lab 2

June 14, 2023

1 2

1.1 - 1

-03

```
[ ]: print("Hello World!")
```

```
Hello World!
```

```
[ ]: alphabet = "                "
     letter_probabilities = [0.0792, 0.0171, 0.0433, 0.0174, 0.0305, 0.0841, 0.0105,␣
      ↪0.0175,
                             0.0683, 0.0112, 0.0336, 0.0500, 0.0326, 0.0672, 0.1108, 0.0281,
                             0.0445, 0.0533, 0.0618, 0.0280, 0.0019, 0.0089, 0.0036, 0.0147,
                             0.0081, 0.0037, 0.0002, 0.0196, 0.0192, 0.0038, 0.0061, 0.0213]
     alphabet_length = len(alphabet)

     key_2 = "  "
     key_3 = "   "
     key_4 = "    "
     key_5 = "     "
     key_6 = "      "
     key_18 = "          "

     f = open("encrypted_text.txt", "r", encoding='utf-8')
     text_to_be_decrypted = f.read()
     f.close()
```

1.1.1

```
[ ]: def preprocess_text(text):
         formatted_text = ""
         for char in text:
             if ' ' <= char and char <= ' ':
                 formatted_text += char
             elif ' ' <= char <= ' ':
                 formatted_text += char.lower()
```

```
        elif char == ' ' or char == ' ':
            formatted_text += ' '      # not pasting everything that is not␣
 ↪alphabet symbols
    return formatted_text
```

1.1.2

```
[ ]: def check_preprocessing(text):
         for char in text:
             if char not in alphabet:
                     print("Faulty letter is:", char)
                     return False
         return True

```

```
[ ]: f = open("text_to_be_encrypted.txt", "r", encoding='utf-8')
     text = f.read()
     f.close()

     processed_text = preprocess_text(text)

     print(check_preprocessing(processed_text)) # check if our text is ok

     print(check_preprocessing(text_to_be_decrypted)) # check if given text is ok


     f = open("text_to_be_encrypted_clear.txt", "w")
     f.write(processed_text)
     processed_text
```

```
True
Faulty letter is:

False
```

```
[ ]: '
```

'

, ,

[ ]:

```
text_to_be_decrypted = "".join(text_to_be_decrypted.split())

print(text_to_be_decrypted)

check_preprocessing(text_to_be_decrypted)
```

```
[ ]: True
```

1.1.3

[ ]:

```python
def Vigenere_encrypt(text, key):
    key_length = len(key)
    result = ""
    for i in range(len(text)):
        index = (alphabet.index(text[i]) + alphabet.index(key[i % key_length]))␣
 ↪% alphabet_length
        result += alphabet[index]
    return result

def Vigenere_decrypt(text, key):
    key_length = len(key)
    result = ""
    for i in range(len(text)):
        index = (alphabet.index(text[i]) - alphabet.index(key[i % key_length]))␣
 ↪% alphabet_length # just changing a + to - :)
        result += alphabet[index]
    return result
```

1.1.4 I_r

[ ]:

```python
def I_r(text):
    sum = 0
    for i in range(alphabet_length):
        N_t = text.count(alphabet[i])
        sum = N_t * (N_t - 1)
    return sum / (len(text) * len(text) - 1)
```

1.1.5

[ ]:

```python
print("I_encrypted: ", I_r(text_to_be_decrypted)) # I_r
print("I_open: ", I_r(processed_text)) # I_R
print("I_2: ", I_r(Vigenere_encrypt(processed_text, key_2)))
print("I_3: ", I_r(Vigenere_encrypt(processed_text, key_3)))
print("I_4: ", I_r(Vigenere_encrypt(processed_text, key_4)))
print("I_5: ", I_r(Vigenere_encrypt(processed_text, key_5)))
print("I_6: ", I_r(Vigenere_encrypt(processed_text, key_6)))
print("I_18: ", I_r(Vigenere_encrypt(processed_text, key_18)))
```

```
I_encrypted:  0.0008613379336495456
I_open:  0.0003377292935330818
I_2:  0.001593006313744536
I_3:  0.0014788044194245468
I_4:  0.0013929170443574472
I_5:  0.0018065187626292915
I_6:  0.0017927809773621486
I_18:  0.002151955897514549
```

I_r $I_0 = 1/m = 0.3125$,

1.1.6 :

[ ]:

```python
def blocking(text,r):
    res = []
    for i in range(r):
        block = text[i::r]
        res.append(block)
    return res
```

[ ]:

```python
def lenKey(text):
    n = len(text)
    D = []  # all the D_r's
    for r in range(6, 31):
        D.append(0) # new D_r
        for i in range(0, n-r):
            if(text[i] == text[i+r]): # Kronecker's symbol
                D[r-6] += 1
        print("D_", r, "=", D[r - 6])

    return D.index(max(D))+6

print(lenKey(text_to_be_decrypted))
```

```
D_ 6 = 199
D_ 7 = 229
D_ 8 = 208
D_ 9 = 231
```

```
D_ 10 = 203
D_ 11 = 176
D_ 12 = 341
D_ 13 = 189
D_ 14 = 188
D_ 15 = 214
D_ 16 = 197
D_ 17 = 199
D_ 18 = 195
D_ 19 = 238
D_ 20 = 219
D_ 21 = 211
D_ 22 = 209
D_ 23 = 196
D_ 24 = 354
D_ 25 = 187
D_ 26 = 203
D_ 27 = 218
D_ 28 = 212
D_ 29 = 231
D_ 30 = 212
24
```

D_{24} 2 D_{12}, , , , ,

1.1.7 :

[ ]:

```
def crack_key_1(text, r):
    key = ""
    most_probable_letter_index = letter_probabilities.
 ↪index(max(letter_probabilities)) # the index of the most probable letter in␣
 ↪the alphabet
    print("The most probable letter:", alphabet[most_probable_letter_index])
    blocks = blocking(text, r)
    for i in range(r):
        most_letter_occurances = 0
        most_occuring_letter_text_index = 0
        for t in range(alphabet_length):
            letters_in_text = blocks[i].count(alphabet[t])
            if(letters_in_text > most_letter_occurances):
                most_letter_occurances = letters_in_text
                most_occuring_letter_text_index = t
        k_i = (most_occuring_letter_text_index - most_probable_letter_index) %␣
 ↪alphabet_length
        key += alphabet[k_i]
    return key

key1 = crack_key_1(text_to_be_decrypted, 24)
```

```
print(key1)

Vigenere_decrypt(text_to_be_decrypted, key1)
```

```
The most probable letter:
```

```
[ ]: '
```

'

,

[ ]:

```
'''
This code don't work 'cause, as I understand, we need to compare the first,␣
 ↪second and third
probable letter to the most, the second most and the third most occurring␣
 ↪letter in the block
but who cares, the other method is better anyway

def crack_key_2(text, r):
    amount_of_keys = 3
    some_keys = []
    most_probable_letters = sorted(letter_probabilities, reverse=True)[:
 ↪amount_of_keys] # the most probable letters in the alphabet
    blocks = blocking(text, r)
    for l in range(amount_of_keys):
        temp_key = ""
        print("The most probable letter", l, ":", alphabet[letter_probabilities.
 ↪index(most_probable_letters[l])])
        for i in range(r):
            most_letter_occurances = 0
            most_occuring_letter_text_index = 0
            for t in range(alphabet_length):
                letters_in_text = blocks[i].count(alphabet[t])
                if(letters_in_text > most_letter_occurances):
                    most_letter_occurances = letters_in_text
                    most_occuring_letter_text_index = t
            k_i = (most_occuring_letter_text_index - letter_probabilities.
 ↪index(most_probable_letters[l])) % alphabet_length
            temp_key += alphabet[k_i]
        some_keys.append(temp_key)
    return some_keys

print(crack_key_2(text_to_be_decrypted, 24))'''
```

```
The most probable letter 0 :
The most probable letter 1 :
The most probable letter 2 :
['             ', '             ',
'             ']
```

1.1.8 ( )

[ ]:

```
def crack_key_Mi(text, r):
    key = ""
    blocks = blocking(text, r)
    for i in range(r):
        k_i = 0
        M_max = 0
```

```
        for g in range(alphabet_length):
            sum = 0
            for t in range(alphabet_length):
                sum += letter_probabilities[t] * blocks[i].count(alphabet[(t + g) % alphabet_length])
            if sum > M_max:
                M_max = sum
                k_i = g
        key += alphabet[k_i]
    return key

keyMi = crack_key_Mi(text_to_be_decrypted, 24)
print(keyMi)

print(Vigenere_decrypt(text_to_be_decrypted, keyMi))
```

, , 12 , 24. , : “ ”