

CW1

PROGRAMMING AND ALGORITHMS 2

Toqa Mahmoud

CU1900305

TABLE OF CONTENTS

Abstract	3
Introduction.....	3
Explanation of the Code.....	3
Rotor settings	3
Reflector settings.....	3
Plugboard settings.....	3
Ciphering/deciphering.....	3
Instructions for using the program	4
Repository's Link	4
Repository's organization	4
Pre-requisites	4
How to use the code?	4
Discussion of the code.....	5
Coding style.....	5
Code Efficiency	6
Extra features.....	7
Class diagram	8
Functions.....	9
Algorithms.....	9
Scenarios.....	13
Test case scenarios	16
First Test Case Scenario.....	16
Second Test Case Scenario	18
Conclusion	21
List of Figures	22
List of Tables.....	23
References.....	24

ABSTRACT

The necessity for a mechanical encryption mechanism became apparent in the 1920s. It was discovered that rotating a substitution cypher was the best solution. This concept has previously been employed in a variety of manual cyphers, but mechanization allowed it to be made significantly more efficient and complex. This is when the Enigma machine was invented. The engima machine is an encryption device used by Nazi Germany in world war II to cipher/decipher messages (Enigma machine - Wikipedia, 2021). On the other hand, after the world's modernization, the Enigma machine can now be simply implemented using a software.

INTRODUCTION

In this coursework, I simulated the Enigma Machine adding to it some extra features, using object oriented programming (OOP), in python. This engima machine program is designed to be highly customisable, which means users define setups. The program was precisely designed by optimal use of data types and data structures, proper use of control statements, efficient modular design using classes and methods, and coherent flow sequence. This report aims to provide detailed analysis of this program.

EXPLANATION OF THE CODE

ROTOR SETTINGS

The program permits choosing 3 rotors from 5 rotors as well as choosing the three starting points of the 3 rotors. Rotor settings are reset after every message input.

REFLECTOR SETTINGS

The program permits choosing 1 reflector from 3 reflectors, it then pairs it automatically with its reflector map, which contains the permutation cipher for each letter. Reflector settings are reset after every message input.

PLUGBOARD SETTINGS

The user has the flexibility on whether they want to use a plugboard or not. If they wish to use one, they also have the flexibility to choose 1 from 3 preset plugboards or configure their own plugboard. Plugboard settings are reset after every message input.

CIPHERING/DECIPHERING

After configuring all these settings, the user has the option to either type the message or import a text file. After that is done, the message passes, letter by letter, through the plugboard which is the first stage of the encryption process. It is based on substitution cipher principles, which is a type of transposition encryption. After passing through the plugboard, the message passes through the three rotors in sequence, each modifying it using a combination of transposition and Caesar ciphers. Every time a letter is encrypted, the first rotor turns by 1 letter.

The rotors are connected to each other so that when the first rotor has turned 60 times, it triggers the second rotor to rotate by one letter. Similarly when the second rotor rotates 60 times, the last rotor rotates by one letter.

Once the letter has gone through the three rotors from rotor 1 to rotor 3, the reflector will reflect the letter back, sending the encrypted letter through the rotors from rotor 3 to rotor 1 for another 3 stages of encryption and then through the plugboard again for a final substitution cipher (Enigma Encoder | 101 Computing, 2021). Finally, after the ciphering process is complete, the user has an option to save the output in a file.

INSTRUCTIONS FOR USING THE PROGRAM

REPOSITORY'S LINK

https://github.com/Toqahassib/Programming_CW1.git

REPOSITORY'S ORGANIZATION

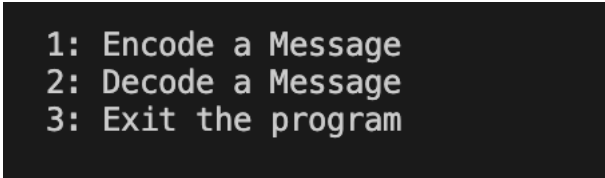
1. README.md
 - Contains information for the user about the code.
2. cw1.py
 - This is the main Enigma file. Users will run this file.
3. plugboard.py
 - Plugboards are stored in this file.
4. rotor.py
 - Rotors and reflectors are stored in this file.
5. test.txt
 - text file to test the import file feature
6. CW1_report.pdf
 - Detailed report on the program

PRE-REQUISITES

Programming Language: Python 3.9.7

HOW TO USE THE CODE?

To start the program, open your terminal and change the current directory to where you want to clone the repository. Then clone it using the repository link and finally run the "cw1.py" file. After running the file the following menu will be printed out:



```
1: Encode a Message
2: Decode a Message
3: Exit the program
```

Figure 1: Printed menu

After choosing selection 1 or 2, the following steps will follow

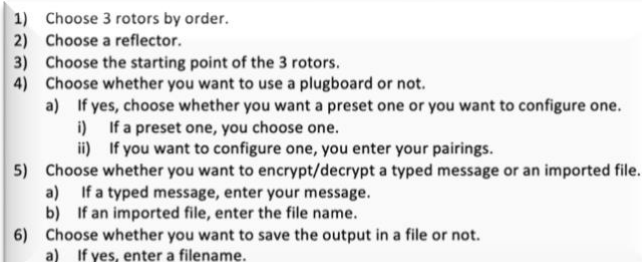
- 
- ```
1) Choose 3 rotors by order.
2) Choose a reflector.
3) Choose the starting point of the 3 rotors.
4) Choose whether you want to use a plugboard or not.
 a) If yes, choose whether you want a preset one or you want to configure one.
 i) If a preset one, you choose one.
 ii) If you want to configure one, you enter your pairings.
5) Choose whether you want to encrypt/decrypt a typed message or an imported file.
 a) If a typed message, enter your message.
 b) If an imported file, enter the file name.
6) Choose whether you want to save the output in a file or not.
 a) If yes, enter a filename.
```

Figure 2: Code steps

## DISCUSSION OF THE CODE

At the beginning of implementing the program, I stored the rotors data in dictionaries, which use *keys* to associate with each value, this means that the same keys will be stored for 5 rotor dictionaries. Taking all this space for the same keys is extremely inefficient, so I decided to use lists. The ciphering/deciphering process needed the rotors and reflector data to be represented as a sequence of items indexed by their integer position, therefore lists were the best option. Moreover, lists are *mutable*, meaning that I can add, delete, or change elements in a flexible manner. Another sequential data structure is a tuple; the difference between these two is that tuples are immutable. Because of that, I stored the plugboard data in a list of tuples. Since all plugboard pairings are fixed, I stored each pair in a tuple and stored the tuples in a list to be represented by an indexed sequence of items (Which Python Data Structure Should You Use?, 2021).

Table 1: Data structures

| Class     | Integers         | Strings     | Lists            | List of tuples |
|-----------|------------------|-------------|------------------|----------------|
| Enigma    | plugboard_chosen | encrypted   | RotorI           | ReflectorA_map |
| Plugboard | selection        | encoded_msg | RotorII          | ReflectorB_map |
| Rotors    | plugboard        | cipher      | RotorIII         | ReflectorC_map |
| Reflector | count            | msg         | RotorIV          | plugboard29    |
|           | rotor1_rotation  | new_msg     | RotorV           | Plugboard30    |
|           | rotor2_rotation  | first_rotor | alphabet         | Plugboard31    |
|           | index            | mid_rotor   | reflectorA       | user_plugboard |
|           | starting_letter  | last_rotor  | reflectorB       |                |
|           | ans              | r1          | reflectorC       |                |
|           | rotor            | r2          | list_msg         |                |
|           | msg_type         | r3          | rotors           |                |
|           |                  | reflector   | reflector        |                |
|           |                  | x           | reflector_map    |                |
|           |                  | start       | plugboard_chosen |                |
|           |                  | file_name   | first_rotor      |                |
|           |                  | save_output | mid_rotor        |                |
|           |                  |             | last_rotor       |                |

## CODING STYLE

The code is designed in a systemized way by conducting OOP. Working with OOP did not only give me the flexibility to clearly structure a clean code, but also eased its modification and maintenance (Python, 2021). The Enigma machine is made up of 3 main parts – the rotors, the reflector, and the plugboard. Therefore, my code is divided into these 3 main parts as classes in addition to the Enigma class where the message is ciphered/deciphered. The following illustrates the 4 classes and objects used in the code:

- ❖ The Plugboard class contains 2 objects – 1 to store the plugboard and 1 to store the encrypted message.
- ❖ The Rotor class contains 4 objects – 1 to store the first rotor, 1 to store the second rotor, 1 to store the third rotor, and finally 1 to store all 3 rotors.
- ❖ The Reflector class contains 2 objects – 1 to store a reflector and 1 to store the reflector map.
- ❖ The Enigma class contains 6 objects – 1 for the rotors (passed from the Rotor class), 1 for the reflector (passed from the Reflector class), 1 for the reflector map (passed from the Reflector class), 1 for the message (passed from the Plugboard class), and 1 for the final ciphered/deciphered message.

I've styled my code in that particular way for various reasons including:

- **ORGANIZATION:** data and procedure were stored at varying levels – storing plugboard data in the plugboard class, storing rotors data in the rotor class, and storing reflector data in the reflector class. Later, this will help me and others understand, edit, and use my code (Python Classes, 2021).
- **MAINTAINABILITY:** this approach made my code more maintainable. Identifying the source of errors was easier because each object is self-contained in its respective class (Python Classes, 2021).

## CODE EFFICIENCY

Since most of my inputs are in while loops (to handle errors), there is a probability that the user does not enter a valid input. Therefore, the while loop will always evaluate to true so this has worst case of  $O(\text{infinity})$ . Time complexity is no longer a valid statistic if the conditions for evaluating the while loop to false are dependent on the user. (How to find the Time Complexity of a Python Code, 2021). Therefore, the following calculations will perform only on the ciphering/deciphering part of the code, assuming the user has entered all inputs successfully.

Table 2: Time Complexity

| Class     | function Name      | Screenshot of Function                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Time Complexity                                                                                             |
|-----------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| Enigma    | encode()           | <pre>def encode(self):     self.rotor1_rotation = 0     self.rotor2_rotation = 0     self.cipher = ""     for i in range(len(self.msg)):         if not self.msg[i].isalnum():             self.cipher += self.msg[i]         else:             self.rotation()             index = alphabet.index(self.msg[i])             encrypted = self.rotors[0][index]             index = alphabet.index(encrypted)             encrypted = self.rotors[1][index]             index = alphabet.index(encrypted)             encrypted = self.rotors[2][index]             index = alphabet.index(encrypted)             encrypted = reflectorB[index]             for x in self.reflector_map:                 if x[0] == encrypted:                     encrypted = x[1]                 elif x[1] == encrypted:                     encrypted = x[0]             index = reflectorB.index(encrypted)             encrypted = alphabet[index]             index = self.rotors[2].index(encrypted)             encrypted = self.rotors[2][index]             encrypted = alphabet[index]             index = self.rotors[1].index(encrypted)             encrypted = self.rotors[1][index]             encrypted = alphabet[index]             index = self.rotors[0].index(encrypted)             encrypted = self.rotors[0][index]             encrypted = alphabet[index]             self.cipher += encrypted</pre> | $O(n \times m)$<br>where $n$ is $\text{len}(\text{self.msg})$<br>and $m$ is<br>$\text{self.reflector\_map}$ |
|           | rotation()         | <pre>def rotation(self):     Rotors.rotate(self, 0)      self.rotor1_rotation += 1      if self.rotor1_rotation == 60:         Rotors.rotate(1)      self.rotor1_rotation = 0     self.rotor2_rotation += 1      if self.rotor2_rotation == 60:         self.rotor2_rotation = 0         Rotors.rotate(2)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | $O(1)$                                                                                                      |
| Plugboard | plugboard_settings | <pre>def plugboard_settings(self, msg):     list_msg = list(msg)      for i in range(len(msg)):         for x in self.plugboard_chosen:             if list_msg[i] == x[0]:                 list_msg[i] = x[1]             elif list_msg[i] == x[1]:                 list_msg[i] = x[0]     self.new_msg = "".join(list_msg)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | $O(n \times p)$<br>Where $n$ is $\text{len}(\text{msg})$ and $p$<br>is $\text{self.plugboard\_chosen}$      |

|        |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                         |
|--------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| Rotors | rotor_order()    | <pre>def rotor_order(self):     while self.rotors == []:         for i in range(3):             rotor = INTValidation("Choose rotor {}: ".format(i + 1))              if rotor == 1:                 self.rotors.append(rotorI)             elif rotor == 2:                 self.rotors.append(rotorII)             elif rotor == 3:                 self.rotors.append(rotorIII)             elif rotor == 4:                 self.rotors.append(rotorIV)             elif rotor == 5:                 self.rotors.append(rotorV)          for elem in self.rotors:             if self.rotors.count(elem) &gt; 1:                 self.rotors = []                 break</pre> | O(1)                                                    |
|        | starting_point() | <pre>def starting_point(self):     for i in range(3):         start = input(             "Enter the starting point of rotor {}: ".format(i + 1))          starting_letter = self.rotors[i].index(start)          for x in range(starting_letter):             self.rotate(i)</pre>                                                                                                                                                                                                                                                                                                                                                                                                | $O(1 \times s) = O(s)$<br>where s is the starting point |
|        | rotate()         | <pre>def rotate(self, index):     self.rotors[index].append(self.rotors[index].pop(0))</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | O(1)                                                    |
|        | reset()          | <pre>def reset(self):     self.rotors = []      while rotorI[0] != "E":         rotorI.append(rotorI.pop(0))     while rotorII[0] != "A":         rotorII.append(rotorII.pop(0))     while rotorIII[0] != "B":         rotorIII.append(rotorIII.pop(0))     while rotorIV[0] != "E":         rotorIV.append(rotorIV.pop(0))</pre>                                                                                                                                                                                                                                                                                                                                                 | $O(k)$<br>Where k is != to a string                     |

Time complexity =  $O(n*m) + O(1) + O(n*p) + O(1) + O(s) + O(1) + O(k)$

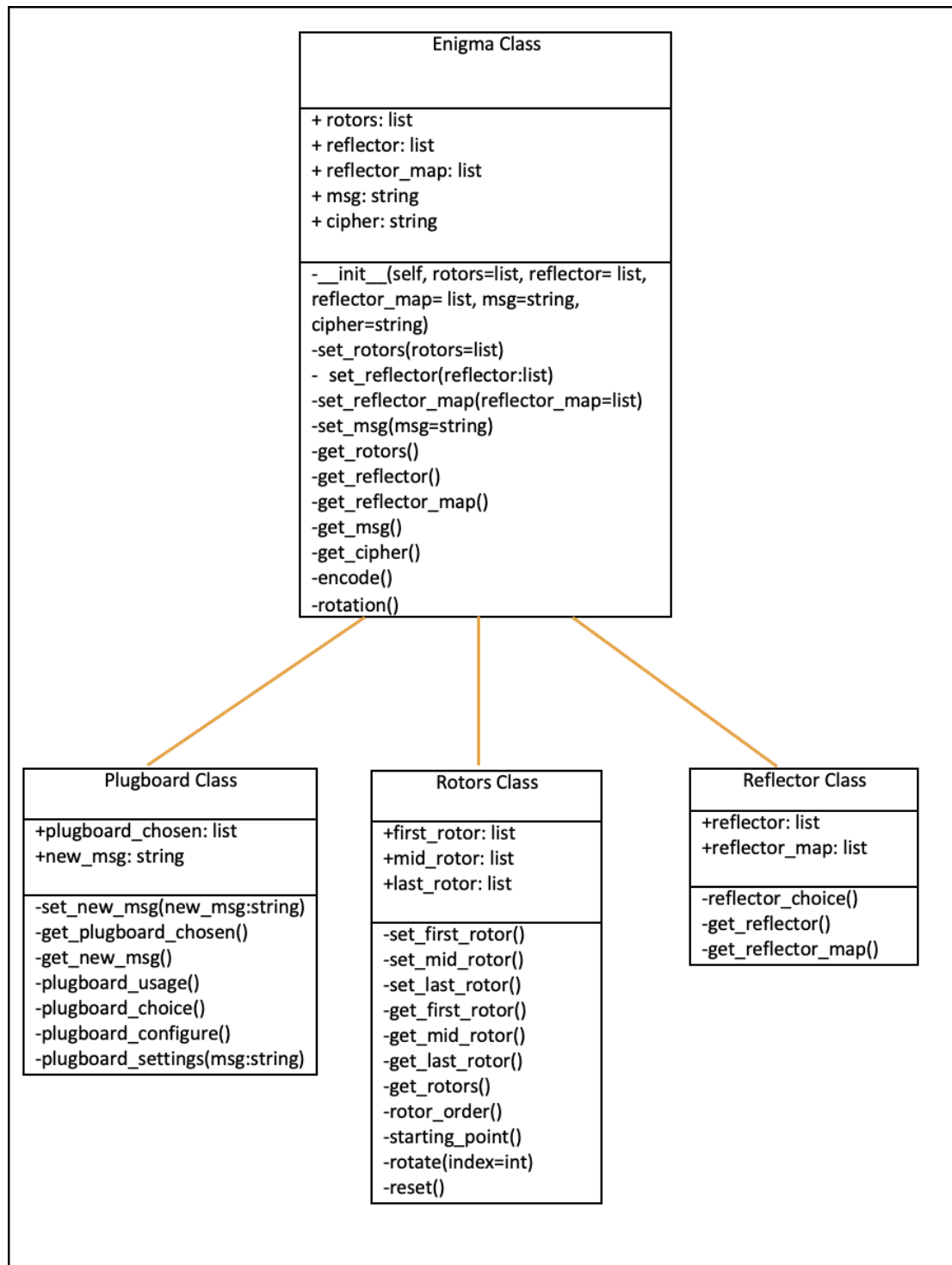
- Keep the fastest growing rate
  - Total time complexity =  $O(n*m)$ 
    - where n is `len(self.msg)` and m is `self.reflector_map`.

## EXTRA FEATURES

1. The program permits the change of the rotor setting each time.
2. The program ciphers/deciphers uppercase letters, lowercase letters, and numbers.
3. The program ciphers/deciphers text files.
4. The program has 5 rotors out of which the user chooses (using a key) only 3 of them in the encryption.
5. The program has 3 reflectors out of which the user chooses (using a key) only 1 of them.
6. The program has 3 plugboards out of which the user chooses (using a key) only 1 of them.
7. The program permits the user to configure their own plugboard pairing (with a 10 pairing limit).
8. The program permits the user to save and display the output in a file.
9. The program handles all special cases from the user.
10. The program accurately handles errors and validates inputs.

## CLASS DIAGRAM

Figure 3: Class Diagram





## FUNCTIONS

Table 3: Functions

| Function Name                | Arguments | Explanation                                                   |
|------------------------------|-----------|---------------------------------------------------------------|
| <b>encode()</b>              | None      | Encodes/decodes the message                                   |
| <b>rotation()</b>            | None      | Rotates the 3 rotors                                          |
| <b>plugboard_usage()</b>     | None      | Sets the plugboard based on user's choice                     |
| <b>plugboard_choice()</b>    | None      | Sets a preset plugboard based on user's choice                |
| <b>plugboard_configure()</b> | None      | Configures a plugboard based on user's choice                 |
| <b>plugboard_settings()</b>  | msg       | Message passes through the plugboard chosen to get encoded    |
| <b>rotor_order()</b>         | None      | Appends the rotors in order to a list                         |
| <b>starting_point()</b>      | None      | Rotates the rotors to the starting point of the user's choice |
| <b>rotate()</b>              | index     | Appends and pops from a list to rotate it                     |
| <b>reset()</b>               | None      | Resets all rotors to initial starting point                   |
| <b>reflector_choice()</b>    | None      | Sets the reflector based on users choice                      |
| <b>save_output()</b>         | None      | Saves output in a file                                        |
| <b>IntValidation()</b>       | Input_int | Validates integer inputs                                      |

## ALGORITHMS

Table 4: Algorithm 1

| Name           | encode()                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| Input          | Message                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                                   |
| Output         | Ciphered/Deciphered Message                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                   |
| Data Structure | Msg: string<br>Index: int<br>Encrypted: string<br>Alphabet: list<br>Rotors: list                                                                                                                                                                                                                                                                                                                                                                                                                                                          | reflector_map: list<br>reflectorB: list<br>self.new_msg: string<br>cipher: string |
| Content        | <p>The pivot is chosen as the index of the letter in the alphabet list. It then corresponds to the same letter in the first rotor list.</p> <p>This process is repeated with the second rotor, third rotor, and reflector lists. After that, the letter is set as its paired letter in the reflector_map list of tuples.</p> <p>Then, the letter will be set as the letter in that index in the alphabet list. It then corresponds to the same letter in the third rotor. This process is repeated with second and first rotor lists.</p> |                                                                                   |

## Solution

```
for i in range(len(msg)):

 index = alphabet.index(msg[i])
 encrypted = rotors[0][index]

 index = alphabet.index(encrypted)
 encrypted = rotors[1][index]

 index = alphabet.index(encrypted)
 encrypted = rotors[2][index]

 index = alphabet.index(encrypted)
 encrypted = reflectorB[index]

 for x in reflector_map:
 if x[0] == encrypted:
 encrypted = x[1]

 elif x[1] == encrypted:
 encrypted = x[0]

 index = reflectorB.index(encrypted)
 encrypted = alphabet[index]

 index = rotors[2].index(encrypted)
 encrypted = rotors[2][index]
 encrypted = alphabet[index]

 index = rotors[1].index(encrypted)
 encrypted = rotors[1][index]
 encrypted = alphabet[index]

 index = rotors[0].index(encrypted)
 encrypted = rotors[0][index]
 encrypted = alphabet[index]

 cipher += encrypted
```

## Analysis

Worst case scenario:  $O(n \times m)$ , where  $n$  is `len(self.msg)` and  $m$  is `self.reflector_map`

Refer to figure 4 for more understanding of how the algorithm works.

Figure 4: Cipherring/Deciphering

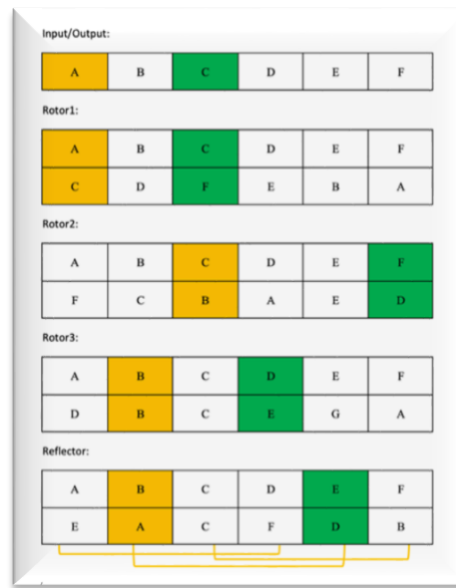


Table 5: Algorithm 2

|                |                                                                                                                                                                                                                                                                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input          | Message                                                                                                                                                                                                                                                                                                                                       |
| Output         | Encoded/decoded message                                                                                                                                                                                                                                                                                                                       |
| Data Structure | Msg: string<br>list_msg: list<br>self.pluginboard_chosen: list of tuples<br>self.new_msg: string                                                                                                                                                                                                                                              |
| Content        | The pivot is chosen as index 0 of the first tuple in the list, it then compares it with the letter until it matches the same letter, it is then changed to the letter in index 1 of the tuple. The same thing happens if the letter matches a letter in index 1 of the tuple, it is then changed to the letter in index 0 of the tuple.       |
| Solution       | <pre> def pluginboard_settings(self, msg):     list_msg = list(msg)      for i in range(len(msg)):         for x in self.pluginboard_chosen:              if list_msg[i] == x[0]:                 list_msg[i] = x[1]             elif list_msg[i] == x[1]:                 list_msg[i] = x[0]         self.new_msg = "".join(list_msg) </pre> |
| Analysis       | Worst case scenario: $O(n \times p)$ , where $n$ is $\text{len}(msg)$ and $p$ is $\text{self.pluginboard\_chosen}$<br>Refer to figure 5 for more understanding of how the algorithm works.                                                                                                                                                    |

Figure 5: Plugboard Encryption

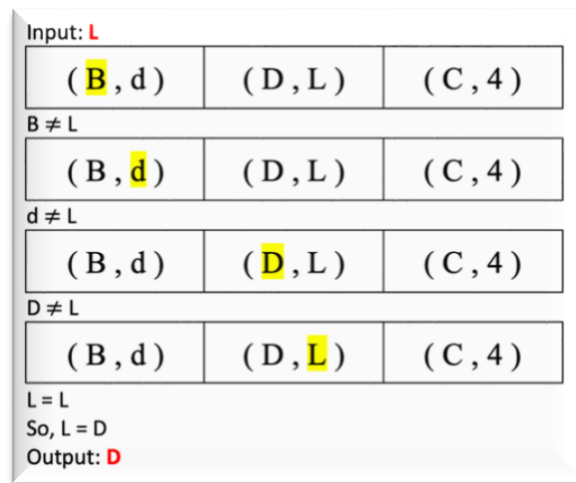
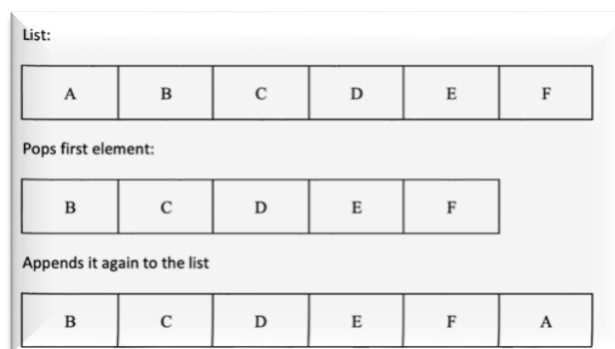


Table 6: Algorithm 3

| Name           | rotate()                                                                                                                  |
|----------------|---------------------------------------------------------------------------------------------------------------------------|
| Input          | List                                                                                                                      |
| Output         | Rotated List                                                                                                              |
| Data Structure | rotors: list                                                                                                              |
| Content        | Pops an item from the beginning of the list and then appends it to the same list.                                         |
| Solution       | <pre>def rotate(index):     rotors[index].append(rotors[index].pop(0))</pre>                                              |
| Analysis       | <p>Worst case scenario: <math>O(1)</math></p> <p>Refer to figure 6 for more understanding of how the algorithm works.</p> |

Figure 6: List Rotation



## SCENARIOS

All inputs handle errors to avoid the program from crashing. The table below shows how these errors are handled.

Table 7: Error Handling

| Testing Error Handling                                                                                                 |                                                                                                                                                                                                                                                  |
|------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Rotors must be chosen only once                                                                                        | <pre>choose rotor 1: 1 choose rotor 2: 1 choose rotor 3: 2  Rotors can't be chosen twice. Please try again. choose rotor 1: █</pre>                                                                                                              |
| All integer inputs cannot be strings or any special character                                                          | <pre>choose rotor 1: A  Invalid, answer should be in numbers. choose rotor 1: █</pre>                                                                                                                                                            |
| A reflector can be chosen from 1 of the following 3 letters (not case sensitive): A, B or C.                           | <pre>Choose a reflector (A, B, or C): d Invalid. You can only enter A, B, or C. Choose a reflector (A, B, or C): █</pre>                                                                                                                         |
| Starting point of the rotors should be a character in the list, which is either a single upper/lowercase or one number | <pre>Enter the starting point of rotor 1: 11  Enter one alphabet characet or a number form 0-9 ONLY. Enter the starting point of rotor 1: #  Enter one alphabet characet or a number form 0-9 ONLY. Enter the starting point of rotor 1: █</pre> |
| Yes or no questions can only be answered using the following inputs (not case sensitive): yes, y, no, or n             | <pre>Do you want to use a plugboard (y/n)? k Invalid. You can only answer with yes, y, no, or n.  Do you want to use a plugboard (y/n)? █</pre>                                                                                                  |

Choices must be included in the printed menu.

```
1: Choose a preset plugboard
2: Configure a plugboard
```

```
Choose one from the above: 3
```

```
Invalid. You can only choose either 1 or 2.
Choose one from the above: a
```

```
Invalid, answer should be in numbers.
Choose one from the above: █
```

A preset plugboard can be chosen from 1 of the following 3 numbers: 29, 30, or 31

```
choose your plugboard (29, 30, 31): 44
```

```
You can enter from 29 to 31 ONLY.
choose your plugboard (29, 30, 31): \
```

```
Invalid, answer should be in numbers.
choose your plugboard (29, 30, 31): █
```

Configuring plugboard parings must be entered in the following format: X,X

```
Example:
Enter pair 1: a,e

Enter pair 1: d J
Invalid input.

Example:
Enter pair 1: a,e

Enter pair 1: Lp
Invalid input.

Example:
Enter pair 1: a,e

Enter pair 1: (k,l)
Invalid input.

Example:
Enter pair 1: a,e

Enter pair 1: m,p
Invalid input.

Example:
Enter pair 1: a,e

Enter pair 1: werfd
Invalid input.

Example:
Enter pair 1: a,e

Enter pair 1: E,l
You have succussfully configured the plugboard!
```

S

Filename should exist, when importing files.

```
Choose one from the above: 2
```

```
Enter your file name: xx
File not found. Please try another filename.
```

```
Enter your file name: █
```

To make the code user-friendly, all inputs are not case sensitive. The table below illustrates this.

Table 8: Case Sensitivity

| Testing Case Sensitivity                                           |                                                                                                                                                                    |
|--------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| In this example, the code continued smoothly choosing reflector A. | <pre>Choose a reflector (A, B, or C): a Enter the starting point of rotor 1: w Enter the starting point of rotor 2: e Enter the starting point of rotor 3: r</pre> |
| In this example, the code continued smoothly choosing a plugboard. | <pre>Do you want to use a plugboard (y/n)? Y 1: Choose a preset plugboard 2: Configure a plugboard Choose one from the above: 1</pre>                              |

## TEST CASE SCENARIOS

### FIRST TEST CASE SCENARIO

The following snapshots show a complete valid scenario with inputs showing how my program can execute all its functionalities correctly. All inputs are colored in green. The program will cipher the following message:

**Hey, this is the 1<sup>st</sup> Test Scenario!**

Figure 7: 1st Test Case Scenario

```
This is an enigma machine, please select what you need to do by inserting the corresponding number

1: Encode a Message
2: Decode a Message
3: Exit the program

Enter your selection: 1
choose rotor 1: 1
choose rotor 2: 2
choose rotor 3: 3

Choose a reflector (A, B, or C): A

Enter the starting point of rotor 1: E
Enter the starting point of rotor 2: a
Enter the starting point of rotor 3: 3

Do you want to use a plugboard (y/n)? y

1: Choose a preset plugboard
2: Configure a plugboard

Choose one from the above: 1

choose your plugboard (29, 30, 31): 30

1. Type a message
2. Import a file

Choose one from the above: 1

Enter your Message: Hey, this is the 1st Test Scenario!
```



```
Result: zXC, uk1b QB sQF t0e PB1d kdrwUxDz!

Do you want to save the output in a text file (y/n)? y
Enter your file name: 1st.txt

File saved Successfully!

1: Encode a Message
2: Decode a Message
3: Exit the program

Enter your selection: 2
choose rotor 1: 1
choose rotor 2: 2
choose rotor 3: 3

Choose a reflector (A, B, or C): A

Enter the starting point of rotor 1: E
Enter the starting point of rotor 2: a
Enter the starting point of rotor 3: 3

Do you want to use a plugboard (y/n)? y

1: Choose a preset plugboard
2: Configure a plugboard

Choose one from the above: 1

choose your plugboard (29, 30, 31): 30

1. Type a message
2. Import a file

Enter your Message: zXC, uk1b QB sQF t0e PB1d kdrwUxDz!

Result: Hey, this is the 1st Test Scenario!

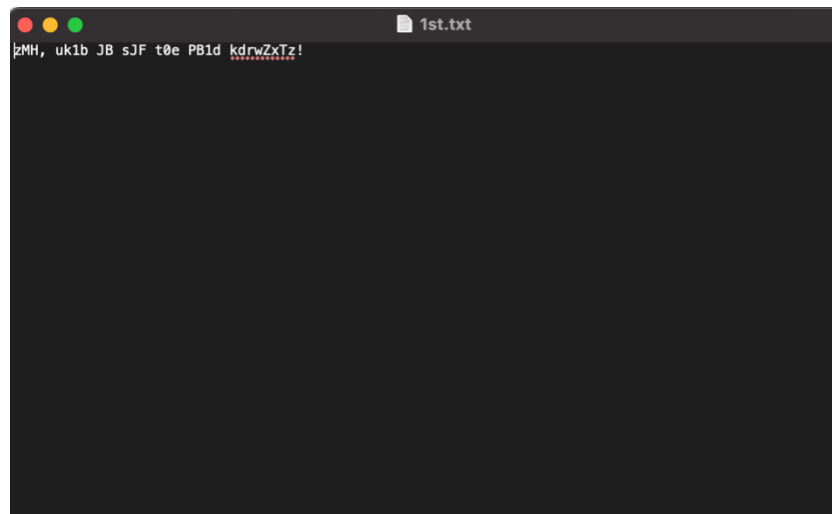
Do you want to save the output in a text file (y/n)? n

1: Encode a Message
2: Decode a Message
3: Exit the program

Enter your selection: |
```

## Saved Output:

Figure 8: Saved Output File



## SECOND TEST CASE SCENARIO

The following snapshots show a complete valid scenario with inputs showing how my program can execute all its functionalities correctly. All inputs are colored in green. The program will cipher the following text file, which contains 3,600 characters:

Figure 9: Import File

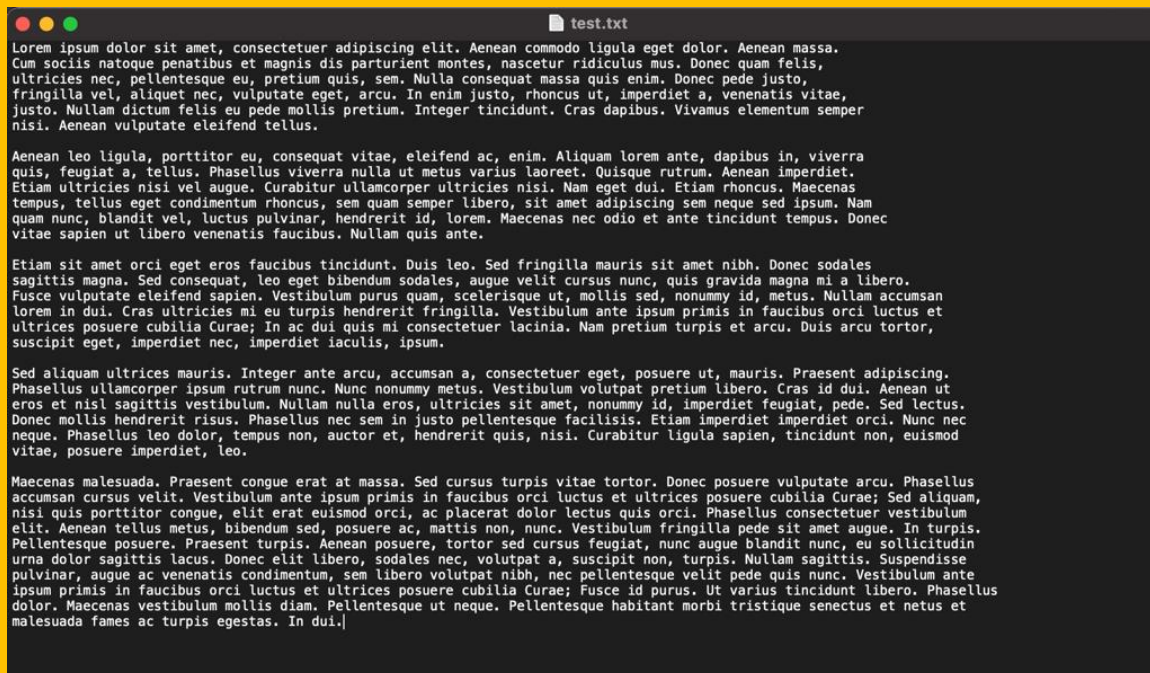


Figure 10: 2nd Test Case Scenario

```
This is an enigma machine, please select what you need to do by inserting the corresponding number

1: Encode a Message
2: Decode a Message
3: Exit the program

Enter your selection: 1

Choose rotor 1: 3
Choose rotor 2: 4
Choose rotor 3: 5

Choose a reflector (A, B, or C): b

Enter the starting point of rotor 1: W
Enter the starting point of rotor 2: s
Enter the starting point of rotor 3: 1

Do you want to use a plugboard (y/n)? y

1: Choose a preset plugboard
2: Configure a plugboard

Choose one from the above: 2

How many pairings will you configure (limit: 10)? 2

Example:
Enter pair 1: a,e

Enter pair 1: H,2
Enter pair 2: t,S
You have succussfully configured the plugboard!
```

1. Type a message
2. Import a file

Choose one from the above: **2**

Enter your text file name: **test.txt**

File imported successfully!

Result: q0P82 ZgnQd UjYc0 p00 rvwJ, uKJzhBXbvr7K luf5vjKvWS n5yF. d63Nqw wIWG1ys uIkVuK Nkan h7MNa. QVmjJo nFY7S. Tzk xVdg0n 24Tj7J4 ajY0UznSS v9 FcUmBo F0e th9CcqeAoZ lcbrc6C, 9MfdMesS 7HzwDg1Y0 52Y. PtR8a aW4e nmda2, nNjKkMors Mkh, oE4mY4s1vJrd jh, mYrQlxt 0rGp, 4bp. MUBi2 LLKP4NoLF Jmgx7 vpFn tFvR. AGFjK MlkZ H9RXa, vTCHAU5N2 axo, AEtlbCOW z7J, xwJLATnC2 C1K4, 8051. HK cs1A WMmq, iqGF3z6 z8, Z006mUyp0 D, DsiDmoJdq ju4Pd, 3n516. tc35ud kv9CLD WVC9Y Nx hyxm 4rod0X V1wmf4w. q6x0LFG KANU2xb5o. dxkS HqQJ6cQ. 5oFrsHD dAYGvVMXn V9ptyi VQxf. lM2bdT QPKesBRAK DGZDUTZD wgm02f.

JQkFnA Y6t 35TK3m, YXs0s78CN 3J, 1qpVOT3AR M9Ybc, 5ujdMLqG sX, 5hK3. ZTcGvdG Fm7ut I344, 39q22Ur MS, FwhbBM3 clvj, POL3UKg X, pLECXQ. r8CY5RJio lLPKhjm iAgbk xX L908l XVJdxK eQ7fqws. lnTLYYP Tbe5f0. LFUMAY wwwThr7oi. uLBLe K1g2yt5WI CaG8 tWB lXvfS. 1hdzp1T56 yYIf1QaUWBV ftocPnwZr HXTT. 3oy PhSN eS0. 16SZa GfdceYF. 4YddE4Q3 FVewwH, IMhEdE Lclz jNMsXcc89vv q2potyW, wHg B3yq Bd48BT uelRzF, IhJ WU4k eZpGb90m7k tiV 056xy vvu efYYc. Lkv PqqQ rrvE, DXx52eg Zqn, NIflcG DnhzyRG8, WrmRwrvh SY, YjMRi. 4lRh1lNp 9uR lz4f np JQ0y i0mjbNWB0 VftXzh. T10If RHg7Q 6t9Ub1 gP 5HN3bm 3yXQGugtT 1GyYqLhM. bDiPiG Lx2v dva8.

akq4j twP 51Sv qZFF sDp9 TEZe Q1PNTJCw 2UbB82vMz. Z301 EwR. 7aZ 8il1JhTcZ DD8D97 3Sb 89Vs v5Jj. VDC1e LEeYBQ5 VA6BQaAS p9J6t. a2G F65IxGtX5, BRL LtEP BfNnVAsi xxYduxj, ybp0n 2jQde XoLm6P hdhJ, Mhsf qPwZEgk ndMwX QL o 6M9tmr. rNGxf aZxSC09zQ ygr7jia0 6e1V11. ZRLWwAgC8f tgC2D ctoH, 6D28lI525L1 ZG, BgKV7x VSW, 5zye4PF Sy, J6YZa. BEr7sU zMhBU9Pm m5ay4 7W Q2J. eEZh cEAZB85mu aR le BMbAQ7 BwCLurZ70 yPp8nVeJp. GM8DQYSwZD z8Fk J6r23 oEXls7 jN xFCuIc7T 64Cq T35ef2 oN hbmeYpxP XWY4PTC 1K9LNor M8h7G; Sd 22 XMY QSxr MG DnmgjABY37fF 0zfQAV1. GpW LisiDVd 9S0QRV lg 6PLq. e0TM 9Mdn B7kR6x, vr2Y0ECi wQ8g, 7DErIHc0S vYD, 1x8QVw3qx 1innaH4, XiZaf.

Zml PtaLjDt 0GZxD141 sG6foN. zb8IuHE EsYM fUav, YTK1vz3m H, bGFLjlQdxBl5 Vmyb, KJp0vXB tJ, Y73lUu. bwdKq9lH gBnxkNJkIi. Twt1SYCSS gui7lPae9bV MTjQq M7VK82 Ma0G. KHbT TvuvIuw TiJsE. imafUp0yzn igFpganU kCUFT07 6lgu5u. HlVf oq e2d. KxLZgA vf RsVi Al mShz HkhY90gc LSdPDikBEC. uNYcio 0LyEL M1Ff, 1CIy9gsac lDm 1p9W, qtRTja8 MW, WH4b1R2YV aZ08cVe, vYXL. 210 Cl6uCQ. FQPxV KN30G0 z5UBQDTPd tc4Tx. OYSZVjqik ig3 JjE HH nhU7M R1J1ahJZHieG n8PrzaiwI. wi3lj l37qJgDbc 0vUqavEn2 L1lZ. C970 0q9 HGkyC. 8wDQS4Ktx t7H UMz2H, GahW20 yCB, fjr0dj pg, 4Q4lJB8zj HXNV, 2gTF. kvjS3pwAM eHQFXJ QFA0cL, ZS9uG3kcU XED, oHt0Yi9 BUyKV, JIBvJJE C5ovz7CuM, znq.

Auv4iSr0 0uRcCtHjS. gY0YDVzL g43kxi Lu8J 2r pWmGf. yP0 Np0n69 iLv1Ho Eox19 CqM89g. 3vusW QP4P9Rl aZx5NU5Q6 MhoP. uMACJTjtY nQrJ02cE 7KoIoP 9PBpB. QiZNxiMDcT si9L cIt8F P5eMmF 7s 9Im6n0dx gahG rDAHDO Wx yIavFAZA FEr9cL7 8U4IFh7 ib1z8; eVU 4jkkNZ8, 0To4 mKYI LKSUCg9mV eedysw, gBMe 2gYc 4vRajwo WxW8, UW NrFHM8io GLPqb vY5Apt aPtN jtaY. oRKv06u0j SlkynxInHr1W kGIueBKKhs fB84. MwakZB 0Ix9LG 1osIt, zYzSNyMy K54, 2GgGqSE yC, kAGHg0 rP0, PzN6. Ktn29RmSgx OdDfBzzXi SI2W eXX tLTd 9PcAa. Nh 94M1c9. E5Ga94EMcgUo P0vr5IP. 4iXv2nPV ympaWt. PGfmR2 87mBDQz, 4Nxryj XPO NIhGdd GFrLPBM, J6kJ KwFCD qXDY7Jg hPZ0, 2j CJEdEWaE29HJ K155 tqGm6 1eB8HmpN bkaTV. pCB1t 706y TOK4kn, Qh3GeHI efl, VyUwdnBq 8, D8AHl7K0 154, 0dEdz2. Wc8LoB SJG7VKAi. mIz0Ja9hKEf 2z7ztGHv, K1Hv0 wy oP2ZmLoCS Xcb7bnCvWEp, CmJ iFcIVu wtr9aKv0 oHF5, lKr yTuKgXISd2Xn 0ECUU eVUV AIP6 ir6F. M6JJmfnpaw WDXb eW7WT 4TM1Y6 Gu 0fbLpW9T V6jb KMhxcL LA yvIvsou3 0zNKM89 9j6tHDL UzPlv; UpbkW at KFZUi. 6Z nHNTdF LEfNWXxhe fRPR7B. 1FxSC3xIM X7kIL. EqGNduCL cHKeQsBy7T z0UdvG HVjA. kRZ9GIbHiRSM UF I9Qfs. 1FL99YddITPs JAp6xNIR 71MK3 FGvjTKtUn t9vVfrUG C1 fmCzX ZE Lds8Avtty sPPj5 nu qQZre6 d8a1wNR. 0w flz.

Do you want to save the output in a text file (y/n)? **n**

- 1: Encode a Message
- 2: Decode a Message
- 3: Exit the program

Enter your selection:

## CONCLUSION

To conclude, this coursework implemented the enigma machine with extra features. Working with classes/OOP made the program very organized which makes it reusable. This program contains many options for the user, not only can the user choose different rotors and reflectors every time but also configure their own plugboard. This gives the user various options and makes the program extremely flexible.

At the beginning of this assignment, I had little experience working with data structures and classes/OOP, so a large portion of time was dedicated to research, understanding and testing, as well as looking at alternative implementations to figure out what was most efficient way. Finally, this made me improve my coding/programming skills and it also enhanced my problem solving skills.

**LIST OF FIGURES**

|                                         |    |
|-----------------------------------------|----|
| Figure 1: Printed menu .....            | 4  |
| Figure 2: Code steps .....              | 4  |
| Figure 3: Class Diagram .....           | 8  |
| Figure 4: Ciphering/Deciphering .....   | 11 |
| Figure 5: Plugboard Encryption .....    | 12 |
| Figure 6: List Rotation .....           | 12 |
| Figure 7: 1st Test Case Scenario .....  | 16 |
| Figure 8: Saved Output File .....       | 18 |
| Figure 9: Import File .....             | 18 |
| Figure 10: 2nd Test Case Scenario ..... | 19 |

**LIST OF TABLES**

|                                 |    |
|---------------------------------|----|
| Table 1: Data structures .....  | 5  |
| Table 2: Time Complexity .....  | 6  |
| Table 3: Functions.....         | 9  |
| Table 4: Algorithm 1 .....      | 9  |
| Table 5: Algorithm 2 .....      | 11 |
| Table 6: Algorithm 3 .....      | 12 |
| Table 7: Error Handling.....    | 13 |
| Table 8: Case Sensitivity ..... | 15 |

## REFERENCES

- 101 Computing. 2021. *Enigma Encoder | 101 Computing*. [online] Available at: <<https://www.101computing.net/enigma-encoder/>> [Accessed 13 November 2021].
- 2019, C., 2021. *A1: Enigma - CS 3110 Spring 2019*. [online] Cs.cornell.edu. Available at: <<https://www.cs.cornell.edu/courses/cs3110/2019sp/a1/#step-6-rotors>> [Accessed 13 November 2021].
- En.wikipedia.org. 2021. *Enigma machine - Wikipedia*. [online] Available at: <[https://en.wikipedia.org/wiki/Enigma\\_machine](https://en.wikipedia.org/wiki/Enigma_machine)> [Accessed 14 November 2021].
- En.wikipedia.org. 2021. *Enigma rotor details - Wikipedia*. [online] Available at: <[https://en.wikipedia.org/wiki/Enigma\\_rotor\\_details#:~:text=not%20altered%20either,-Turnover%20notch%20positions,rotors%20had%20two%20turnover%20notches.](https://en.wikipedia.org/wiki/Enigma_rotor_details#:~:text=not%20altered%20either,-Turnover%20notch%20positions,rotors%20had%20two%20turnover%20notches.)> [Accessed 13 November 2021].
- Medium. 2021. *How To Calculate Time Complexity With Big O Notation*. [online] Available at: <<https://medium.com/dataseries/how-to-calculate-time-complexity-with-big-o-notation-9afe33aa4c46>> [Accessed 13 November 2021].
- Medium. 2021. *How to find the Time Complexity of a Python Code*. [online] Available at: <<https://medium.com/analytics-vidhya/how-to-find-the-time-complexity-of-a-python-code-95b0237e0f2d>> [Accessed 13 November 2021].
- Medium. 2021. *Program Efficiency-Python programming | 6.00.1X*. [online] Available at: <<https://medium.com/self-training-data-science-enthusiast/python-programming-program-efficiency-mit-6-00-1x-2b3e1b2ce0ab>> [Accessed 13 November 2021].
- Medium. 2021. *Which Python Data Structure Should You Use?*. [online] Available at: <<https://towardsdatascience.com/which-python-data-structure-should-you-use-fa1edd82946c>> [Accessed 13 November 2021].
- People.duke.edu. 2021. *Algorithmic complexity — Computational Statistics in Python 0.1 documentation*. [online] Available at: <<https://people.duke.edu/~Eccc14/sta-663/AlgorithmicComplexity.html>> [Accessed 13 November 2021].
- Python, R., 2021. *Object-Oriented Programming (OOP) in Python 3 – Real Python*. [online] Realpython.com. Available at: <<https://realpython.com/python3-object-oriented-programming/>> [Accessed 13 November 2021].
- W3schools.com. 2021. *Python Classes*. [online] Available at: <[https://www.w3schools.com/python/python\\_classes.asp](https://www.w3schools.com/python/python_classes.asp)> [Accessed 13 November 2021].