[3]: (10, 'a', '\n') 97 `>>> ord('b')` 98 `>>> ord('c')` 99

`'\x0b'` and `'\x0c'` are the vertical tab and form feed characters, respectively; Jupyter ignores them. For code run from the command line, `'\x0b'` and `'\x0c'` behave the same: the next character follows the previous one but on the next line (form feed is meant to force a printer to move to the next sheet of paper). Both `'\x0b'` and `'\x0c'` actually have escaped characters as alternatives, namely, `'\v'` and `'\f'`, respectively:

`'\x0b'` 和 `'\x0c'` 是垂直制表符和换页符,

In [4]: `'\v', '\f'` `'\x0b'` 和 `'\x0c'` 实际上都有转义字符作为替代品, 即 `'\v'` 和 `'\f'`,

Out[4]: (`'\x0b'`, `'\x0c'`) 忽略打印中的最后3个字符并接受所有其他字符作为消息和键的可能组成部分是明智的:

It is sensible to ignore the last 3 characters in `printable` and accept all others as the possible constituents of messages and keys:

In [5]: `characters = printable[:-3]`  
`len(characters)`

Out[5]: 97

字符中的任何字符都可以在键中使用, 并根据消息中的字符数移动来确定; 移位的值由制作的字符的字符位置给出  
每次需要时, 在字符中查找给定字符的位置是低效的: 例如, 如果一个键长7个字符并以 'G' 开头, 则效率低下  
从头开始从左到右扫描字符, 在位置42找到 'G', 并推断加密消息中的第一个, 第八个, 第五十个... 字符应该向右移动42个位置, 如果需要可以绕过。我们定义一个字典, 其中字符作为键, 字符中的位置作为值, 因此后者可以在前面的恒定时间内获得

Any character in `characters` can be used in keys and determine by how much to shift characters in messages; the values of the shifts are given by the positions in `characters` of the characters that make up the key. It would be inefficient to look for the position of a given character in `characters` every time it is needed: for instance, if a key was 7 characters long and started with 'G', it would be inefficient to scan characters left to right from the beginning, to find 'G' at position 42 and infer that the first, eight, fiftieth... characters in the message to encrypt should be shifted 42 positions to the right, wrapping around if needed. We define a dictionary with the characters in `characters` as keys, and their positions in `characters` as values, so the latter can be obtained from the former in constant time:

这是一个字典 我们定义一个字典，其中字符作为键，字符中的位置作为值，

```
In [6]: shifts = {characters[i]: i for i in range(len(characters))}
        print(shifts)

{'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, ...
... '9': 9, 'a': 10, 'b': 11, 'c': 12, 'd': 13, 'e': 14, 'f': 15, ...
... 'g': 16, 'h': 17, 'i': 18, 'j': 19, 'k': 20, 'l': 21, 'm': 22, ...
... 'n': 23, 'o': 24, 'p': 25, 'q': 26, 'r': 27, 's': 28, 't': 29, ...
... 'u': 30, 'v': 31, 'w': 32, 'x': 33, 'y': 34, 'z': 35, 'A': 36, ...
... 'B': 37, 'C': 38, 'D': 39, 'E': 40, 'F': 41, 'G': 42, 'H': 43, ...
... 'I': 44, 'J': 45, 'K': 46, 'L': 47, 'M': 48, 'N': 49, 'O': 50, ...
... 'P': 51, 'Q': 52, 'R': 53, 'S': 54, 'T': 55, 'U': 56, 'V': 57, ...
... 'W': 58, 'X': 59, 'Y': 60, 'Z': 61, ' ': 62, '"': 63, '#': 64, ...
... '$': 65, '%': 66, '&': 67, "'": 68, '(': 69, ')': 70, '*': 71, ...
... '+': 72, ',': 73, '-': 74, '.': 75, '/': 76, ':': 77, ';': 78, ...
... '<': 79, '=': 80, '>': 81, '?': 82, '@': 83, '[': 84, '\\': 85, ...
... ']': 86, '^': 87, '_': 88, '`': 89, '{': 90, '|': 91, '}': 92, ...
... '~': 93, '\t': 94, '\n': 95, '\r': 96}
```

加密

Let us give an example of a message and its encryption with the key 'Take it Easy!' (we will see below how the encryption has been <sup>计算</sup>computed, now it is given and supposed to have been properly computed). 'Take it Easy!' is a key of length 13, so we align the key, the message displayed over 13 columns, and the encrypted message displayed over 13 columns. As the message length is not a multiple of 13, the last lines of message and encrypted message are less than 13 characters long; formatted strings 格式化字符串 can given a field width to display a string, padding with spaces to the right for shorter strings, ignoring the field width for longer strings: 格式化字符串 可以给出一个字段宽度来显示一个字符串，用右边的空格填充较短的字符串，忽略较长字符串的字段宽度

```
In [7]: print('|'.join(f'{w:4}' for w in ('Alice', 'was', 'beginning',
                                         'to', 'get', 'very', 'tired'
                                         )))
```

所以这里两个空格

Alice|was |beginning|to |get |very|tired

Next to key and message, we display the ASCII codes of the characters in key and message, similarly aligned. We observed that characters has 97 characters.

- 加密的消息
- The **first character** in key, namely, 'T', has position 55 in characters. The first character in message, namely 'A', has position 36 in characters; it is therefore encrypted in encrypted\_message as the character that in characters, has position  $(55 + 36) \bmod 97 = 91 \bmod 97 = 91$ , namely, '| '.
  - The **fifth character** in key, namely a space, has position 94 in characters. The fifth character in message, namely 'e', has position 14 in characters; it is therefore encrypted in encrypted\_message as the character that in characters, has position  $(94 + 14) \bmod 97 = 108 \bmod 97 = 11$ , namely, 'b'.

```
In [8]: message = ("Alice was beginning to get very tired of sitting by her "
                  "sister on the bank, and of having nothing to do: once or "
```

```

        "twice she had peeped into the book her sister was reading, "
        "but it had no pictures or conversations in it, 'and what "
        "is the use of a book,' thought Alice, 'without pictures or "
        "conversations?'"
    )

    encrypted_message = ( '|vCqbfZ7\'7DM;,xHwkyqq#7IM|QFyFvfWf&oFv])7MwqLL'
        'kU7D\'X+oLbpAVqSBpW\\QDBs|tDkY@pI\\\'7It|zDsWxI'
        'v\\<DBwkyqq#7FWGQyHqbfRoBDYQ-(7MvbfK7R7RM/=oxbf'
        'FWlBDJMX%yIy|zHoBCK!|(BhK7KqoSfFQ\\*@hprLqf(7JI'
        '.QxIbmAFq)BG!X<BhqLFYb&CC"=<xMbFfQf(@p5+;nhKesW'
        '|WCp"<(70GbFRcBkpJ]<u~?|LKl)qJ"X|vCqb|q$+sVP]\\\'
        'DhDfuWr&oUv]?7wCkNHo\'kVQ];C5?'
    )

    key = 'Take it Easy!'
    print(f'{key}' + ' ' + ' '.join(f'{shifts[c]:2}' for c in key))
    print()
    print('\n'.join(f'{message[i * 13: (i + 1) * 13]:13}' + ' ' +
        ' '.join(f'{shifts[c]:2}'
            for c in message[i * 13: (i + 1) * 13])
        for i in range(len(message) // 13 + 1))
    )
    print()
    print('\n'.join(f'{encrypted_message[i * 13: (i + 1) * 13]:13}' + ' ' +
        ' '.join(f'{shifts[c]:2}'
            for c in encrypted_message[i * 13: (i + 1) * 13])
        for i in range(len(encrypted_message) // 13 + 1))
    )
    print()

```

Take it Easy! 55 10 20 14 94 18 29 94 40 10 28 34 62

Alice was beg	36	21	18	12	14	94	32	10	28	94	11	14	16
inning to get	18	23	23	18	23	16	94	29	24	94	16	14	29
very tired o	94	31	14	27	34	94	29	18	27	14	13	94	24
f sitting by	15	94	28	18	29	29	18	23	16	94	11	34	94
her sister on	17	14	27	94	28	18	28	29	14	27	94	24	23
the bank, an	94	29	17	14	94	11	10	23	20	73	94	10	23
d of having n	13	94	24	15	94	17	10	31	18	23	16	94	23
othing to do:	24	29	17	18	23	16	94	29	24	94	13	24	77
once or twic	94	24	23	12	14	94	24	27	94	29	32	18	12
e she had pee	14	94	28	17	14	94	17	10	13	94	25	14	14
ped into the	25	14	13	94	18	23	29	24	94	29	17	14	94
book her sist	11	24	24	20	94	17	14	27	94	28	18	28	29
er was readin	14	27	94	32	10	28	94	27	14	10	13	18	23

g, but it had	16	73	94	11	30	29	94	18	29	94	17	10	13
no pictures	94	23	24	94	25	18	12	29	30	27	14	28	94
or conversati	24	27	94	12	24	23	31	14	27	28	10	29	18
ons in it, 'a	24	23	28	94	18	23	94	18	29	73	94	68	10
nd what is th	23	13	94	32	17	10	29	94	18	28	94	29	17
e use of a bo	14	94	30	28	14	94	24	15	94	10	94	11	24
ok,' thought	24	20	73	68	94	29	17	24	30	16	17	29	94
Alice, 'witho	36	21	18	12	14	73	94	68	32	18	29	17	24
ut pictures o	30	29	94	25	18	12	29	30	27	14	28	94	24
r conversatio	27	94	12	24	23	31	14	27	28	10	29	18	24
ns?'	23	28	82	68									

vCqbfZ7'7DM;	91	31	38	26	11	15	61	7	68	7	39	48	78
,xHwkyqq#7IM	73	33	43	32	20	34	26	26	64	7	44	48	91
QFyFvWf&oFv]	52	41	34	41	31	15	58	15	67	24	41	31	86
)7MwqLLkU7D'X	70	7	48	32	26	47	47	20	56	7	39	68	59
+oLbpAVqSBpW\	72	24	47	11	25	36	57	26	54	37	25	58	85
QDBs tDKY@pI\	52	39	37	28	91	29	39	20	60	83	25	44	85
'7It zDsWxIv\	68	7	44	29	91	35	39	28	58	33	44	31	85
<DBwkyqq#7FWG	79	39	37	32	20	34	26	26	64	7	41	58	42
QyHqbfRoBDYQ-	52	34	43	26	11	15	53	24	37	39	60	52	74
(7MvbfK7R7RM/	69	7	48	31	11	15	46	7	53	7	53	48	76
=oxbfFWlBDJMX	80	24	33	11	15	41	58	21	37	39	45	48	59
%yIy zHoBCK!	66	34	44	34	91	35	43	24	37	38	46	62	91
(BhK7KqoSfQ\	69	37	17	46	7	46	26	24	54	20	41	52	85
*@hprLqf(7JI.	71	83	17	25	27	47	26	15	69	7	45	44	75
QxIbmAFq)BG!X	52	33	44	11	22	36	41	26	70	37	42	62	59
<BhqLFYb&CC"=	79	37	17	26	21	41	60	11	67	38	38	63	80
<xMbFfQf(@p5+	79	33	48	11	15	41	26	15	69	83	25	5	72
;nhKesW WCp"<	78	23	17	46	14	28	58	91	58	38	25	63	79
(7OGbfRcBkpJ]	69	7	50	42	11	15	53	12	37	20	25	45	86
<u~? LKl)qJ"X	79	30	93	82	91	47	46	21	70	26	45	63	59
vCqb q\$+sVP]	91	31	38	26	11	91	26	65	72	28	57	51	86
\DhDfuWr&oUv]	85	39	17	39	15	30	58	27	67	24	56	31	86
?7wCkNHo'kvQ]	82	7	32	38	20	49	43	24	68	20	57	52	86
;C5?	78	38	5	82									

加密

解密

Let us write a function, `encrypt()`, to encrypt a message, and a function, `decrypt()`, to decrypt an encrypted message, both with a key  $K$  provided as first argument, the message being provided as second argument. Both functions perform almost identically: 两种功能几乎完全相同

为了加密消息 $M$ ，每次遍历 $M$ 中的每个字符，确定其索引模 $K$ 的长度，并将其向右移动字符为 $n$ ，其中 $n$ 为字符的位置索引 $i$ 的 $K$ 中的字符，如果需要可以包裹

- To encrypt a message  $M$ , one goes over each character in  $M$ , determine its index  $i$  modulo the length of  $K$ , and shifts it to the right in characters by  $n$  with  $n$  the position in characters of the character in  $K$  of index  $i$ , wrapping around if needed.
- To decrypt an encrypted message  $E$ , one goes over each character in  $E$ , determine its index  $i$  modulo the length of  $K$ , and shifts it to the left in characters by  $n$  with  $n$  equal to the position in characters of the character in  $K$  of index  $i$ , wrapping around if needed.

#### 辅助函数

It is therefore natural to define an auxiliary function, `encrypt_or_decrypt()` called by both `encrypt()` and `decrypt()`, being directed by either whether shifting should be “to the right” or “to the left”, which corresponds to adding or subtracting positions: 这对应于增加或减少位置

```
In [9]: def encrypt(key, text):
        return encrypt_or_decrypt(key, text, 1)

        def decrypt(key, text):
            return encrypt_or_decrypt(key, text, -1)

        模式
        def encrypt_or_decrypt(key, text, mode):
            return ''.join(characters[(shifts[text[i]] +
                                      shifts[key[i % len(key)]] * mode
                                      ) % len(shifts)]
                            for i in range(len(text))

        )

        encrypt(key, message)
        print()
        decrypt(key, encrypted_message)
        print()
        encrypt(key, decrypt(key, encrypted_message))
        print()
        decrypt(key, encrypt(key, message))

Out[9]: '|vCqbfZ7\'7DM;,xHwkyqq#7IM|QFyFvfwf&oFv])7MwqLLkU7D\'X+oLbpAVqSBpw\\Q...
...DBs|tDkY@pI\\\'7It|zDsWxIv\\<DBwkyqq#7FWGQyHqbfRoBDYQ-(7MvbfK7R7RM/=oxbf...
...FWlBDJMX%yIy|zHoBCK!|(BhK7KqoSfQ\\*@hprLqf(7JI.QxIbmAFq)BG!X<BhqLFYb&CC...
...<=xMbFqf(@p5+;nhKesW|WCp"<(70GbRcBkpJ]<u~?|LKl)qJ"X|vCqb|q$+sVP]\\DhD...
...fuWr&oUv]?7wCkNHo\'kVQ];C5?'
```

```
Out[9]: "Alice was beginning to get very tired of sitting by her sister on the...
...bank, and of having nothing to do: once or twice she had peeped into the...
...book her sister was reading, but it had no pictures or conversations in...
...it, 'and what is the use of a book,' thought Alice, 'without pictures or...
...conversations?'"
```

```
Out[9]: '|vCqbfZ7\'7DM;,xHwkyqq#7IM|QFyFvfwf&oFv])7MwqLLkU7D\'X+oLbpAVqSBpw\\Q...
...DBs|tDkY@pI\\\'7It|zDsWxIv\\<DBwkyqq#7FWGQyHqbfRoBDYQ-(7MvbfK7R7RM/=oxbf...
...FWlBDJMX%yIy|zHoBCK!|(BhK7KqoSfQ\\*@hprLqf(7JI.QxIbmAFq)BG!X<BhqLFYb&CC...
...<=xMbFqf(@p5+;nhKesW|WCp"<(70GbRcBkpJ]<u~?|LKl)qJ"X|vCqb|q$+sVP]\\DhD...
...fuWr&oUv]?7wCkNHo\'kVQ];C5?'
```

```
Out[9]: "Alice was beginning to get very tired of sitting by her sister on the...
...bank, and of having nothing to do: once or twice she had peeped into the...
...book her sister was reading, but it had no pictures or conversations in...
...it, 'and what is the use of a book,' thought Alice, 'without pictures or...
...conversations?'"
```

Let us now tackle the task of breaking the code of an encrypted message  $E$ , that is, discovering the key  $K$  that was used to encrypt a message  $M$  into  $E$ , and decrypting  $E$  into  $M$  with  $K$ . A program can only tentatively suggest candidates for  $K$  that the user would then have to validate. Also, the program might not always succeed, especially when the encrypted message is short. Let us fix an upper bound on the length of  $K$ , the first of a number of parameters we will use in our heuristics:

```
In [10]: max_key_length = 16
```

With that value, the number of possible keys is huge:

```
In [11]: sum(len(characters) ** i for i in range(1, 17))
```

```
Out[11]: 62065212901958868055012327674640
```

现在让我们来解决破解加密消息E的代码的任务，即发现密钥K，用于将消息M加密到E，并用E解密为M。一个程序可以只是暂时建议用户必须验证的K候选人。此外，该计划可能并不总是成功，尤其是当加密消息很短时。让我们修一个上限K的长度，我们将在我们的启发式中使用的许多参数中的第一个：

If the message is long enough, it is likely to contain many  $n$ -grams (i.e.,  $n$  consecutive characters) that occur many times, and less likely but still likely enough,  $n$ -grams with many vertically aligned occurrences. For instance, “the”, “ing”, “able” could be  $n$ -grams (with  $n$  equal to 3, 3 and 4, respectively) some of whose occurrences in a message could be aligned:

```
... the...
.....
. able....
.....ing
.....
.....ing
.....
.....
.....
... the...
. able....
... the...
.....
.....
.....ing
```

连续的特点  
如果消息足够长，则可能包含许多n-gram（即，n个连续字符）发生多次，并且不太可能但仍然可能足够，n-gram与许多垂直对齐的事件发生。例如，“the”，“ing”，“able”可以是n-gram（n分别等于3,3和4）其中一些消息中的某些内容可以对齐

All vertically aligned occurrences of a given  $n$ -gram are identically encrypted. Hence one can look for  $n$ -grams in the encrypted message that occur many times, and assume that they have a significant chance to correspond to  $n$ -grams in the original message that happen to be vertically aligned. Suppose for instance that %O occurs many times in the encrypted message. Suppose that the second occurrence of %O is 40 positions to the right of the first occurrence of %O. Conjecturing that both occurrences of %O are aligned, and assuming that the key is at most 16 characters long, there are 4 options (writing ~ for a slot for a character):

给定n-gram的所有垂直对齐出现都是相同加密的。因此，可以在加密消息中查找多次出现的n-gram，并假设它们有很大机会对应于恰好垂直对齐的原始消息中的n-gram。假设为 % ) O在加密消息中多次出现的实例。假设第二次出现的 % ) O是第一次出现 % ( ) O的右边的40个位置。猜测两个出现的 % ) O是对齐的，并且假设密钥长度最多为16个字符，则有4个选项（写入~表示角色的插槽）：



- The key is 4 characters long and the second occurrence of %)O is 10 lines below the first occurrence of %)O, e.g.:

```

~ %) O
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ ~ ~ ~
~ %) O

```

发生

- The key is 5 characters long and the second occurrence of %)O is 8 lines below the first occurrence of %)O, e.g.:

```

~ %) O ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~
~ ~ ~ ~ ~
~ %) O ~

```

%)O 在每一行出现的位置是一样的，间隔40

- The key is 8 characters long and the second occurrence of %)O is 5 lines below the first occurrence of %)O, e.g.:

```

~ ~ %) O ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ %) O ~ ~ ~

```

- The key is 10 characters long and the second occurrence of %)O is 4 lines below the first occurrence of %)O, e.g.:

```

~ ~ ~ %) O ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ %) O ~ ~ ~

```

克 公分

In other words, the conjecture that two successive occurrences of an  $n$ -gram are aligned leads to the conclusion that the length of the key is a divisor at least equal to  $n$  and at most equal to the value of `max_key_length` of the distance between (the start of) both occurrences. The following function returns a generator expression for the relevant divisors of its first argument, namely, those of its factors that are at least equal to its second argument and at most equal to the value of `max_key_length`:

换句话说，两个连续出现的 $n$ -gram对齐的猜想得出的结论是，密钥的长度是一个除数，至少等于 $n$ ，最多等于 $n$ 的值。

`max_key_length`两次出现之间的距离（开始）。以下函数返回其第一个参数的相关除数的生成器表达式，即其因子的至少等于其第二个参数的因子，最多等于`max_key_length`的值



```
In [12]: def factors_from_successive_n_grams(num, min_value):
        return (i for i in range(min_value, max_key_length + 1)
                if num % i == 0)           % 取模，返回除法的余数

        tuple(factors_from_successive_n_grams(40, 3))
```

Out[12]: (4, 5, 8, 10)

检索  
`factors_from_successive_n_grams()` returns a generator expression; each factor will be retrieved on demand, e.g., by calls to `next()`, or thanks to a `for` statement. We would like to generate on demand not just the relevant factors of the distance between two successive occurrences of a given  $n$ -gram, but the relevant factors of the distances of all pairs of successive occurrences of all  $n$ -grams (for the chosen values of  $n$ ). To this aim, we introduce another mechanism, namely, generator functions. A generator function is a function with **yield statements** in its body. A `yield` statement can, but does not have to, be followed by an expression. In case an expression follows, the value of that expression is provided when the `yield` statement is executed; otherwise, `None` is provided. In any case, execution of the generator function then stops, waiting to be resumed, possibly, on demand:

```
In [13]: def f(n):
        print(f'Before yielding {n}')
        yield n
        print(f'Before yielding {n + 1}')
        yield n + 1
        print('Before yielding None')
        yield
        print(f'Before yielding {n + 2}')
        yield n + 2
        print(f'After yielding {n + 2}')

        F = f(10)
        F

        next(F)
        next(F)
        next(F)
        next(F)
        next(F)
```

Out[13]: <generator object f at 0x108745308>

Before yielding 10

Out[13]: 10

Before yielding 11

Out[13]: 11

Before yielding None  
Before yielding 12

Out[13]: 12

After yielding 12

---

```
StopIteration                                Traceback (most recent call last)

<ipython-input-13-e01b2fdaadee> in <module>()
    17 next(F)
    18 next(F)
----> 19 next(F)
```

StopIteration:

A generator function can also contain return statements:

```
In [14]: def f(n):
        print(f'Before yielding {n}')
        yield n
        print(f'Before yielding {n + 1}')
        yield n + 1
        print('Before returning')
        return
        # Won't be printed out
        print('After returning')
        # Won't be yielded
        yield n + 2
```

```
F = f(10)
F
```

```
next(F)
next(F)
next(F)
next(F)
next(F)
```

Out[14]: <generator object f at 0x1086892b0>

Before yielding 10

Out[14]: 10

Before yielding 11

Out[14]: 11

Before returning

```
-----  
StopIteration                                Traceback (most recent call last)  
  
<ipython-input-14-83fd57077d03> in <module>()  
    16 next(F)  
    17 next(F)  
----> 18 next(F)  
    19 next(F)  
    20 next(F)
```

StopIteration:

生成器函数可以产生由另一个生成器函数或任何其他可迭代函数生成的值：

A generator function can yield values yielded by another generator function or any other iterable:

```
In [15]: def f():  
        yield 1  
        yield 2  
  
        def g():  
            yield 0  
            for x in f():  
                yield x  
            for x in range(3, 6):  
                yield x  
            for x in (x for x in [6, 7, 8]):  
                yield x  
            for x in [9, 10]:  
                yield x  
  
        G = g()  
        for x in G:  
            print(x, end = ' ')
```

0 1 2 3 4 5 6 7 8 9 10

更简洁的选择

In such a context, yield from statements offer a more concise alternative:

```
In [16]: def f():
        yield 1
        yield 2

        def g():
            yield 0
            yield from f()
            yield from range(3, 6)
            yield from (x for x in [6, 7, 8])
            yield from [9, 10]

        list(g())
```

```
Out[16]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

连续发生

启发式

Let us get back to the problem of discovering successive occurrences of  $n$ -grams. As part of our heuristics, let us set the sizes of  $n$ -grams to look for to the default values of 3, 4 and 5:

```
In [17]: n_gram_range = range(3, 6)
```

To look for substrings within a string, we can use either `find()` or `index()` str's methods. They are exactly the same, except that `find()` returns `-1` whereas `index()` raises a `ValueError` exception when they fail to find their argument as a substring. Both methods take an optional second argument and a second optional third argument to limit the search within a narrower range than the full string to which the method is applied:

要查找字符串中的子字符串，我们可以使用`find()`或`index()` str的方法。它们完全相同，只是`find()`返回-1，而`index()`在找不到它们作为子字符串的参数时引发`ValueError`异常。两种方法都采用可选的第二个参数和第二个可选的第三个参数来限制搜索范围，而不是完整的字符串

```
In [18]: '123123'.find('2')
        '123123'.find('2', 2)
        '123123'.find('2', 2, 4)
```

```
Out[18]: 1
```

```
Out[18]: 4
```

```
Out[18]: -1
```

When we find a pair of occurrences of a 5-gram, we are guaranteed to find as well 2 pairs of occurrences of 4-grams, and 3 pairs of occurrences of 3-grams, but there does not seem to be a good reason to prevent the redundancy (in particular because between two consecutive occurrences of an  $n$ -gram that extends a substring  $s$ , there can be other occurrences of  $s$  in between. The following code fragment takes as example the first 91 characters of `message` (so the first 7 lines of `message` displayed over 13 columns as above) and collects in a list the relevant factors of the distances between all successive occurrences of all 3, 4 and 5-grams in this text. We trace execution and display, in particular, the contexts in which two successive occurrences of a 3, 4 or 5-gram have been detected:

```
In [19]: text = message[: 91]

        relevant_factors = []
        for n in n_gram_range:
```

当我们发现一对5克的出现时，我们保证找到2对4克的出现，3对3克的出现，但似乎没有充分的理由防止冗余（特别是因为在两个连续出现的 $n$ -gram之间扩展了一个子串 $s$ ，其间可能会出现 $s$ 。下面的代码片段以消息的前91个字符为例（所以前7个）消息行显示在13列以上）并在清单中收集本文中所有连续出现的3,4和5克之间距离的相关因素。我们跟踪执行和显示，特别是检测到连续两次出现3,4或5克的上下文：

```

print(f'Looking for consecutive occurrences of {n}-grams:')
for i in range(len(text) - 2 * n + 1):
    n_gram = text[i: i + n]
    j = text.find(n_gram, i + n)
    if j != -1:
        print(f' Found "{n_gram}" at indexes {i} and {j}\n'
              f' {text[i: j + n]}\n'
              f' Distance: {j - i}. Adding as factors:'
              f' {tuple(factors_from_successive_n_grams(j - i, n))}\n'
              )
        relevant_factors.extend(
            factors_from_successive_n_grams(j - i, n)
        )

print(relevant_factors)

```

Looking for consecutive occurrences of 3-grams:

Found "ing" at indexes 16 and 45  
ing to get very tired of sitting  
Distance: 29. Adding as factors: ()

Found "ng " at indexes 17 and 46  
ng to get very tired of sitting  
Distance: 29. Adding as factors: ()

Found "d o" at indexes 36 and 78  
d of sitting by her sister on the bank, and o  
Distance: 42. Adding as factors: (3, 6, 7, 14)

Found " of" at indexes 37 and 79  
of sitting by her sister on the bank, and of  
Distance: 42. Adding as factors: (3, 6, 7, 14)

Found "of " at indexes 38 and 80  
of sitting by her sister on the bank, and of  
Distance: 42. Adding as factors: (3, 6, 7, 14)

Found " si" at indexes 40 and 55  
sitting by her si  
Distance: 15. Adding as factors: (3, 5, 15)

Found "ing" at indexes 45 and 86  
ing by her sister on the bank, and of having  
Distance: 41. Adding as factors: ()

Found "ng " at indexes 46 and 87  
ng by her sister on the bank, and of having  
Distance: 41. Adding as factors: ()

```

Found "er " at indexes 53 and 60
er sister
Distance: 7. Adding as factors: (7,)

```

Looking for consecutive occurrences of 4-grams:

```

Found "ing " at indexes 16 and 45
ing to get very tired of sitting
Distance: 29. Adding as factors: ()

```

```

Found "d of" at indexes 36 and 78
d of sitting by her sister on the bank, and of
Distance: 42. Adding as factors: (6, 7, 14)

```

```

Found " of " at indexes 37 and 79
of sitting by her sister on the bank, and of
Distance: 42. Adding as factors: (6, 7, 14)

```

```

Found "ing " at indexes 45 and 86
ing by her sister on the bank, and of having
Distance: 41. Adding as factors: ()

```

Looking for consecutive occurrences of 5-grams:

```

Found "d of " at indexes 36 and 78
d of sitting by her sister on the bank, and of
Distance: 42. Adding as factors: (6, 7, 14)

```

```
[3, 6, 7, 14, 3, 6, 7, 14, 3, 6, 7, 14, 3, 5, 15, 7, 6, 7, 14, 6, 7, 14, 6, 7, 14]
```

事实证明，在前七行中，我们没有得到单因子13：没有3-、4-或5-gram两次对齐连续出现。无论如何，搜索连续出现的n-gram都是在加密消息中进行的，结果是不同的；我们使用原始信息来更友好地说明该技术

It turns out that in these first seven lines, we do not get a single factor of 13: no 3-, 4- or 5-gram has two aligned successive occurrences. The search for successive occurrences of  $n$ -grams is anyway meant to be done in the encrypted message, and results are different; we used the original message for a friendlier illustration of the technique.

The following generator function adapts the previous code to, for an arbitrary text passed as argument, not collect in a list the relevant factors of the distances between all successive occurrences of all  $n$ -grams in the text (for the chosen values of  $n$ ), but have a mechanism to return them on demand:

```

In [20]: def all_collected_factors(text):
          for n in n_gram_range:
              for i in range(len(text) - 2 * n + 1):
                  j = text.find(text[i: i + n], i + n)
                  if j != -1:
                      yield from factors_from_successive_n_grams(j - i, n)

          print(list(all_collected_factors(encrypted_message)))

```

对于作为参数传递的任意文本，以下生成器函数使先前的代码适应，不在列表中收集文本中所有n-gram的所有连续出现之间的距离的相关因子（对于所选择的n值），但有一个机制可以按需返回它们

```
[4, 5, 10, 13, 4, 5, 10, 13, 4, 5, 10, 13, 4, 8, 13, 3, 6, 13, 3, 6, 13, 3, ...
...6, 13, 3, 6, 13, 3, 6, 13, 5, 10, 13, 3, 6, 13, 3, 13, 4, 5, 10, 13, ...]
```

可以说，一个因子越频繁，就越有可能等于用于加密消息的密钥长度。因此，我们希望从最频繁到最不频繁地排名所有已经为所有连续的n-gram对收集的因子。第一步是计算每个因素。由于集合模块中的Counter类，可以直接从集合中获取a（嵌入的对象类型）字典，其键和值是集合的成员以及它们在集合中出现的次数，

```
...4, 5, 10, 13, 6, 13, 6, 13, 6, 13, 6, 13, 5, 10, 13, 6, 13, 6, 13, 6, 13]
```

#### 第一种

Arguably, the more frequent a factor is, the more likely it is to be equal to the length of the key that has been used to encrypt the message. We therefore want to rank, from most frequent to least frequent, all factors that have been collected for all consecutive pairs of  $n$ -grams. The first step is to count each factor. Thanks to the Counter class from the collections module, it is straightforward to get from a collection a (type of object that embeds a) dictionary whose keys and values are the members of the collection and the number of times they occur in the collection, respectively:

```
In [21]: Counter([4, 5, 10, 13, 4, 5, 10, 13, 4, 5, 10, 13, 4, 8, 13])
```

排序，并且输出次数

```
Out[21]: Counter({4: 4, 5: 3, 10: 3, 13: 4, 8: 1})
```

#### 第二种

Factors should be ranked from the most frequent ones to the least frequent ones, but what to do when two factors have equal counts? E.g., what if both 8 and 16 occur 6,811 times? (This is the case when encrypting the text in carroll.txt with the key '0123456789ABCDEF'.) It seems probable that if the key length was 8 rather than 16, then we would have more  $n$ -grams that reveal vertical alignments over 8 columns than  $n$ -grams that reveal vertical alignments over 16 columns (actually, not having a single extra factor of 8 strongly suggests that the key length is 16 indeed). So we decide to rank the divisors from most frequent ones to least frequent ones, and for a given count, from largest ones to smallest ones. So the key of the sorting method needs to take into account first the values of the dictionary embedded in the counter, and then the keys. We can retrieve from a dictionary (be it embedded in a counter) keys only, values only, or both keys and values, in the form of list-like objects:

```
In [22]: D = {4: 4, 5: 3, 10: 3, 13: 4, 8: 1}
         D.keys()
         D.values()
         D.items()
```

```
Out[22]: dict_keys([4, 5, 10, 13, 8])
```

```
Out[22]: dict_values([4, 3, 3, 4, 1])
```

```
Out[22]: dict_items([(4, 4), (5, 3), (10, 3), (13, 4), (8, 1)])
```

#### 第三种

The third way of sorting the items of a dictionary offers the solution to ranking properly our collected factors:

排序字典项目的第三种方法提供了正确排列我们收集的因素的解决方案：

```
In [23]: D = {4: 4, 5: 3, 10: 3, 13: 4, 8: 1}
```

```
sorted(D.items(), reverse = True)
[x[0] for x in sorted(D.items(), reverse = True)]

sorted(D.items(), key = lambda x: x[1], reverse = True)
[x[0] for x in sorted(D.items(), key = lambda x: x[1], reverse = True)]

sorted(D.items(), key = lambda x: (x[1], x[0]), reverse = True)
[x[0] for x in sorted(D.items(),
```

Lambda表达式：“Lambda 表达式” (lambda expression) 是一个匿名函数，Lambda表达式基于数学中的 演算得名，直接对应于其中的lambda抽象 (lambda abstraction)，是一个匿名函数，即没有函数名的函数。Lambda表达式可以表示闭包（注意和数学传统意义上的不同）。Lambda x : 2\*x + 1 冒号的前面是原函数的参数，冒号的后面是函数的返回值。不用命名函数，直接使用。



iterable -- 可迭代对象。  
cmp -- 比较的函数，这个具有两个参数，参数的值都是从可迭代对象中取出，此函数必须遵守的规则为，大于则返回1，小于则返回-1，等于则返回0。  
key -- 主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是取自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。  
reverse -- 排序规则，reverse = True 降序，reverse = False 升序（默认）

```

        key = lambda x: (x[1], x[0]),
        reverse = True
    )

]

```

Out[23]: [(13, 4), (10, 3), (8, 1), (5, 3), (4, 4)]

Out[23]: [13, 10, 8, 5, 4]

Out[23]: [(4, 4), (13, 4), (5, 3), (10, 3), (8, 1)]

Out[23]: [4, 13, 5, 10, 8]

Out[23]: [(13, 4), (4, 4), (10, 3), (5, 3), (8, 1)]

Out[23]: [13, 4, 10, 5, 8]

我们也可以使用运算符模块中的itemgetter()，而不是使用lambda表达式：

Rather than using a lambda expression, we can also use itemgetter() from the operator module:

```

In [24]: L = ['A', 'B', 'C']
        itemgetter(0)(L)
        itemgetter(0, 2)(L)
        itemgetter(2, 0, 1)(L)

```

```

D = {4: 4, 5: 3, 10: 3, 13: 4, 8: 1}
sorted(D.items(), key = itemgetter(1, 0), reverse = True)
[x[0] for x in sorted(D.items(), key = itemgetter(1, 0), reverse = True)]

```

Out[24]: 'A'

Out[24]: ('A', 'C')

Out[24]: ('C', 'A', 'B')

Out[24]: [(13, 4), (4, 4), (10, 3), (5, 3), (8, 1)]

Out[24]: [13, 4, 10, 5, 8]

operator模块提供的itemgetter函数用于获取对象的哪些维的数据，参数为一些序号（即需要获取的数据在对象中的序号），下面看例子。

```
a = [1,2,3]
```

```
>>> b=operator.itemgetter(1)           //定义函数b，获取对象的第1个域的值
```

```
>>> b(a)
```

```
2
```

```
>>> b=operator.itemgetter(1,0)         //定义函数b，获取对象的第1个域和第0个的值
```

```
>>> b(a) (2, 1)
```

要注意，operator.itemgetter函数获取的不是值，而是定义了一个函数，通过该函数作用到对象上才能获取值。

Having with all\_collected\_factors() a mechanism to generate some key lengths from most promising to least promising, we still would like to consider all possible key lengths between 1 and the value of max\_key\_length, starting with 1 (in case the message has actually been encrypted with the simple Caesar code), then those given by all\_collected\_factors(), and finally all others, from shortest to longest. This is the purpose of the following generator function:

```

In [25]: def key_lengths_from_most_to_least_promising(text):
        yield 1
        key_lengths = [x[0] for x in

```

```

sorted(Counter(all_collected_factors(text)).items(),
        key = itemgetter(1, 0),
        reverse = True

```

使用all\_collected\_factors()生成从最有前途到最有希望的一些密钥长度的机制，我们仍然希望考虑所有可能的密钥长度在1和max\_key\_length的值之间，从1开始（如果消息实际上已经加密了简单的Caesar代码），然后是all\_collected\_factors()给出的那些，最后是所有的其他的，从最短到最长。这是以下生成器功能的目的：

```

        )
    ]
    yield from key_lengths
    yield from (i for i in range(2, max_key_length + 1)
                if i not in key_lengths
                )

    list(key_lengths_from_most_to_least_promising(encrypted_message))

```

```
Out[25]: [1, 13, 6, 10, 5, 3, 4, 8, 2, 7, 9, 11, 12, 14, 15, 16]
```

For all  $n < 13$ , the  $n$ -th character of key encrypts the  $n$ -th column of message, yielding the  $n$ -th column of encrypted\_message, which can be decrypted back to the  $n$ -th column of message:

```
In [26]: for n in range(13):
          decrypt(key[n], encrypted_message[n: : 13])
```

```
Out[26]: 'Ai fh do epbeg ooneoAurn'
```

```
Out[26]: 'lnv et to eor,nrnd klt s'
```

```
Out[26]: 'inesrhohnsdo o s u,i c?'
```

```
Out[26]: "ciri efich kwb c ws'cpo'"
```

```
Out[26]: 'enyts neei aupoihe ein'
```

```
Out[26]: ' g tibhg nhstinna t,cv'
```

```
Out[26]: 'w tisaa ohte cv toh te'
```

```
Out[26]: "atintnvtraorritei fo'ur"
```

```
Out[26]: 'sorgekio d eturti uwrs'
```

```
Out[26]: ' e r,n t tsa rs,sagiea'
```

```
Out[26]: 'bgdb gdwphidhea htst'
```

```
Out[26]: "ee yoa oieesiast'tbth i"
```

```
Out[26]: 'gto nnn:ce tnd iaho ooo'
```

与解密第 $n$ 列encrypted\_message时获得的结果相比较，而不是使用正确的字符（键中的第 $n$ 个字符），而是使用任意字符：

Compare with the result obtained when decrypting the  $n$ -th column of encrypted\_message not with the right character (the  $n$ -th character in key), but with an arbitrary character:

```
In [27]: for n in range(13):
          decrypt('m+2I|b~!AxZ=I', encrypted_message[n: : 13])
```

```
Out[27]: '(10q;F+hgAj@pN::z[W@tNNh'
```

```

Out[27]: '9WDYusb-\t*YP{ZWz]t~y%3*- '
Out[27]: "g'w4RqM+7f(Z)}(f4nD0,;\nF"
Out[27]: '4VD\\hhx&^\tLP23Ao#Qv]Z35l '
Out[27]: "]"Jt<v=\tT+.PbY5Lj'k0\tK/["
Out[27]: '{Xd3GiD(/<:Q2pZD y4Pt|g'
Out[27]: 'DPU3"sHZhd Y<4%W<#G0Zma '
Out[27]: "?Pd,w9wZ\*VF:{P9'01p3__"
Out[27]: 'K`$cYN!21k,SaL\t$p#q-avz '
Out[27]: '?wmYH+BG3*.T,?!ADI9u""\['
Out[27]: 'h(D}veM/ok>" n&A;veN}ko '
Out[27]: 'q,to#xz~gf [<8m^ZW(y&}}j '
Out[27]: 'Uj[f|-` :CH\tbFR[;s\\."oOR'

```

The columns derived from the correct key contain more letters. But is that all? Is the distribution of letters in a given column good, in some sense? A column consisting of nothing but q's, z's and j's would not appear as natural.

“etaoinshrdlcumwfgypbvkjxqz” is all lowercase letters ordered in decreasing frequency of use in English: “e” is most common, then comes “t”, then “a”... etaoins得分的概念将把这个想法变成一个精确的衡量标准。

A good distribution of letters in a given column should be relatively consistent with the ordering of letters given by the previous sequence. The notion of etaoins score will turn this idea into a precise measure.

Consider the first column, “Ai fh do epbeg ooneoAurn”. Let us convert all letters to lowercase and get a count of the number of occurrences of the letters that occur in the resulting string:

考虑第一栏 “Ai fh do epbeg ooneoAurn”。让我们将所有字母转换为小写，并计算结果字符串中出现的字母出现次数：

```

In [28]: column = Counter(c.lower() for c in 'Ai fh do epbeg ooneoAurn'
        列                                     if c.isalpha() Python isalpha() 方法检测字符串是否只由字母组成。isalpha
        )                                     ()方法语法: str.isalpha()

        print(column)

```

```

Counter({'o': 4, 'e': 3, 'a': 2, 'n': 2, 'i': 1, 'f': 1, 'h': 1, 'd': 1, ...
        原例句当中，不同字母出现的 次数 ... 'p': 1, 'b': 1, 'g': 1, 'u': 1, 'r': 1})

```

Let us order the letters by decreasing frequency in column. When two letters  $\alpha$  and  $\beta$  share the same count, let us be “pessimistic” about column’s quality and rank  $\alpha$  before  $\beta$  if  $\alpha$  occurs after  $\beta$  in “etaoinshrdlcumwfgypbvkjxqz”. So “o” comes first with 4 occurrences, then comes “e” with 3 occurrences. Both “a” and “n” have 2 occurrences, but “a” is more frequent than “n” in English (they occur at index 2 and 5 in ‘etaoinshrdlcumwfgypbvkjxqz’, respectively), so we rank “n” before “a”. All other letters occur only once in column, with “p” least frequent in English, so ranked next, and “i” most frequent in English, so ranked last:

让我们通过降低列中的频率来对字母进行排序。当两个字母 和 共享相同的数时，让我们对 的质量和 之前的等级 “悲观”，如果 出现在 之后

“etaoinshrdlcumwfgypbvkjxqz”。所以 “o” 首先出现4次，然后出现 “e” 出现3次。 “a” 和 “n” 都有2次出现，但 “a” 比英语中的 “n” 更频繁（它们分别出现在 ‘etaoinshrdlcumwfgypbvkjxqz’ 的索引2和5），因此我们在 “a” 之前排名 “n”。所有其他字母在列中只出现一次，“p” 在英语中最不常见，因此排名第二，“i” 最常用英语，因此排名最后：

```
In [29]: ranked_letters_in_column =\
        ''.join(sorted(column,
                        key = lambda x: (column[x],
                                         'etaoinshrdlcumwfgypbvjkxqz'.index(x)),
                        reverse = True)
        )

        ranked_letters_in_column
```

Out[29]: 'oenabpgfudrhi'

之前的代码可以通过从'etaoinshrdlcumwfgypbvjkxqz'定义一个字典来改进，这样可以更有效地检索小写字母的字符串中的索引，类似于从字符串定义字典移位的方式

The previous code can be improved by defining from 'etaoinshrdlcumwfgypbvjkxqz' a dictionary thanks to which the index in that string of a lowercase letter can be more effectively retrieved, similarly to the way the dictionary shifts has been defined from the string characters:

```
In [30]: ranked_letters_in_column =\
        ''.join(sorted(column,
                        key = lambda x: (column[x],
                                         'etaoinshrdlcumwfgypbvjkxqz'.index(x)),
                        reverse = True)
        )

        ranked_letters_in_column
```

Out[30]: 'oenabpgfudrhi'

The etao in score of ranked\_letters\_in\_column is actually a function of both ranked\_letters\_in\_column and a nonzero natural number  $l$  at most equal to the length of this string: for such a number  $l$ , it is equal to the number of elements common to the initial segments of 'oenabpgfudrhi' and 'etaoinshrdlcumwfgypbvjkxqz' of length  $l$ . Let us display for  $l$  at most equal to 10,  $l$  itself, then those initial segments of length  $l$ , then the letters both segments have in common, then the number of those letters, so the corresponding etao in score:

```
In [31]: for i in range(1, 11):
        s_1 = ranked_letters_in_column[:i]
        s_2 = 'etaoinshrdlcumwfgypbvjkxqz'[:i]
        s = ''.join(set(s_1) & set(s_2))
        print(f'{i:2} {s_1:10} {s_2:10} {s:9} {len(s)}')
```

这里的功能实现了每行元素递增的结果

1 o	e		0	{i : 2 }就是依次打印从1 到10的数字 ,
2 oe	et	e	1	{s_1 : 10} 就是依次打印从第一行到第10行的内容
3 oen	eta	e	1	
4 oena	etao	oea	3	
5 oenab	etaoi	oae	3	
6 oenabp	etaoin	noae	4	

7	oenabpg	etaoins	noae	4
8	oenabpgf	etaoinsh	noae	4
9	oenabpgfu	etaoinshr	noae	4
10	oenabpgfud	etaoinshrd	nedo	5

Converting strings to sets and taking their intersections is not the most efficient approach here. We can instead look whether each the  $l$  first letters in `ranked_letters_in_column` is one of the first  $l$  characters in `'etaoinshrdlcumwfgypbvjkxqz'`. To get the etaoins score of `ranked_letters_in_column` for  $l = 10$ , we need to see how many times the `if` statement below is executed:

```
In [32]: for i in range(10):
          s = ranked_letters_in_column[i]
          print(f'{{s}}', end = ' ')
          if etaoins[s] < 10:
              print('occurs', end = ' ')
          else:
              print('does not occur', end = ' ')
          print(f'in {{'etaoinshrdlcumwfgypbvjkxqz'}}[: 10]})"
```

将字符串转换为集合并将其交叉点转换为此处不是最有效的方法。我们可以改为查看`ranking_letters_in_column`中的每个 $l$ 个首字母是否是前 $l$ 个字符之一。在`'etaoinshrdlcumwfgypbvjkxqz'`。要获得 $l = 10$ 的`ranking_letters_in_column`的etaoins得分，我们需要查看下面的`if`语句执行次数：

```
o occurs in etaoinsrd
e occurs in etaoinsrd
n occurs in etaoinsrd
a occurs in etaoinsrd
b does not occur in etaoinsrd
p does not occur in etaoinsrd
g does not occur in etaoinsrd
f does not occur in etaoinsrd
u does not occur in etaoinsrd
d occurs in etaoinsrd
```

This is easily done with the `sum()` function:

```
In [33]: sum(1 for i in range(10) if etaoins[ranked_letters_in_column[i]] < 10)

Out[33]: 5
```

Putting it all together in a function:

```
In [34]: def etaoins_score(text, length):
          letter_counts = Counter(c.lower() for c in text if c.isalpha())
          ranked_letters = sorted(letter_counts,
                                   key = lambda x: (letter_counts[x], etaoins[x]),
                                   reverse = True
                                   )
          return sum(1 for i in range(min(length, len(ranked_letters)))
                      if etaoins[ranked_letters[i]] < length
                      )

          etaoins_score('Ai fh do epbeg ooneoAurn', 10)
```

```
Out[34]: 5
```

As part of the heuristic, we have to chose a value for the second argument `length` of `etaoin_score()`, which clearly should be neither too small not too large:

作为启发式的一部分，我们必须为`etaoin_score()`的第二个参数长度选择一个值，它显然不应该太小而不能太大：

```
In [35]: etaoin_length = 6
```

The 8th column of message displayed over 13 columns has be encrypted with “E”, the 8th character of “Take it Easy!”. Deciphering this column with “E” and “e” shows similar results, with at many positions, the same letter, but lowercase for one, and uppercase for the other. The etaoin score is the same for both:

```
In [36]: decrypt('E', encrypted_message[8: : 13])
decrypt('e', encrypted_message[8: : 13])

etaoin_score(decrypt('E', encrypted_message[8: : 13]), etaoin_length)
etaoin_score(decrypt('e', encrypted_message[8: : 13]), etaoin_length)
```

显示在13列上的第8列消息用“E”加密，即第8个字符“别紧张！”。用“E”和“e”对该列进行解密显示了类似的结果，在许多位置，相同的字母，但一个是小写，另一个是大写。两者的etaoin得分相同

```
Out[36]: 'sorgekio d eturti uwrs'
```

```
Out[36]: 'SORGEKIO nDnnETURTI nUWRS'
```

```
Out[36]: 3
```

```
Out[36]: 3
```

If we assume that the original message is “usual” text, with mostly lowercase letters, then we should consider “E” to be more promising than “e”. More generally, when deciphering a given column of the encrypted message with two letters, if the ataoin scores differ, then the letter with the highest score is arguably more likely to be correct; but if both ataoin scores are the same, then the letter that yields the highest proportion of lowercase letters over all letters is arguably more likely to be correct.

The following code fragment tentatively deciphers the 8th column of `encrypted_message` with each of the 97 members of `characters` (1-character subkeys). It computes the 97 etaoin scores as well as the 97 proportions of lowercase letters over all letters plus 1 (to prevent division by 0). It then ranks each of the 97 1-character subkeys in decreasing value of etaoin score, and for a given etaoin score, in decreasing proportion of lower letters. Eventually, it displays the top 10 results, together with the corresponding tentative deciphering of the 8th column:

```
In [37]: scores = []
for subkey in characters:
    decrypted_column = decrypt(subkey, encrypted_message[8: : 13])
    nb_of_lowercase_letters = 0
    nb_of_letters = 1
    letters = (c for c in decrypted_column if c.isalpha())
    for c in letters:
        nb_of_letters += 1
        if c.islower():
            nb_of_lowercase_letters += 1
    scores.append((subkey,
                  etaoin_score(decrypted_column, etaoin_length),
```

```

        nb_of_lowercase_letters / nb_of_letters
    )
)
scores.sort(key = itemgetter(1, 2), reverse = True)

for scored_subkey in scores[: 10]:
    print(scored_subkey, decrypt(scored_subkey[0],
                                encrypted_message[8: : 13]
                                )
    )

('E', 3, 0.95) sorgekio d eturti uwrs
('I', 3, 0.9473684210526315) okncagek{9{{apqnpe{qsno
('K', 3, 0.9411764705882353) mila8eci_7__8nolnc_oqlm
('S', 3, 0.9230769230769231) ead2064a=
==0fgdf4=gide
('e', 3, 0.16666666666666666) SORGEKIOndnnETURTIInUWRS
('J', 2, 0.9411764705882353) njmb9fdj`8``9opmod`prmn
('O', 2, 0.9285714285714286) ieh64a8e[3[[4jkhj8[kmhi
('W', 2, 0.875) a69 ~206/}/~/~bc9b0/ce9a
('r', 2, 0.4583333333333333) FBETrxvBaqaarGHEGvaHJEF
('n', 2, 0.4166666666666667) JFIxvBzFeueevKLIKzeLNIJ

```

给定一个暂定密钥长度 $l$ ，希望是隐藏密钥的实际长度，有 $l$ 个1字符要发现的子键，一列显示在多列上显示的消息的每一列。 有人想

对于每一列，只考虑其最有前途的1字符子密钥。 限制自己对于每列，其 $k$ 个最有希望的子密钥，总共 $k$ 升

然后可以从 $l$ 组装完整的密钥1个字符的子键。 所以 $k$ 必须很小。 让我们仍然把它作为整体启发式的另一个参数并将其设置为默认值2

Given a tentative key length  $l$ , hoped to be the actual length of the hidden key, there are  $l$  1-character subkeys to discover, one for each column of the message displayed over  $l$  many columns. One would like to consider, for each column, nothing but its most promising 1-character subkeys. Restricting ourselves to, for each column, its  $k$  most promising subkeys, a total of  $k^l$  full keys can then be assembled from the  $l$  1-character subkeys. So  $k$  has to be small. Let us still make it another parameter of the overall heuristics and set it to a default of 2:

```
In [38]: nb_of_options_for_subkey = 2
```

To easily generate all possible keys from all choices of 1-character subkeys, the product class from the itertools module is useful: 要从所有选择的1个字符的子键轻松生成所有可能的键，产品类来自 itertools模块很有用

```
In [39]: # The cartesian product {0, 1} x {'A', 'B'} x {'c', 'd'}
list(product(range(2), ['A', 'B'], (x for x in 'cd')))
# The cartesian product ({0, 1} x {'A', 'B'})^2
list(product(range(2), ['A', 'B'], repeat = 2))

```

```
Out[39]: [(0, 'A', 'c'),
(0, 'A', 'd'),
(0, 'B', 'c'),
(0, 'B', 'd'),
(1, 'A', 'c'),
(1, 'A', 'd'),
(1, 'B', 'c'),
(1, 'B', 'd')]

```



```
Out[39]: [(0, 'A', 0, 'A'),
          (0, 'A', 0, 'B'),
          (0, 'A', 1, 'A'),
          (0, 'A', 1, 'B'),
          (0, 'B', 0, 'A'),
          (0, 'B', 0, 'B'),
          (0, 'B', 1, 'A'),
          (0, 'B', 1, 'B'),
          (1, 'A', 0, 'A'),
          (1, 'A', 0, 'B'),
          (1, 'A', 1, 'A'),
          (1, 'A', 1, 'B'),
          (1, 'B', 0, 'A'),
          (1, 'B', 0, 'B'),
          (1, 'B', 1, 'A'),
          (1, 'B', 1, 'B')]
```

以下代码片段作为参数传递给list()生成所有的生成器表达式  
长度为3的键，来自三个1个字符子键中每个子键的两个可能选项的三个，“0”或“1”  
对于第一个子键，第二个子键为“A”或“B”，第三个子键为“c”或“d”

The following code fragment passes as an argument to list() a generator expression to generate all keys of length 3 from a triple of 2 possible options for each of three 1-character subkeys, with '0' or '1' for the first subkey, 'A' or 'B' for the second subkey, 'c' or 'd' for the third subkey:

```
In [40]: subkeys = ('0', '1'), ('A', 'B'), ('c', 'd')

list(key for key in (''.join(subkey) for subkey in product(*subkeys)))
```

```
Out[40]: ['0Ac', '0Ad', '0Bc', '0Bd', '1Ac', '1Ad', '1Bc', '1Bd']
```

Having assembled a complete key from 1-character subkeys, the encrypted message can be tentatively decrypted as a whole. Whereas 1-character subkeys look more or less promising depending on the distribution of letters in a tentatively decrypted column, whole keys look more or less promising depending on whether the tentatively decrypted message contains enough letters amongst all characters, and enough English words amongst all words, here defining a word as a longest sequence of consecutive letters. So any longest sequence of consecutive nonletters is a word separator; that is something not for the split() method of the str class, but for the split() function of the re module, using the syntax of regular expressions:

从1个字符的子密钥组装完整的密钥后，加密的消息可以是暂时的

```
In [41]: # Using any of the characters
# between the square brackets as a separator
split('[+$(^?=_)]', '0+$abc(^?DEF=_')
# Using any longest sequence of the character
# between the square brackets as a separator
split('[+$(^?=_)]+', '0+$abc(^?DEF=_')
# Using any lowercase letter as a separator
split('[a-z]', '0+$abc(^?DEF=_')
# Using anything but a lowercase letter as a separator
split('[^a-z]', '0+$abc(^?DEF=_')
# Using any longest sequence of letters as a separator
split('[a-zA-Z]+', '0+$abc(^?DEF=_')
# Using any longest sequence of anything but letters
```

解密为一个整体。虽然1个字符的子键看起来或多或少看起来很有希望，这取决于暂时解密列中字母的分布，但是整个键看起来或多或少看起来很有希望，具体取决于暂时解密的消息是否包含所有字符中的足够字母，并且足够所有单词中的英语单词，这里将单词定义为连续字母的最长序列。所以任何最长的连续非连续序列都是一个字符分隔符；那不是分裂()的东西。str类的方法，但是对于re模块的split()函数，使用常规expressions的语法：

```

# or digits as a separator
split('^a-zA-Z0-9', '0+$abc(^?DEF=_')

Out[41]: ['0', '', 'abc', '', '', 'DEF', '', '']

Out[41]: ['0', 'abc', 'DEF', '']

Out[41]: ['0+$', '', '', ' (^?DEF=_']

Out[41]: ['', '', '', 'abc', '', '', '', '', '', '', '', '']

Out[41]: ['0+$', ' (^?', '=']

Out[41]: ['0', '', 'abc', '', '', 'DEF', '', '']

```

So '^a-zA-Z)+' is the regular expression we need:

```

In [42]: print(split('^a-zA-Z+', message))

['Alice', 'was', 'beginning', 'to', 'get', 'very', 'tired', 'of', 'sitting',...
... 'by', 'her', 'sister', 'on', 'the', 'bank', 'and', 'of', 'having',...
... 'nothing', 'to', 'do', 'once', 'or', 'twice', 'she', 'had',...
... 'peeped', 'into', 'the', 'book', 'her', 'sister', 'was', 'reading',...
... 'but', 'it', 'had', 'no', 'pictures', 'or', 'conversations', 'in',...
... 'it', 'and', 'what', 'is', 'the', 'use', 'of', 'a', 'book',...
... 'thought', 'Alice', 'without', 'pictures', 'or', 'conversations', '']

```

To complete the heuristics, we make use of two last parameters. First a desired fraction of letters over all letters in the tentatively deciphered message, set to a default of 70%. Second a desired fraction of English words over all words in the tentatively deciphered message, set to a default of 50%:

```

In [43]: fraction_of_letters = 0.7
         fraction_of_words = .5

```

为了完成启发式算法，我们使用了两个最后的参数。首先是所需的一小部分字母暂时解密的消息中的所有字母，默认设置为70%。第二个是英语的理想分数在暂时解密的消息中的所有单词上的单词，设置为默认值50%

With dictionary denoting a list of all lowercase English words (first read from a file in practice), the following function returns True or False depending on whether the tentatively decrypted message passed as first argument looks right or not. In case the function returns True, the tentatively decrypted message will be displayed to the user to accept, or to reject, in which case the program will resume its search for the correct key:

```

In [44]: def looks_like_English(text, dictionary):
         if sum(1 for c in text if c.isalpha()) / len(text) < \
            fraction_of_letters:
             return False
         possible_words = split('^a-zA-Z+', text)
         nb_of_words = sum(1 for w in possible_words if w in dictionary)
         return nb_of_words / len(possible_words) > fraction_of_words

```

We are ready to put everything together.

使用字典表示所有小写英文单词的列表（在实践中首先从文件中读取），  
 以下函数返回True或False，具体取决于暂时解密的消息  
 作为第一个参数传递看起来正确与否。如果函数返回True，则暂时解密  
 消息将显示给用户接受或拒绝，在这种情况下程序将恢复它  
 搜索正确的密钥

函数encrypt\_file() 用于加密文件中包含的文本，其名称为作为第二个参数提供，第一个参数是加密密钥。默认情况下，该功能将加密的消息显示在屏幕上，但第三个参数（默认设置为None）可以是更改为文件名，然后将加密的邮件写入此文件

- A function encrypt\_file() is designed to encrypt the text contained in a file, whose name is provided as second argument, the first argument being the encryption key. By default, the function displays the encrypted message to the screen, but the third argument, set by default to None, can be changed to the name of a file and the encrypted message will then be written to this file instead.
  - To read from a file, the second argument should be the name of an existing file. Otherwise, open() will raise a FileNotFoundError exception, which the codes catches in an except statement.
  - To write to a new file or overwrite the contents of an existing file, the open() function is given 'w' as second argument.
  - encrypt\_file() just calls the encrypt() function to perform the encryption. That function expects the message to encrypt to be given as a single string, with as many embedded '\n' characters in the string as there are lines in the message. To read the whole contents of a file as a single string, we use the read() method of the object returned by the open() function.

函数decrypt\_file() 用于解密文件中包含的文本，其名称为作为第二个参数提供，第一个参数是加密密钥。默认情况下，该功能将解密的消息显示在屏幕上，但第三个参数（默认设置为None）可以

- A function decrypt\_file() is designed to decrypt the text contained in a file, whose name is provided as second argument, the first argument being the encryption key. By default, the function displays the decrypted message to the screen, but the third argument, set by default to None, can be changed to the name of a file and the decrypted message will then be written to this file instead, provided that it does not exist.
  - To write to a new file, the open() function is given 'x' as second argument; in case the file exists, then open() will not overwrite it but instead, raise a FileExistsError exception.
  - decrypt\_file() just calls the decrypt() function to perform the decryption, also using read() to get the message to decrypt as a single string.

更改为文件的名称，然后将解密的消息写入此文件，只要它不存在

- A function break\_key\_for\_file() is designed to try and break the key of an encrypted message stored in a file whose name is passed as argument to break\_key\_for\_file(). Here again, read() is used to get the text to decrypt (from a key that first, has to be discovered) as a single string, that is passed an an argument to another function, break\_key().
  - break\_key() opens a file meant to contain a list of English words, one per line. The strip() method is used to discard the new line character that ends each line of this file, the lower() method to convert each word to all lowercase.

函数break\_key\_for\_file() 旨在尝试破解加密消息的密钥。在尝试破解加密消息的密钥时，该文件的名称作为参数传递给break\_key\_for\_file()。再来，再读一个英文单词列表，每行一个。线条()

用于将文本解密（从首先必须被发现的密钥）解析为单个字符串

传递一个参数到另一个函数break\_key()，包含一个英文单词列表，每行一个。线条()

方法用于丢弃结束此文件各行的新行字符，lower()

将每个单词转换为全小写的方法。try:

- break\_key() 快速显示它出现的所有密钥

以暂时解密加密的密钥

信息;回车用于在一条线上快速显示给定长度的所有键，

每个这样的密钥都会覆盖前一个密钥。

- 当Looks\_like\_English() 返回True时，break\_key() 显示前200个字符

暂时解密的消息，并提示用户表达是否有消息

已成功解密，或者是否应恢复搜索其他密钥。如果

它耗尽了它所提供的所有密钥，其中没有一个是

由提议或验证的

user, break\_key() 承认失败

```
In [45]: def encrypt_file(key, filename, encrypted_filename = None):
    try:
        with open(filename) as file:
            if encrypted_filename:
                with open(encrypted_filename, 'w') as encrypted_file:
                    print(encrypt(key, file.read()), end = '',
                          file = encrypted_file)
    except:
        pass
```

```

        else:
            return encrypt(key, file.read())
    except FileNotFoundError:
        print(f'Could not open {filename}, giving up.')

def decrypt_file(key, filename, decrypted_filename = None):
    try:
        with open(filename) as file:
            if decrypted_filename:
                try:
                    with open(decrypted_filename, 'x') as decrypted_file:
                        print(decrypt(key, file.read()),
                              file = decrypted_file
                        )
                except FileExistsError:
                    print(f'{decrypted_filename} exists, giving up.')
            else:
                return decrypt(key, file.read())
    except FileNotFoundError:
        print(f'Could not open {filename}, giving up.')

def break_key_for_file(filename):
    try:
        with open(filename) as file:
            break_key(file.read())
    except FileNotFoundError:
        print(f'Could not open {filename}, giving up.')

def break_key(text):
    try:
        with open(dictionary_file) as file:
            dictionary = {w.strip().lower() for w in file}
    except FileNotFoundError:
        print(f'Could not open the file {dictionary_file}, giving up.')
        return
    for key_length in key_lengths_from_most_to_least_promising(text):
        print(f'\nNow working with keys of length {key_length}')
        subkeys = []
        for n in range(key_length):
            scores = []
            for subkey in characters:
                decrypted_column = decrypt(subkey, text[n: : key_length])
                nb_of_lowercase_letters = 0
                nb_of_letters = 1
                letters = (c for c in decrypted_column if c.isalpha())
                for c in letters:
                    nb_of_letters += 1
                    if c.islower():

```

```

        nb_of_lowercase_letters += 1
    scores.append((subkey,
                   etaoi_score(decrypted_column, etaoi_length),
                   nb_of_lowercase_letters / nb_of_letters
                   )
                 )
    scores.sort(key = itemgetter(1, 2), reverse = True)
    subkeys.append(x[0] for x in scores[: nb_of_options_for_subkey])
for key in (''.join(subkey) for subkey in product(*subkeys)):
    print('\r', key, end = '')
    decrypted_text = decrypt(key, text)
    if looks_like_English(decrypted_text, dictionary):
        print('\nWhat about this?\n')
        print(decrypted_text[: 200], '...', sep = '')
        print()
        print('Enter Y[es] if happy, otherwise press any key '
              'and I\'ll keep working.')
        yes_or_no = input('> ')
        if yes_or_no in {'YES', 'Yes', 'yes', 'Y', 'y'}:
            print(f'The key is: "{key}"')
            return
    print('Sorry, I did my best...')

```

```

In [46]: encrypt_file('Take it Easy!', 'carroll.txt', 'encrypted_carroll.txt')
         break_key_for_file('encrypted_carroll.txt')

```

Now working with keys of length 1

1

Now working with keys of length 13

Take it Easy!

What about this?

Alice was beginning to get very tired of sitting by her sister on the bank, and of...

...having nothing to do: once or twice she had peeped into the book her sister...

...was reading, but it had no pictures or co...

Enter Y[es] if happy, otherwise press any key and I'll keep working.

> Yes

The key is: "Take it Easy!"