Matveeva Lada Ivanna

Report Lab 1.

When the program starts, the **initialize()** function is called in which the alphabet is initialized in direct, reverse, and divided orders for encryption.

```
def initialize():
    direct_order = {}
    reverse_order = {}
    divide_order = {}

    for i in range(26):
        letter = chr(ord('a') + i)
        direct_order[letter] = i + 1
        reverse_order[letter] = 27 - (i + 1)
        if i < 13:
            divide_order[letter] = 13 - i
        else:
            divide_order[letter] = 39 - i

    return direct_order, reverse_order, divide_order
```

The program has two execution modes:

1. Input of an decrypted string.
2. Input of an encrypted message with a delimiter ",".

**Input of an decrypted string**

The entered string is transformed into lowercase for further use.

```
    user_input = user_input.lower()
```

The **encoding()** function is called.
Parameters:
*text* - the text entered by the user
*direct* - array for encoding the message
*letters* - dictionary with the order of encryption alphabet

```
def encoding(text, letters):
    message = []
    for i in text:
        direct_number = letters[i]
        message.append(direct_number)
    return message
```

In the function, a loop finds the corresponding ordinal number for each letter in the text. The function returns an array of the encoded message.

Then, the **decoding()** function is called.
Parameters:
*letters* - dictionary with the order of encryption for the alphabet
*message* - encoded message

```
def decoding(text, letters):
    decoded_message = "".join(find_key_by_value(letters, num) for num in text)
    return decoded_message
```

In this function, the function **find_key_by_value()** is called.
Parameters:
*dictionary* - dictionary with the direct order of encryption alphabet
*search_value* - number from the encoded message

```
def find_key_by_value(dictionary, search_value):
    for key, value in dictionary.items():
        if value == search_value:
            return key
    return None
```

Using a loop, we find the letter corresponding to the number. Then, we write out the encoded/decoded message.

```
print("[{:.3f}ms] {} message:\t\t{} {}".format( *args: duration(start_time, end_time) * 1000, function_message,
                                               input_message, output))
```

The execution time of the program is calculated as follows:

Before executing the program, we save the start time and at the end, we save the end time.

```
start_time_direct = time.perf_counter()
```

```
end_time_direct = time.perf_counter()
```

Then, in function **duration()** we calculate the difference between the end and the start.

```
def duration(start_time, end_time):
    return end_time - start_time
```

For the reverse and divided alphabet orders, the functions encoding and decoding are used. Each function uses the reverse and divided dictionary respectively.

**Input of an encrypted string**

When inputting a message in the form of an encoded message, the delimiter **","** is removed from the string, and the message is saved into an array.

```
user_input_array = [int(num) for num in user_input.split(',')]
```

Then, the same sequence of program execution occurs as when inputting a decoded message.

**Example of use**

When the program starts, an example of what data can be entered will first be listed.

```
Example:
     decoded message: matveeva
     encoded message: 13,1,20,22,5,5,22,1
```

Then we can write our data:

**Decoded message.** When the program finishes, we can see the three methods (direct, reverse, divided) of alphabet encoding listed. At the beginning of the line, the execution time will be written. Then, we see the message and its encoding.

```
Please enter surname: matveeva
[0.008ms] Direct  message:      matveeva [13, 1, 20, 22, 5, 5, 22, 1]
[0.003ms] Reverse message:      matveeva [14, 26, 7, 5, 22, 22, 5, 26]
[0.002ms] Divided message:      matveeva [1, 13, 20, 18, 9, 9, 18, 13]
```

**Encoded message:**

```
Please enter surname: 13,1,20,22,5,5,22,1
[0.026ms] Direct  message:      [13, 1, 20, 22, 5, 5, 22, 1] matveeva
[0.016ms] Reverse message:      [13, 1, 20, 22, 5, 5, 22, 1] nzgevvez
[0.031ms] Divided message:      [13, 1, 20, 22, 5, 5, 22, 1] amtriirm
```