Lada Ivanna Matveeva AIS ID 99198

The main.py contains the input and output for encoding and decoding in reversed order, divided order, and shifted order (Caesar cipher) message.

According to the assignments:

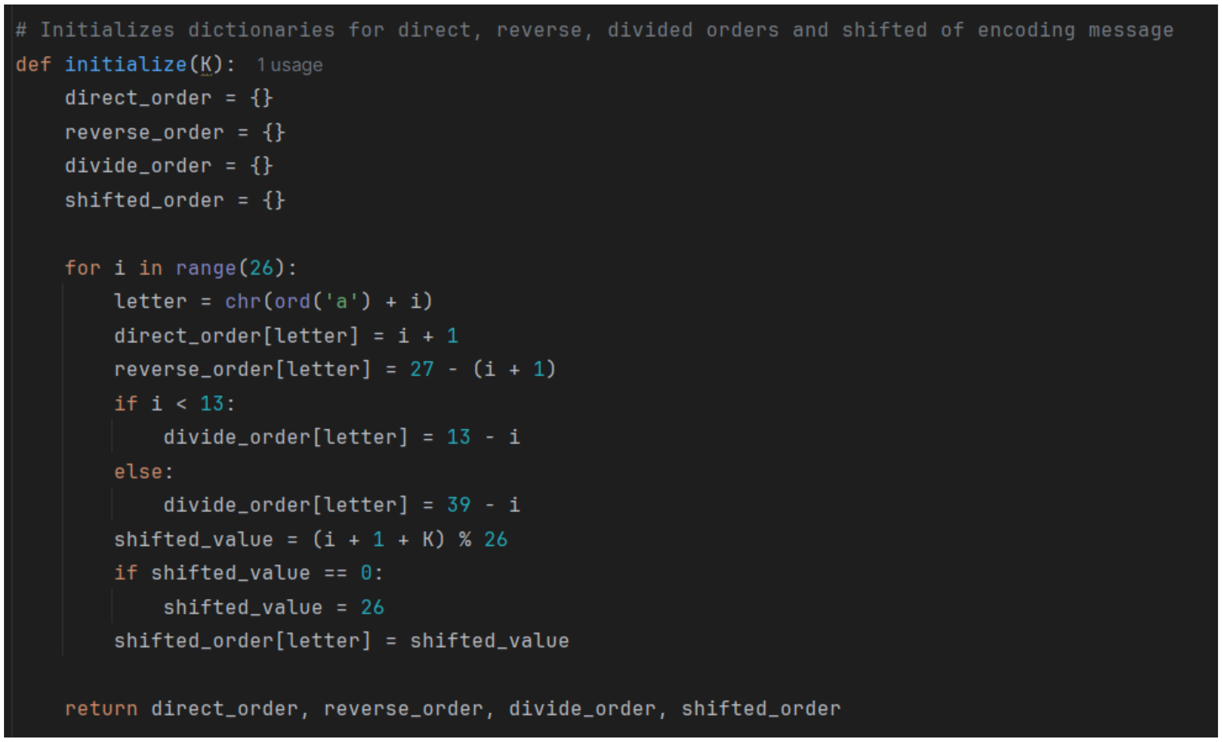
Lab work №1 task 1a – encoding for reverse order
Lab work №1 task 2a – encoding for divided order
Lab work №2 – encoding for Caesar cipher (shifted order)
Lab work №3 – decoding for reverse order, divided order, and Caesar cipher (shifted order).

In the program and the report, I will use the term "shifted order" to refer to the Caesar cipher.

Description of the execution of main.py

When the program starts, the **initialize()** function is called in which the alphabet is initialized in reverse, divided and shifted order.

Parameters:
K - The value of K is individual for every student as it was given at the start of the semester. The value of my K is 2.



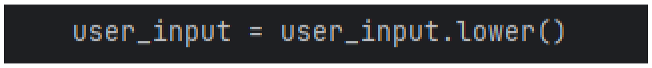
```python
# Initializes dictionaries for direct, reverse, divided orders and shifted of encoding message
def initialize(K):  1 usage
    direct_order = {}
    reverse_order = {}
    divide_order = {}
    shifted_order = {}

    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
        shifted_value = (i + 1 + K) % 26
        if shifted_value == 0:
            shifted_value = 26
        shifted_order[letter] = shifted_value

    return direct_order, reverse_order, divide_order, shifted_order
```

The program has two execution modes:

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

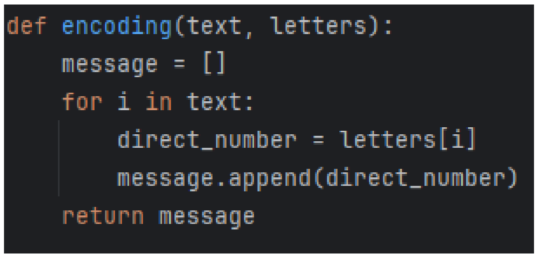
Input of an decrypted string (ENCODING)

The entered string is transformed into lowercase for further use.



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

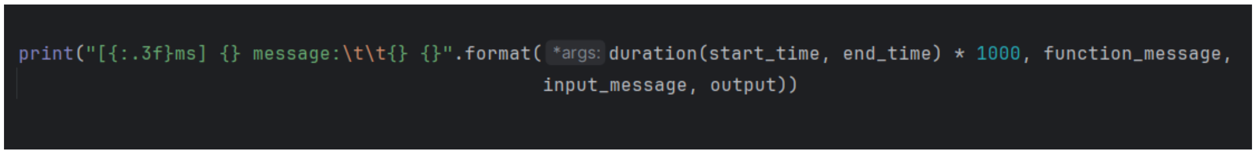
The **encoding()** function is called.
Parameters:
text - the text entered by the user
letters - dictionary with the order of encryption alphabet



```
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.

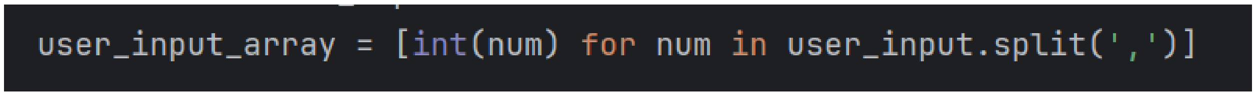
Then, we write out the message.



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

Input of an encrypted string (DECODING)

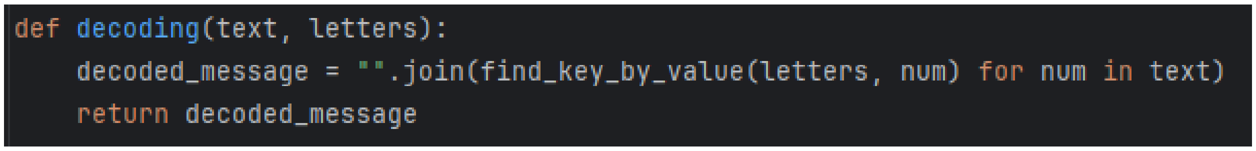
When inputting a message in the form of an encoded message, the delimiter **","** is removed from the string, and the message is saved into an array. The input should be in strict form with a delimiter “,”. Message example: 13,1,20,22,5,5,22,1



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

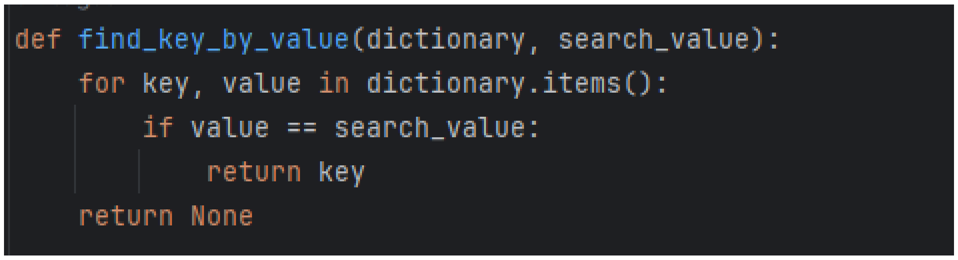
Then, the **decoding()** function is called.

Parameters:
text - encoded message
letters - dictionary with the order of encryption for the alphabet



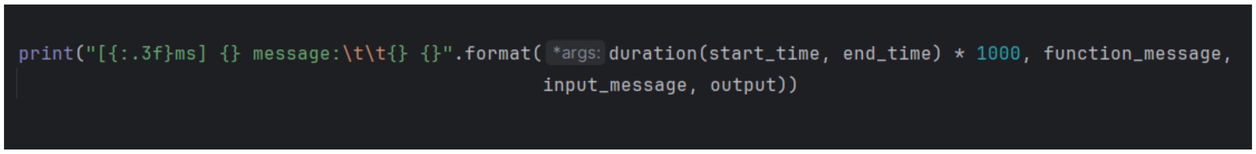
```
def decoding(text, letters):
    decoded_message = "".join(find_key_by_value(letters, num) for num in text)
    return decoded_message
```

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 message.



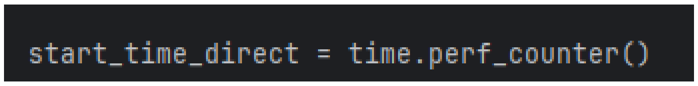
```
print("[{:.3f}ms] {} message:\t\t{} {}".format( *args: duration(start_time, end_time) * 1000, function_message,
                                                 input_message, output))
```

For execution encoding and decoding functions we use a function **exec_with_time()**

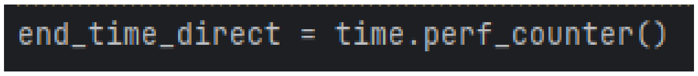
Parameters:
input_message - user input
function - encoding or decoding function
alphabet - the alphabet in direct/reverse/divide/shifted order
function_message - text message for output

Execution time of the program

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.

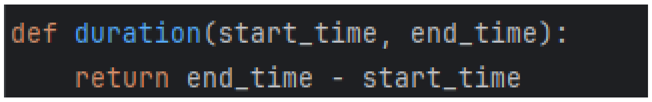


```
start_time_direct = time.perf_counter()
```



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

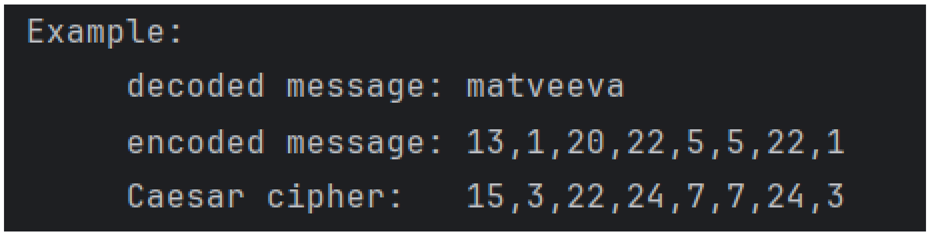
Then, in function **duration()** we calculate the difference between the end and the start.



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

Example of use

When the program starts, an example of the data that can be entered will be shown first.
For example, we used my surname. In the encoded message, the order will be 13,1,20,22,5,5,22,1 and for the shifted order, it will be 15,3,22,24,7,7,24,3. We can use these inputs to test our program.

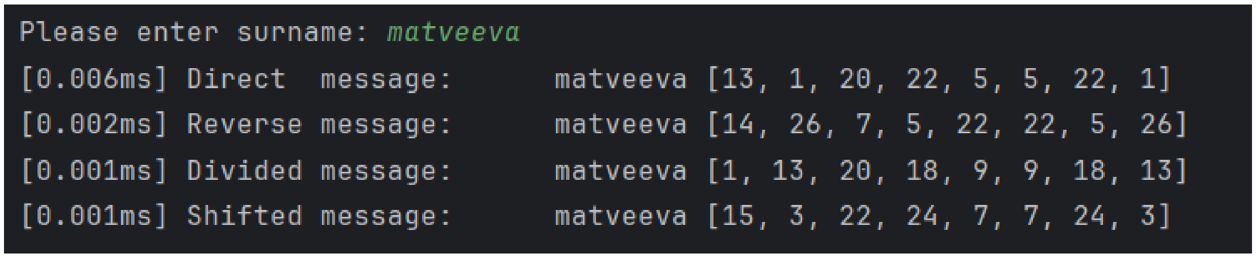


```
Example:
     decoded message: matveeva
     encoded message: 13,1,20,22,5,5,22,1
     Caesar cipher:   15,3,22,24,7,7,24,3
```

Then we can enter our data.

Decoded message

After the program is executed, we will see the result printed in the console. When entering the string "matveeva", we get the following result.



```
Please enter surname: matveeva
[0.006ms] Direct  message:     matveeva [13, 1, 20, 22, 5, 5, 22, 1]
[0.002ms] Reverse message:     matveeva [14, 26, 7, 5, 22, 22, 5, 26]
[0.001ms] Divided message:     matveeva [1, 13, 20, 18, 9, 9, 18, 13]
[0.001ms] Shifted message:     matveeva [15, 3, 22, 24, 7, 7, 24, 3]
```

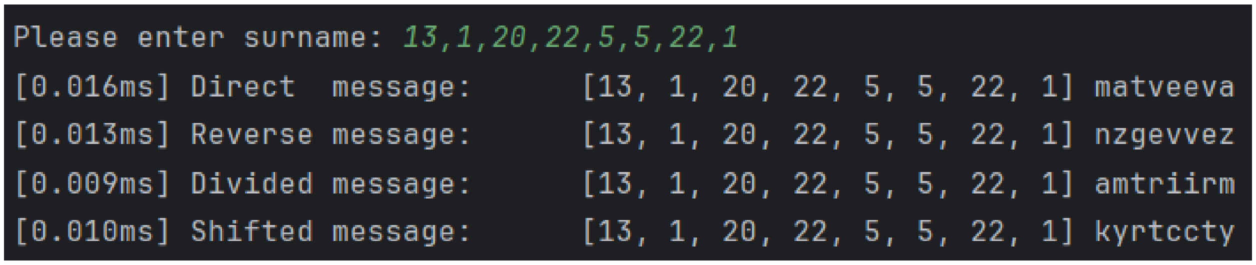
In the first line, we have the alphabet in direct order.
The second line is in reverse order.
The third line has the alphabet divided into two parts.
The last line represents the Caesar cipher shifted order.

At the beginning of each line, we can see the execution time of each method.

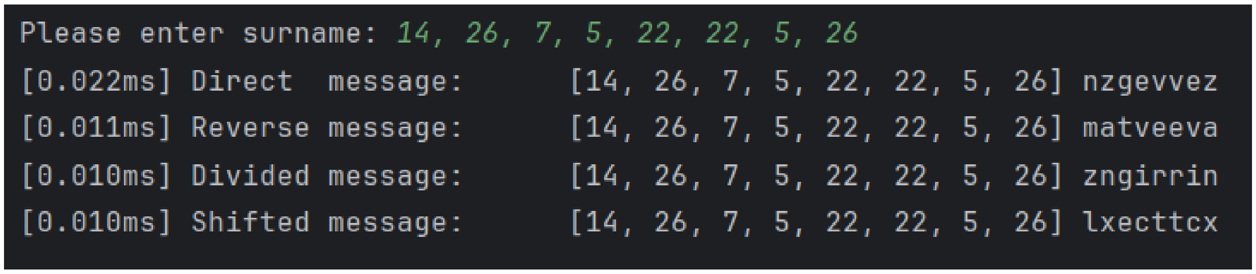
Encoded message

Let's check the correctness of each method's execution. In each case, only one of the four lines should be decoded.

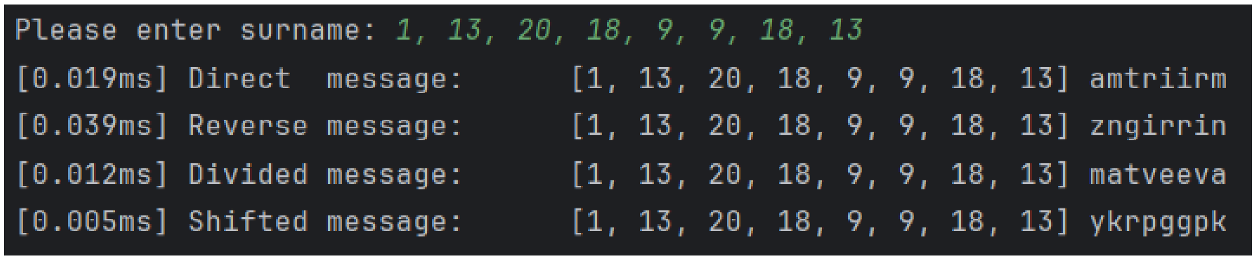
When entering a string that follows the direct order, we only receive a message in the direct order.


```
Please enter surname: 13,1,20,22,5,5,22,1
[0.016ms] Direct  message:      [13, 1, 20, 22, 5, 5, 22, 1] matveeva
[0.013ms] Reverse message:      [13, 1, 20, 22, 5, 5, 22, 1] nzgevvez
[0.009ms] Divided message:      [13, 1, 20, 22, 5, 5, 22, 1] amtriirm
[0.010ms] Shifted message:      [13, 1, 20, 22, 5, 5, 22, 1] kyrtccty
```

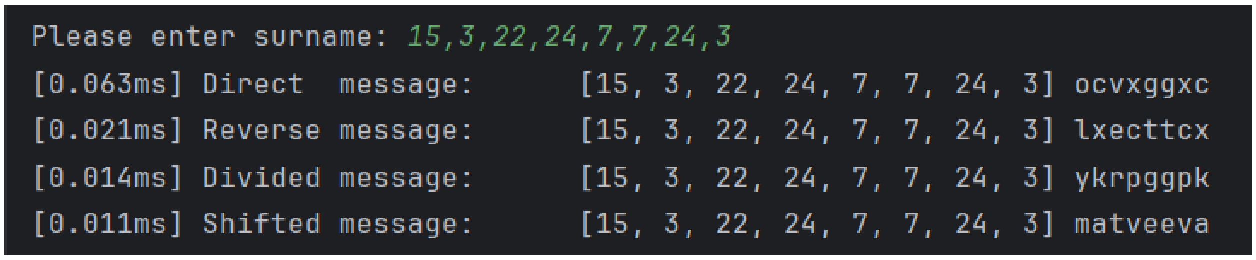
Next, we enter the reverse order and receive only the reverse message as a result.


```
Please enter surname: 14, 26, 7, 5, 22, 22, 5, 26
[0.022ms] Direct  message:      [14, 26, 7, 5, 22, 22, 5, 26] nzgevvez
[0.011ms] Reverse message:      [14, 26, 7, 5, 22, 22, 5, 26] matveeva
[0.010ms] Divided message:      [14, 26, 7, 5, 22, 22, 5, 26] zngirrin
[0.010ms] Shifted message:      [14, 26, 7, 5, 22, 22, 5, 26] lxecttcx
```

Next is the divided order, and we also receive only the divided message.


```
Please enter surname: 1, 13, 20, 18, 9, 9, 18, 13
[0.019ms] Direct  message:      [1, 13, 20, 18, 9, 9, 18, 13] amtriirm
[0.039ms] Reverse message:      [1, 13, 20, 18, 9, 9, 18, 13] zngirrin
[0.012ms] Divided message:      [1, 13, 20, 18, 9, 9, 18, 13] matveeva
[0.005ms] Shifted message:      [1, 13, 20, 18, 9, 9, 18, 13] ykrpggpk
```

Let's check the last method, shifted order. We also received a decoded message only for this method.


```
Please enter surname: 15,3,22,24,7,7,24,3
[0.063ms] Direct  message:      [15, 3, 22, 24, 7, 7, 24, 3] ocvxggxc
[0.021ms] Reverse message:      [15, 3, 22, 24, 7, 7, 24, 3] lxecttcx
[0.014ms] Divided message:      [15, 3, 22, 24, 7, 7, 24, 3] ykrpggpk
[0.011ms] Shifted message:      [15, 3, 22, 24, 7, 7, 24, 3] matveeva
```

Github: https://github.com/MatveevaLI/ZKGRA