# Brute-forcing The Enigma Cipher

**Patrick Collins – 1900609@uad.ac.uk**

**Introduction to security – CMP110A**

**BSc Ethical Hacking Year 1**

**2019/20**

## Abstract

This paper consists of explaining how the Enigma Cipher encrypts text and its major flaw which can be brute forced using this flaw with the programming language Python.

The aim of this is to show how fast outdated ciphers are brute forced with even the smallest amount of processing power, and how a flaw in a cipher can make it totally useless.

To brute-force the Enigma Cipher a “Crib text” is used which is a decrypted part of the whole ciphertext. A known word in an enigma ciphertext.

In the python script, it asks the user to input their plaintext and a section of this is used as the “Crib Text”. To encrypt the crib text and the plaintext, the user inputs the settings of the Enigma Machine.

Once the crib text is encrypted, a brute-force algorithm is run to find the settings used. Finally, once the settings are found, an Enigma Machine is set up to decrypt the full ciphertext.

After the investigator brute forced the Enigma Cipher, it’s clear that with today’s processing capabilities this cipher can be easily cracked faster than the bombe machine used during WW2. This shows a great improvement in computing machines.

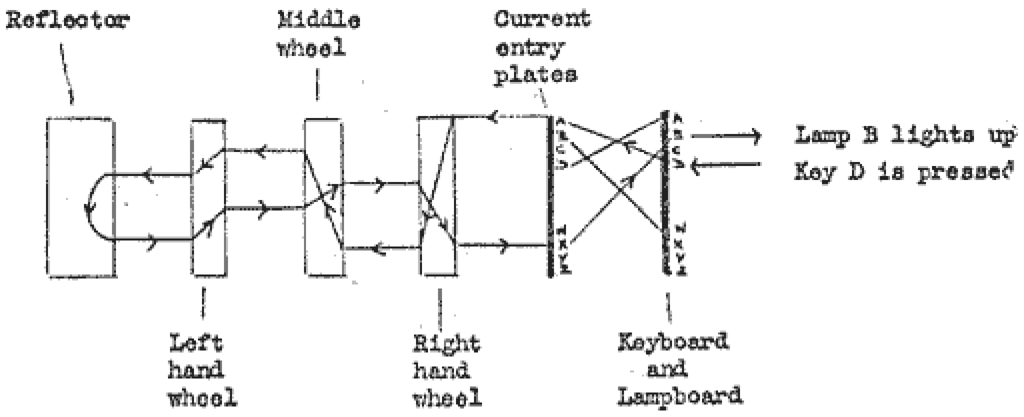
## Contents

## Introduction

### Background

A cipher is a method to either encrypt or decrypt information. The ciphertext is unreadable without decrypting it first, as it is just a random set of letters or characters. The Enigma Cipher is one method to encrypt information and is done using a machine.

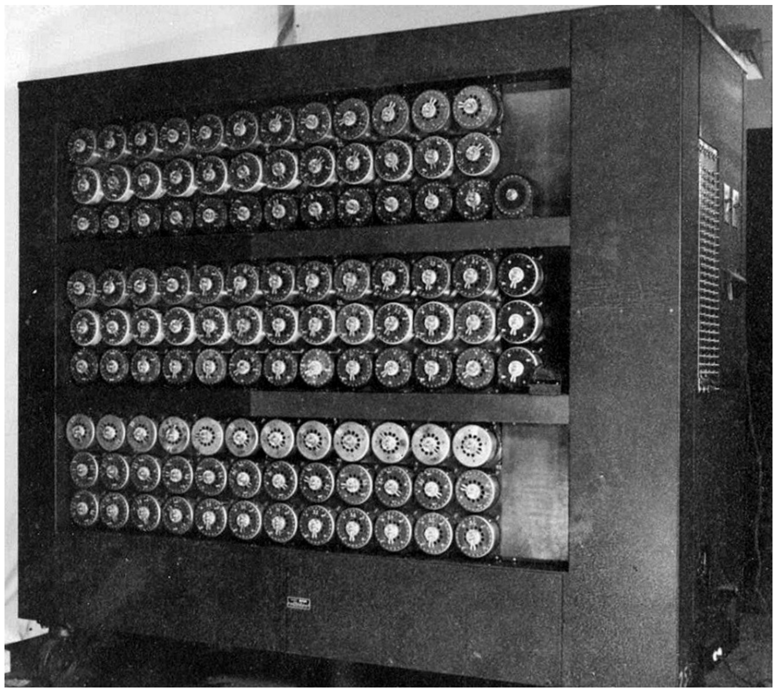
The Enigma Machine was used during world war II by the German military to encrypt communications. How the Enigma machine encrypted its plaintext was to set the machine to a certain position. It had multiple rotors, a plugboard, a reflector and a typewriter-style keyboard to input the text. The number of rotors can be 3 to 5. The plugboard and reflector is how it reverses the letters, therefore, encrypting the text.



*Letter reversing through the plugboard and reflector. Available at: http://www.ellsbury.com/gne/gne-002.htm [Accessed 7 May 2020]*

Back then, the cipher seemed impossible to break. However, the help of Polish cryptographers' previous success on cracking the old models of the Enigma gave Alan Turing's team a head start. The determination of the code breakers at Bletchley Park cracked this cipher and exposed the flaw with the Enigma Cipher (Available at: https://medium.com/lessons-from-history/how-allied-forces-cracked-enigma-code-6f67d3edb65c [Accessed 7 May 2020]).

One of its flaws is that knowing a part of the ciphertext can make it vulnerable to brute-force attacks, which is what the bombe machine did (Available at: https://www.cryptomuseum.com/crypto/bombe/ [Accessed 7 May 2020]). For example, the daily weather reports always started with "Weather" (German: "Wetter") making it easier to crack. The same concept can be used for this project, using a part of the decrypted ciphertext to reveal the full ciphertext.



*Bombe machine, By Unknown author - Set of wartime photos of GC&CS at Bletchley Park, Public Domain, [https://commons.wikimedia.org/w/index.php?curid=72819587](https://commons.wikimedia.org/w/index.php?curid=72819587) [Accessed 7 May 2020]*

It took around 20 minutes for each Rotor Setting (Alexander c 1945, ch 1 para 44. Available at: [http://www.ellsbury.com/gne/gne-012.htm](http://www.ellsbury.com/gne/gne-012.htm) [Accessed 7 May 2020]) . However, Alan's team needed to get the state of Enigma machine before it got changed the next day which made every minute count. This project aims to get through a Rotor in much less time to show the speed of today's processors, even a low-priced one.

*Raspberry Pi 3 Model B Processor*

Available at: [http://www.datasheetcafe.com/bcm2837-datasheet-quad-core-processor-broadcom/](http://www.datasheetcafe.com/bcm2837-datasheet-quad-core-processor-broadcom/) [Accessed May 7, 2020]

## Importance of this topic in Computer Security

Data needs to be protected. Privacy is important, especially in the ever-growing and advancing cyber world. Cybercriminals can use this information to make money, selling it off. This is one reason why Encryption is a very important topic in Computer Security, as it is the forefront of data privacy and security.

Encryption protects the user's information, and there are various methods to do so. From the Enigma Cipher to the Advanced Encryption Standard (AES) we use today, ways to secure information is improving. The issue is, how long will it take to "crack" the encryption, therefore making it obsolete, and a new method is needed?

## Aim

The aims of this project will be:

- To replicate what the bombe device did, brute-forcing the enigma cipher, in modern-day programming languages.
- Attempt to show how far computers have come with the technology used in this project to brute-force the Enigma.
- Reducing the average time of 20 minutes for each Rotor.
- Letting the user encrypt their plaintext, then attempt to brute-force it and display the decrypted message to the user.

## Objectives

- Attempt to code an enigma machine or use an existing Enigma library.
- Use a cheap device, but with reasonable computing power.
- Speeding up the process depends on the hardware, and how well the brute-force attack functions.
- In the code a user input could be requested, this will store the plaintext. This plaintext will be passed into whatever method of the enigma machine.

## Procedure

### Setup

Hardware

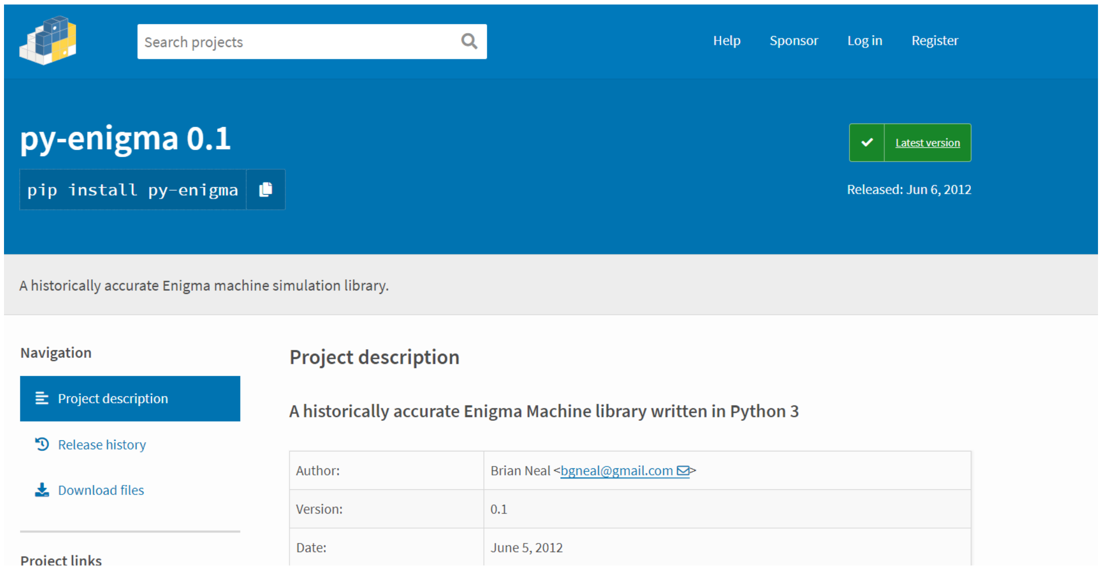
For meeting the objective of reasonable and cheap computing power, a Raspberry Pi 3 Model B is used. This hardware costs around £35 (Raspberry Pi Foundation approved retailer, https://thepihut.com/products/raspberry-pi-3-model-b?src=raspberrypi. See figure 1). This will show one part of advancement in technology, as the Bombe was a large machine whereas a Raspberry Pi is the size of a palm.

Figure 1: Raspberry Pi 3 Model B

Accompanying the Pi, a micro SD card is needed. This is needed for the raspberry pi's operating system.

Software

A Python 3 Enigma library is used to recreate the Enigma Machine in code, meeting one of the outlined objectives. (Author: Brian Neal, https://pypi.org/project/py-enigma/ See figure 2). This great resource is very helpful in simulating the enigma machine to brute-force it. Therefore, the programming language to be used is python 3. Installing this library will come later.



py-enigma 0.1

```
pip install py-enigma
```

Released: Jun 6, 2012

A historically accurate Enigma machine simulation library.

Project description

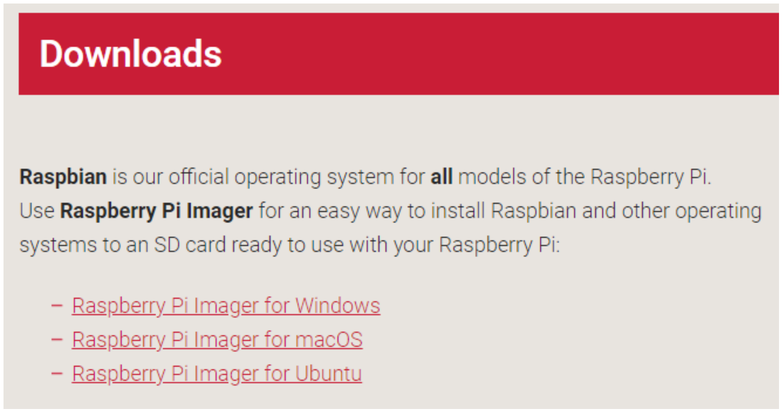
A historically accurate Enigma Machine library written in Python 3

| | |
|---|---|
| Author: | Brian Neal <bgneal@gmail.com> |
| Version: | 0.1 |
| Date: | June 5, 2012 |

*Figure 2: Py-enigma library*

On the Raspberry Pi, the Raspbian operating system is to be used. To image the Raspbian OS to your SD card, simply use the Foundations own Raspberry Pi imager (Available at: https://www.raspberrypi.org/downloads/ Figure 3) Select and download the imager. Once it is downloaded, run the application and install the imager (See figure 4).

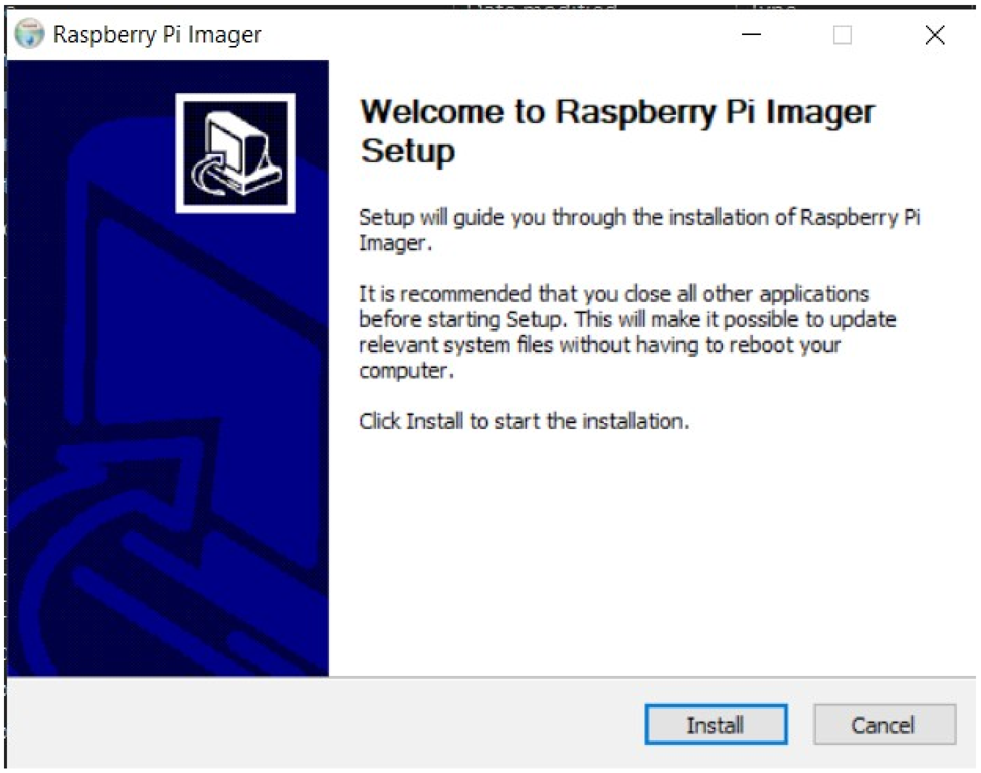
In the imager itself, selecting “Choose OS” will show the Raspbian OS (See figure 5&6). Setup is almost complete, now insert the SD card into the raspberry pi and boot it up. Once booted up, open the terminal to install the final software.


Downloads

**Raspbian** is our official operating system for **all** models of the Raspberry Pi. Use **Raspberry Pi Imager** for an easy way to install Raspbian and other operating systems to an SD card ready to use with your Raspberry Pi:

- Raspberry Pi Imager for Windows
- Raspberry Pi Imager for macOS
- Raspberry Pi Imager for Ubuntu

Figure 3: Raspberry Pi imager download link



Raspberry Pi Imager

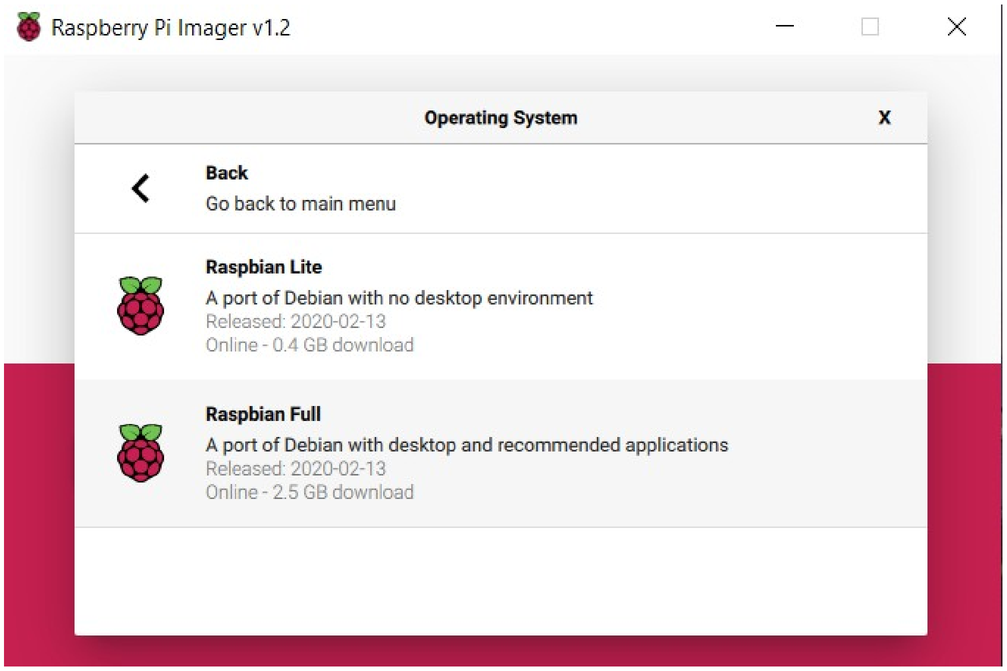
Welcome to Raspberry Pi Imager Setup

Setup will guide you through the installation of Raspberry Pi Imager.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Install to start the installation.

*Figure 4: Raspberry Pi imager setup menu*



Raspberry Pi Imager v1.2

Operating System

Back
Go back to main menu

Raspbian Lite
A port of Debian with no desktop environment
Released: 2020-02-13
Online - 0.4 GB download

Raspbian Full
A port of Debian with desktop and recommended applications
Released: 2020-02-13
Online - 2.5 GB download

*Figure 5: Raspbian OS in menu*

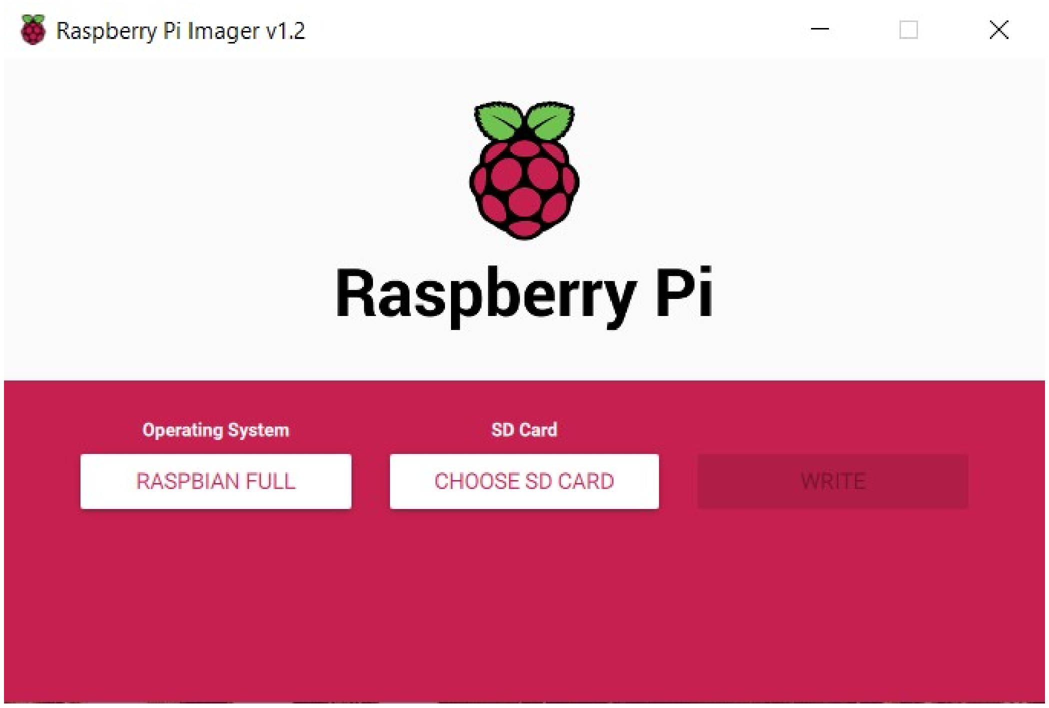
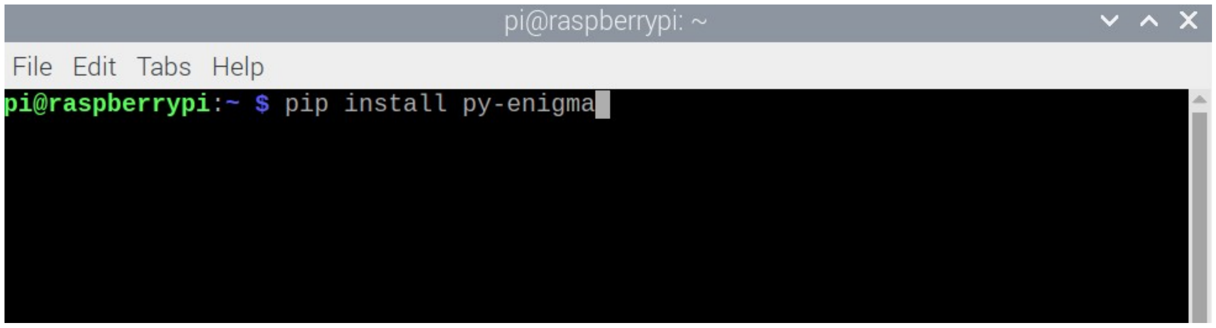


Figure 6: Raspberry Pi Imager

Installing Py-Enigma is very simple from the command line. The command "pip install py-enigma" (See figure 7) will install the python library and enable its use in python programs.



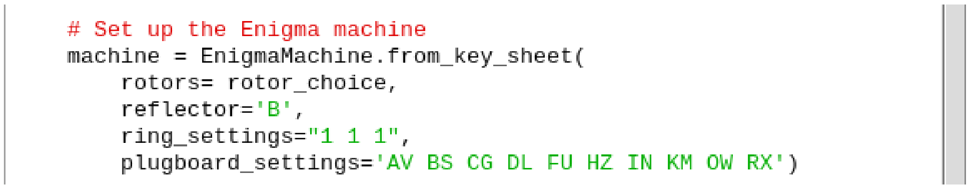
```
pi@raspberrypi:~ $ pip install py-enigma
```

*Figure 7: Installing py-enigma*

## Coding the brute-force attack

To code this part of the program, a tutorial by GCHQ and The Raspberry Pi Foundation (Tutorial available at: https://projects.raspberrypi.org/en/projects/octapi-brute-force-enigma/7) is used for the brute-forcing the enigma.

In Appendix A, it shows a finished brute-force attack coded using the tutorial and py-enigma. You create the enigma machine by including the py-enigma class and creating the machine (See figure 8).



```
# Set up the Enigma machine
machine = EnigmaMachine.from_key_sheet(
    rotors= rotor_choice,
    reflector='B',
    ring_settings="1 1 1",
    plugboard_settings='AV BS CG DL FU HZ IN KM OW RX')
```

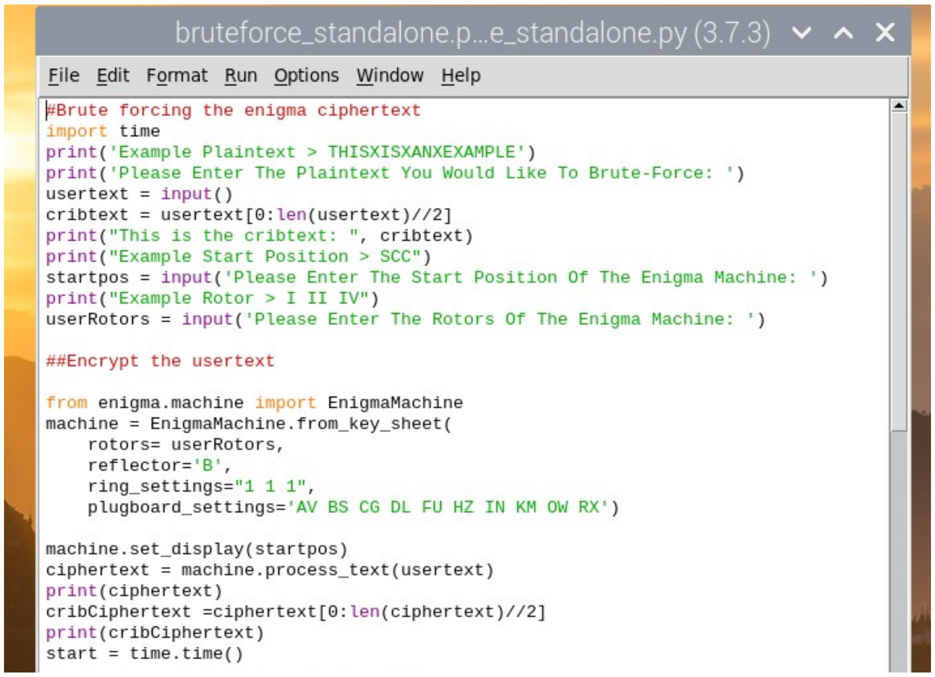
Figure 8: Machine creation

For the moment the Reflector is default to B, as is the ring and plugboard settings. What this section does (See Appendix A) is selects a rotor (From the possible rotor list), start position (Using the alphabet and 3 for loops) and then sets the enigma machine to this setting using "machine.set_display(start_position)".

Furthermore, it attempts to decrypt the crib's ciphertext. If the decrypted crib ciphertext is the same as the cribtext plaintext then the correct setting is found which returns the settings for the user. After the brute-force attack has been coded, the next objectives/aims are to be focused on.

## User input

However, only part of this project's aims will have been met so far. One of its aims is to let the user choose what to encrypt, with the settings they desire. To do this, input statements are needed. The user inputs their plaintext and then the Enigma Settings to encrypt it (See figure 9). However, to encrypt what the user has entered a new enigma machine needs to be created with these settings. The settings that the user chooses will be the brute-force attack's goal to find. (Also see appendix B for more information).


bruteforce_standalone.p...e_standalone.py (3.7.3)
File Edit Format Run Options Window Help

```
#Brute forcing the enigma ciphertext
import time
print('Example Plaintext > THISXISXANXEXAMPLE')
print('Please Enter The Plaintext You Would Like To Brute-Force: ')
usertext = input()
cribtext = usertext[0:len(usertext)//2]
print("This is the cribtext: ", cribtext)
print("Example Start Position > SCC")
startpos = input('Please Enter The Start Position Of The Enigma Machine: ')
print("Example Rotor > I II IV")
userRotors = input('Please Enter The Rotors Of The Enigma Machine: ')

##Encrypt the usertext

from enigma.machine import EnigmaMachine
machine = EnigmaMachine.from_key_sheet(
    rotors= userRotors,
    reflector='B',
    ring_settings="1 1 1",
    plugboard_settings='AV BS CG DL FU HZ IN KM OW RX')

machine.set_display(startpos)
ciphertext = machine.process_text(usertext)
print(ciphertext)
cribCiphertext =ciphertext[0:len(ciphertext)//2]
print(cribCiphertext)
start = time.time()
```

Figure 9: User Input and machine creation

## Decrypting user text

With the settings found after the brute-force attempt, it's now possible to decrypt the full message that the user entered.

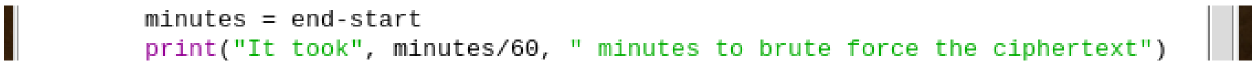
To decrypt the full message, the state of the enigma machine needs to be set to the valid settings found. Once the state has been set to the correct position the full message will be decrypted and displayed to the user. (See Appendix D, figure 3).

After the full message has been decrypted the brute-force attack has been successful. Now it's just timing how long the attack takes.

## Timing the brute-force attempt

To time the brute-force attempt, time.time() can be used. You will see this in Figure 9 with "start = time.time()". After the user's plaintext is encrypted the brute-force attempt begins, which is why the time must start here. Once the settings are found, the time stops with "end = time.time()".

Converting the time to minutes is done by subtracting the two and then dividing by 60 (See figure 10 for the calculation). This will give an accurate and understandable timestamp of how long the program took to brute-force their enigma settings.



```
        minutes = end-start
        print("It took", minutes/60, " minutes to brute force the ciphertext")
```

*Figure 10: calculation of seconds to minutes.*

## Results

The program is complete, now it's time to run it. Running the program, the user decided to choose a rotor position not too far down the list of possible rotors. You should do the same for testing.  For example, I II III.

### Entering text and enigma settings

The user entered "THEXALLIESXAREXADVANCING" and got correctly split into a Cribtext displaying it to the user. Next, a Start Position "WRZ" is entered with Rotor Position "I II IV" which is the second possible rotor in the list. (See Appendix E, figure 1).

### Attempts of brute-force attack

Letting the program run and attempt to crack the settings entered, the lines got up to the 2000s. Meaning this is 2000 attempts so far of the one Rotor Position and multiple Start Positions. (See Appendix E, figure 2).

### Enigma settings cracked

Once ended, the program had an incredible 32,937 attempts. This is just the second Rotor in the list. Time calculation also worked showing it took 7 minutes for the two Rotors. Finally, the full text that the user entered at the beginning is successfully decrypted. This result can be seen in Appendix E, figure 3.

## Discussion

### General Discussion

The most significant result of this project is the time of the brute-force attack. Back with Turing’s bombe it took 20 minutes for each rotor. In the example shown in Appendix E, 2 rotors had to be checked before the settings were found. In WW2 this would have taken around 40 minutes.

However, with the speed of today’s single processor, it took 7 minutes. Significant time reduction. Also, keep in mind this is also done on the miniature raspberry pi in comparison to the size of the bombe used in WW2. This meets two of the aims, showing the advancement in technology and reducing time of the brute-force attack on the enigma cipher. The Raspberry Pi minicomputer can do what the bombe did, but better and faster despite being smaller.

After the investigator successfully run the brute-force attack, another project aim is met. Replicating what the bombe device did seemed like a big task for this project, and the steps needed to take not so clear. Now, the program successfully imitates what the bombe device did. Also, with the help of py-enigma it has been possible to do it on a modern-day programming language.

Finally, an important aim to have user interaction is achieved. The user can enter any plaintext they want. Even further, the user can set some settings of the enigma machine to their liking, which was out of this aim’s scope, improving the aim further.

### Countermeasures

To counter the flaw of the enigma cipher, the TypeX was used by the British. This meant the methods used to brute-force the enigma cipher cannot be used on this improved machine. It fixed the flaws of the enigma making it a more secure communication method. (Available at: https://www.cryptomuseum.com/crypto/uk/typex/ [Accessed May 8, 2020)

## Conclusions

In conclusion, all of the project aims have been achieved. Some aims taking the project beyond expectation. This project is a good insight into cryptography and brute-forcing ciphers. Finally, a great experience to see how far computers have evolved since WW2 with the help of Alan Turing and his proposition of “Turing Machine”(Available at: http://www.turingarchive.org/viewer/?id=466&title=01d [Accessed May 8, 2020]).

## Future Work

If the project had more time and resources, a bigger scope could be achieved. Such as an addition to the brute-force section for the Reflector, Plugboard and Ring Settings. This would allow the user to fully enter the settings of the enigma machine to their liking.

However, doing so would increase the time significantly. To combat this, 8 raspberry pi’s would be needed. Just as more bombe machines needed to be created to decrypt faster.

## References

(n.d.). Retrieved May 7, 2020, from https://www.cryptomuseum.com/crypto/bombe/

(n.d.). Retrieved May 7, 2020, from https://medium.com/lessons-from-history/how-allied-forces-cracked-enigma-code-6f67d3edb65c

c, A. (n.d.). *Reflector drawing*. Retrieved May 7, 2020, from http://www.ellsbury.com/gne/gne-002.htm

c, A. (n.d.). *Time For One Rotor*. Retrieved May 7, 2020, from http://www.ellsbury.com/gne/gne-012.htm

GCHQ, R. P. (22, November 2017). *Enigma Brute Force Tutorial*. Retrieved May 7, 2020, from https://projects.raspberrypi.org/en/projects/octapi-brute-force-enigma/7

*Image Of Bombe Machine*. (n.d.). Retrieved May 7, 2020, from https://commons.wikimedia.org/w/index.php?curid=72819587

Lycett, A. (n.d.). Retrieved May 5, 2020, from http://www.bbc.co.uk/history/worldwars/wwtwo/enigma_01.shtml

*Raspberry Pi 3*. (n.d.). Retrieved May 7, 2020, from https://thepihut.com/products/raspberry-pi-3-model-b?src=raspberrypi

*Raspberry Pi 3 Model B Processor*. (n.d.). Retrieved May 7, 2020, from http://www.datasheetcafe.com/bcm2837-datasheet-quad-core-processor-broadcom/

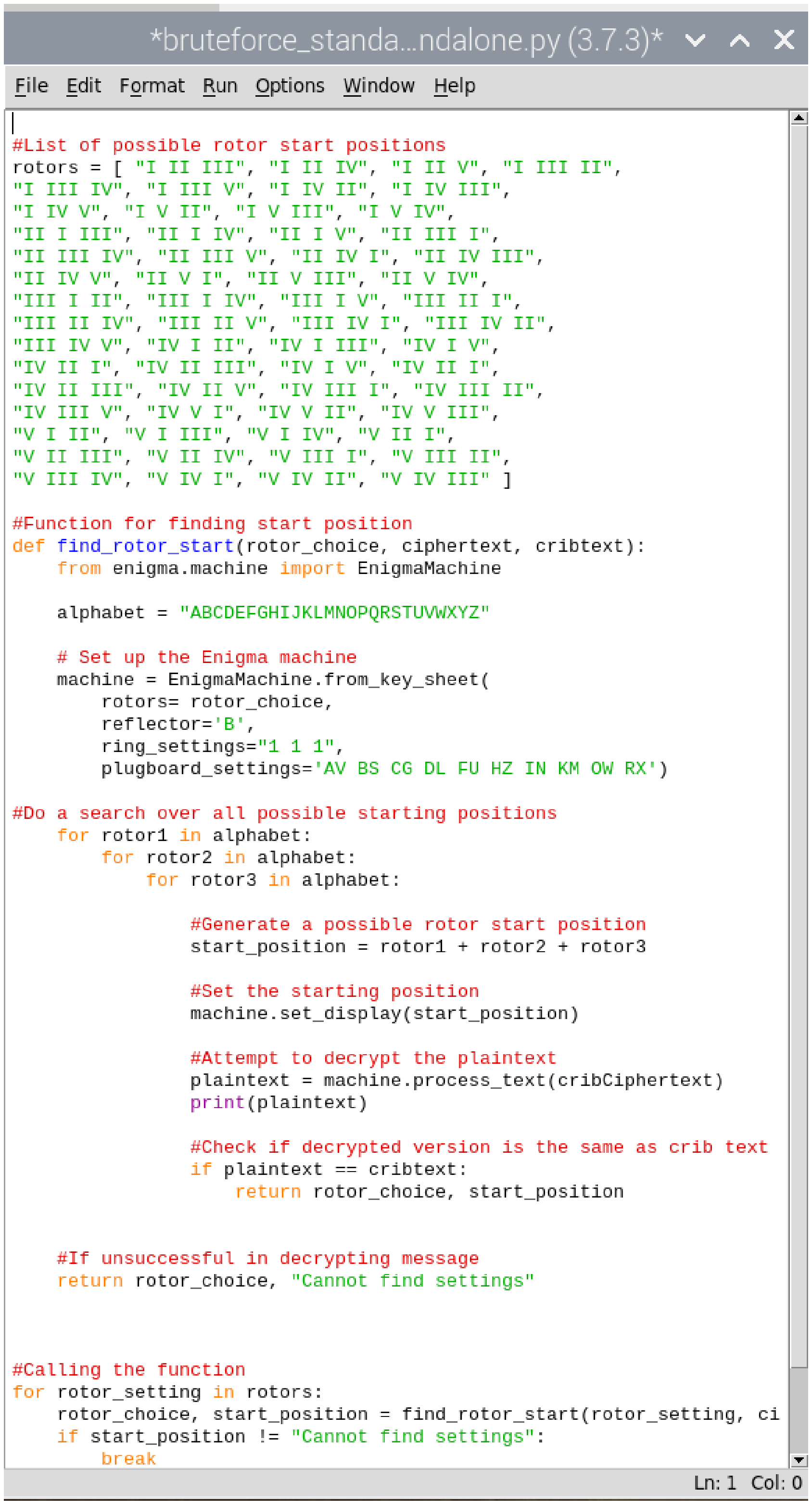
*Raspberry Pi Imager* . (n.d.). Retrieved May 8, 2020, from https://www.raspberrypi.org/downloads/

Turing, A. M. (n.d.). *Turing Machine*. Retrieved May 8, 2020, from http://www.turingarchive.org/viewer/?id=466&title=01d

*TypeX*. (n.d.). Retrieved May 8, 2020, from https://www.cryptomuseum.com/crypto/uk/typex/

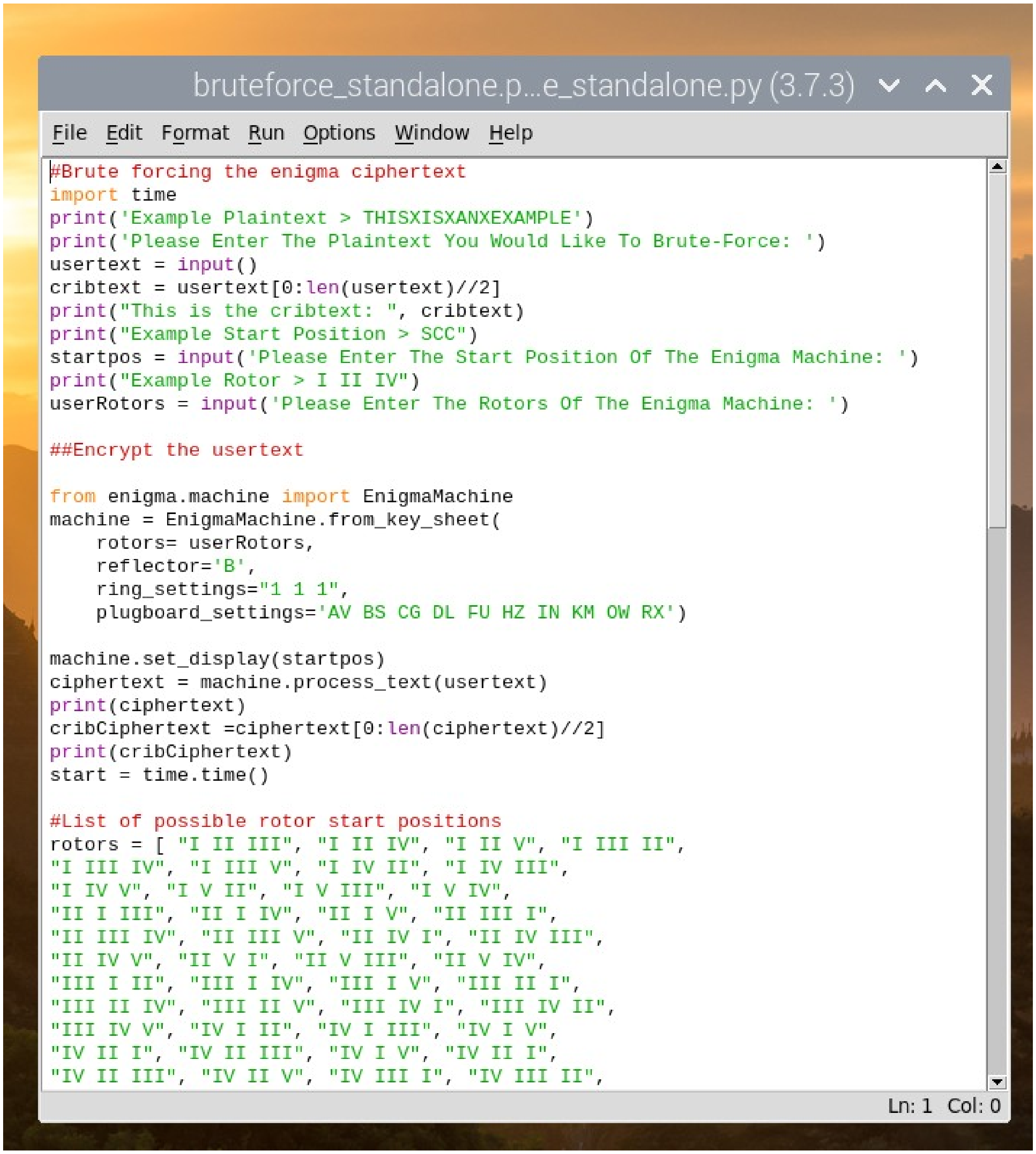
## Appendices

## Appendix A: Brute-force section


*bruteforce_standa...ndalone.py (3.7.3)*
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```
#List of possible rotor start positions
rotors = [ "I II III", "I II IV", "I II V", "I III II",
"I III IV", "I III V", "I IV II", "I IV III",
"I IV V", "I V II", "I V III", "I V IV",
"II I III", "II I IV", "II I V", "II III I",
"II III IV", "II III V", "II IV I", "II IV III",
"II IV V", "II V I", "II V III", "II V IV",
"III I II", "III I IV", "III I V", "III II I",
"III II IV", "III II V", "III IV I", "III IV II",
"III IV V", "IV I II", "IV I III", "IV I V",
"IV II I", "IV II III", "IV I V", "IV II I",
"IV II III", "IV II V", "IV III I", "IV III II",
"IV III V", "IV V I", "IV V II", "IV V III",
"V I II", "V I III", "V I IV", "V II I",
"V II III", "V II IV", "V III I", "V III II",
"V III IV", "V IV I", "V IV II", "V IV III" ]

#Function for finding start position
def find_rotor_start(rotor_choice, ciphertext, cribtext):
    from enigma.machine import EnigmaMachine

    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    # Set up the Enigma machine
    machine = EnigmaMachine.from_key_sheet(
        rotors= rotor_choice,
        reflector='B',
        ring_settings="1 1 1",
        plugboard_settings='AV BS CG DL FU HZ IN KM OW RX')

#Do a search over all possible starting positions
    for rotor1 in alphabet:
        for rotor2 in alphabet:
            for rotor3 in alphabet:

                #Generate a possible rotor start position
                start_position = rotor1 + rotor2 + rotor3

                #Set the starting position
                machine.set_display(start_position)

                #Attempt to decrypt the plaintext
                plaintext = machine.process_text(cribCiphertext)
                print(plaintext)

                #Check if decrypted version is the same as crib text
                if plaintext == cribtext:
                    return rotor_choice, start_position


    #If unsuccessful in decrypting message
    return rotor_choice, "Cannot find settings"



#Calling the function
for rotor_setting in rotors:
    rotor_choice, start_position = find_rotor_start(rotor_setting, ci
    if start_position != "Cannot find settings":
        break
```

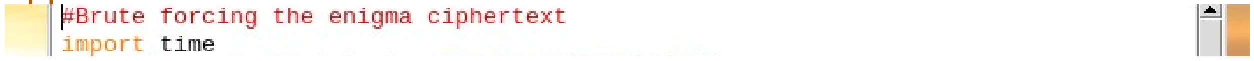
## Appendix B: User input with machine creation


bruteforce_standalone.p...e_standalone.py (3.7.3)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```
#Brute forcing the enigma ciphertext
import time
print('Example Plaintext > THISXISXANXEXAMPLE')
print('Please Enter The Plaintext You Would Like To Brute-Force: ')
usertext = input()
cribtext = usertext[0:len(usertext)//2]
print("This is the cribtext: ", cribtext)
print("Example Start Position > SCC")
startpos = input('Please Enter The Start Position Of The Enigma Machine: ')
print("Example Rotor > I II IV")
userRotors = input('Please Enter The Rotors Of The Enigma Machine: ')

##Encrypt the usertext

from enigma.machine import EnigmaMachine
machine = EnigmaMachine.from_key_sheet(
    rotors= userRotors,
    reflector='B',
    ring_settings="1 1 1",
    plugboard_settings='AV BS CG DL FU HZ IN KM OW RX')

machine.set_display(startpos)
ciphertext = machine.process_text(usertext)
print(ciphertext)
cribCiphertext =ciphertext[0:len(ciphertext)//2]
print(cribCiphertext)
start = time.time()

#List of possible rotor start positions
rotors = [ "I II III", "I II IV", "I II V", "I III II",
"I III IV", "I III V", "I IV II", "I IV III",
"I IV V", "I V II", "I V III", "I V IV",
"II I III", "II I IV", "II I V", "II III I",
"II III IV", "II III V", "II IV I", "II IV III",
"II IV V", "II V I", "II V III", "II V IV",
"III I II", "III I IV", "III I V", "III II I",
"III II IV", "III II V", "III IV I", "III IV II",
"III IV V", "IV I II", "IV I III", "IV I V",
"IV II I", "IV II III", "IV I V", "IV II I",
"IV II III", "IV II V", "IV III I", "IV III II",
```

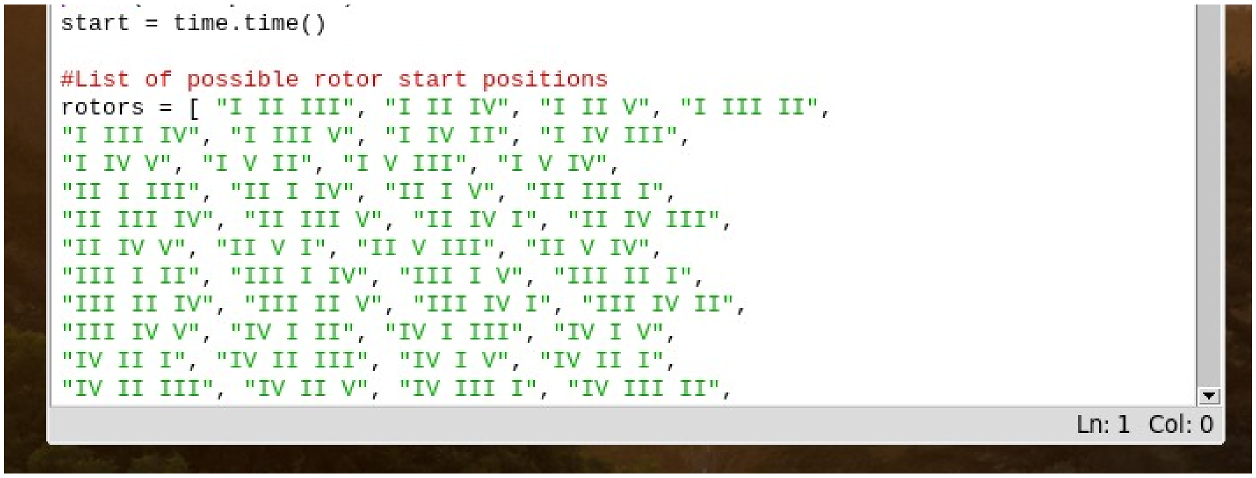
*Figure 1: User's text is split into 2 for the cribtext. Seen by "usertext[0:len(usertext)//2]". This will be used for the brute-force attack.*

## Appendix C: Time calculation



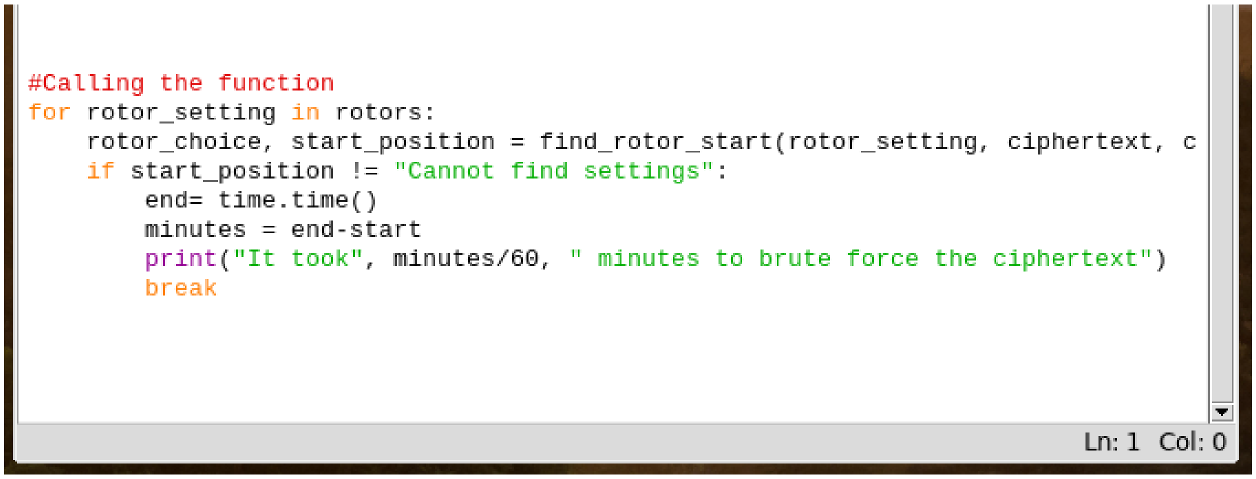
```
#Brute forcing the enigma ciphertext
import time
```

*Figure 1: Importing time to use the time.time() function in order to begin calculating length of attack.*



```
start = time.time()

#List of possible rotor start positions
rotors = [ "I II III", "I II IV", "I II V", "I III II",
"I III IV", "I III V", "I IV II", "I IV III",
"I IV V", "I V II", "I V III", "I V IV",
"II I III", "II I IV", "II I V", "II III I",
"II III IV", "II III V", "II IV I", "II IV III",
"II IV V", "II V I", "II V III", "II V IV",
"III I II", "III I IV", "III I V", "III II I",
"III II IV", "III II V", "III IV I", "III IV II",
"III IV V", "IV I II", "IV I III", "IV I V",
"IV II I", "IV II III", "IV I V", "IV II I",
"IV II III", "IV II V", "IV III I", "IV III II",
```

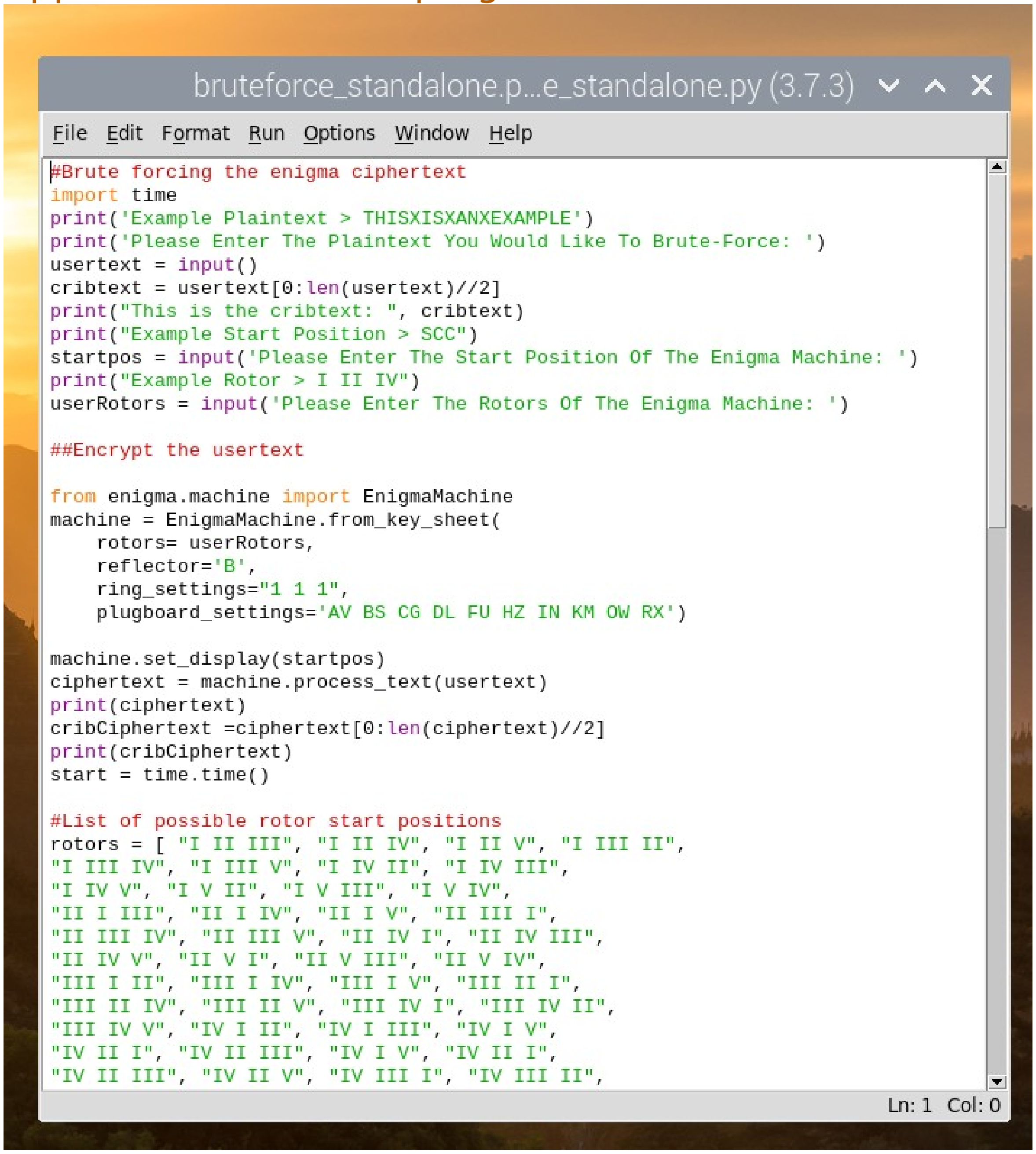
*Figure 2: Start of time counter- after user input and machine creation finished*



```
#Calling the function
for rotor_setting in rotors:
    rotor_choice, start_position = find_rotor_start(rotor_setting, ciphertext, c
    if start_position != "Cannot find settings":
        end= time.time()
        minutes = end-start
        print("It took", minutes/60, " minutes to brute force the ciphertext")
        break
```

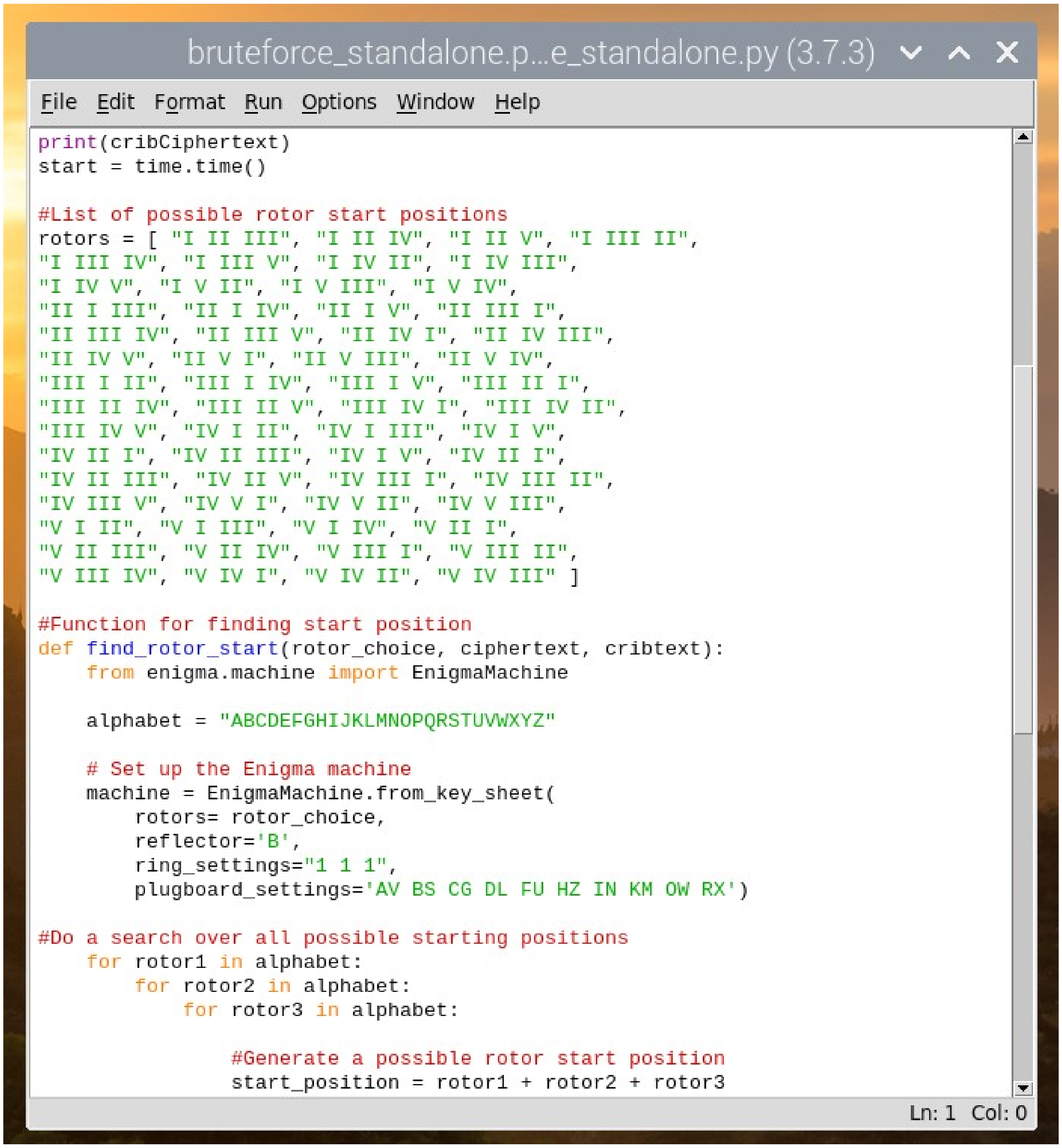
*Figure 3: End of time counter- once attack has found valid settings counting the time stops. Then a calculation converts the time from seconds to minutes.*

## Appendix D: Finished program


bruteforce_standalone.p...e_standalone.py (3.7.3)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

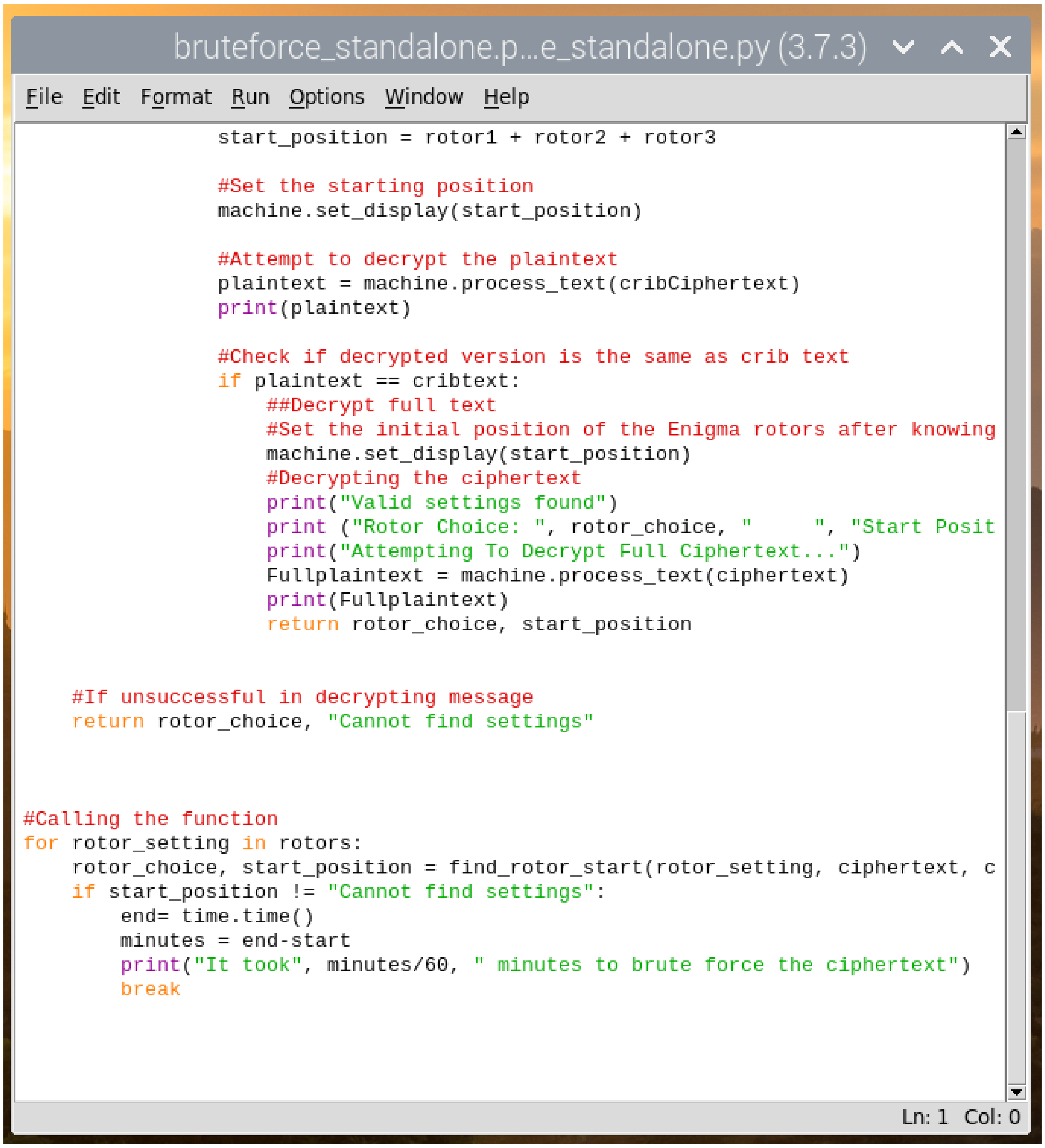
```
#Brute forcing the enigma ciphertext
import time
print('Example Plaintext > THISXISXANXEXAMPLE')
print('Please Enter The Plaintext You Would Like To Brute-Force: ')
usertext = input()
cribtext = usertext[0:len(usertext)//2]
print("This is the cribtext: ", cribtext)
print("Example Start Position > SCC")
startpos = input('Please Enter The Start Position Of The Enigma Machine: ')
print("Example Rotor > I II IV")
userRotors = input('Please Enter The Rotors Of The Enigma Machine: ')

##Encrypt the usertext

from enigma.machine import EnigmaMachine
machine = EnigmaMachine.from_key_sheet(
    rotors= userRotors,
    reflector='B',
    ring_settings="1 1 1",
    plugboard_settings='AV BS CG DL FU HZ IN KM OW RX')

machine.set_display(startpos)
ciphertext = machine.process_text(usertext)
print(ciphertext)
cribCiphertext =ciphertext[0:len(ciphertext)//2]
print(cribCiphertext)
start = time.time()

#List of possible rotor start positions
rotors = [ "I II III", "I II IV", "I II V", "I III II",
"I III IV", "I III V", "I IV II", "I IV III",
"I IV V", "I V II", "I V III", "I V IV",
"II I III", "II I IV", "II I V", "II III I",
"II III IV", "II III V", "II IV I", "II IV III",
"II IV V", "II V I", "II V III", "II V IV",
"III I II", "III I IV", "III I V", "III II I",
"III II IV", "III II V", "III IV I", "III IV II",
"III IV V", "IV I II", "IV I III", "IV I V",
"IV II I", "IV II III", "IV I V", "IV II I",
"IV II III", "IV II V", "IV III I", "IV III II",
```

*Figure 1: User input section of the program*


bruteforce_standalone.p...e_standalone.py (3.7.3)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```python
print(cribCiphertext)
start = time.time()

#List of possible rotor start positions
rotors = [ "I II III", "I II IV", "I II V", "I III II",
"I III IV", "I III V", "I IV II", "I IV III",
"I IV V", "I V II", "I V III", "I V IV",
"II I III", "II I IV", "II I V", "II III I",
"II III IV", "II III V", "II IV I", "II IV III",
"II IV V", "II V I", "II V III", "II V IV",
"III I II", "III I IV", "III I V", "III II I",
"III II IV", "III II V", "III IV I", "III IV II",
"III IV V", "IV I II", "IV I III", "IV I V",
"IV II I", "IV II III", "IV I V", "IV II I",
"IV II III", "IV II V", "IV III I", "IV III II",
"IV III V", "IV V I", "IV V II", "IV V III",
"V I II", "V I III", "V I IV", "V II I",
"V II III", "V II IV", "V III I", "V III II",
"V III IV", "V IV I", "V IV II", "V IV III" ]

#Function for finding start position
def find_rotor_start(rotor_choice, ciphertext, cribtext):
    from enigma.machine import EnigmaMachine

    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    # Set up the Enigma machine
    machine = EnigmaMachine.from_key_sheet(
        rotors= rotor_choice,
        reflector='B',
        ring_settings="1 1 1",
        plugboard_settings='AV BS CG DL FU HZ IN KM OW RX')

#Do a search over all possible starting positions
    for rotor1 in alphabet:
        for rotor2 in alphabet:
            for rotor3 in alphabet:

                #Generate a possible rotor start position
                start_position = rotor1 + rotor2 + rotor3
```

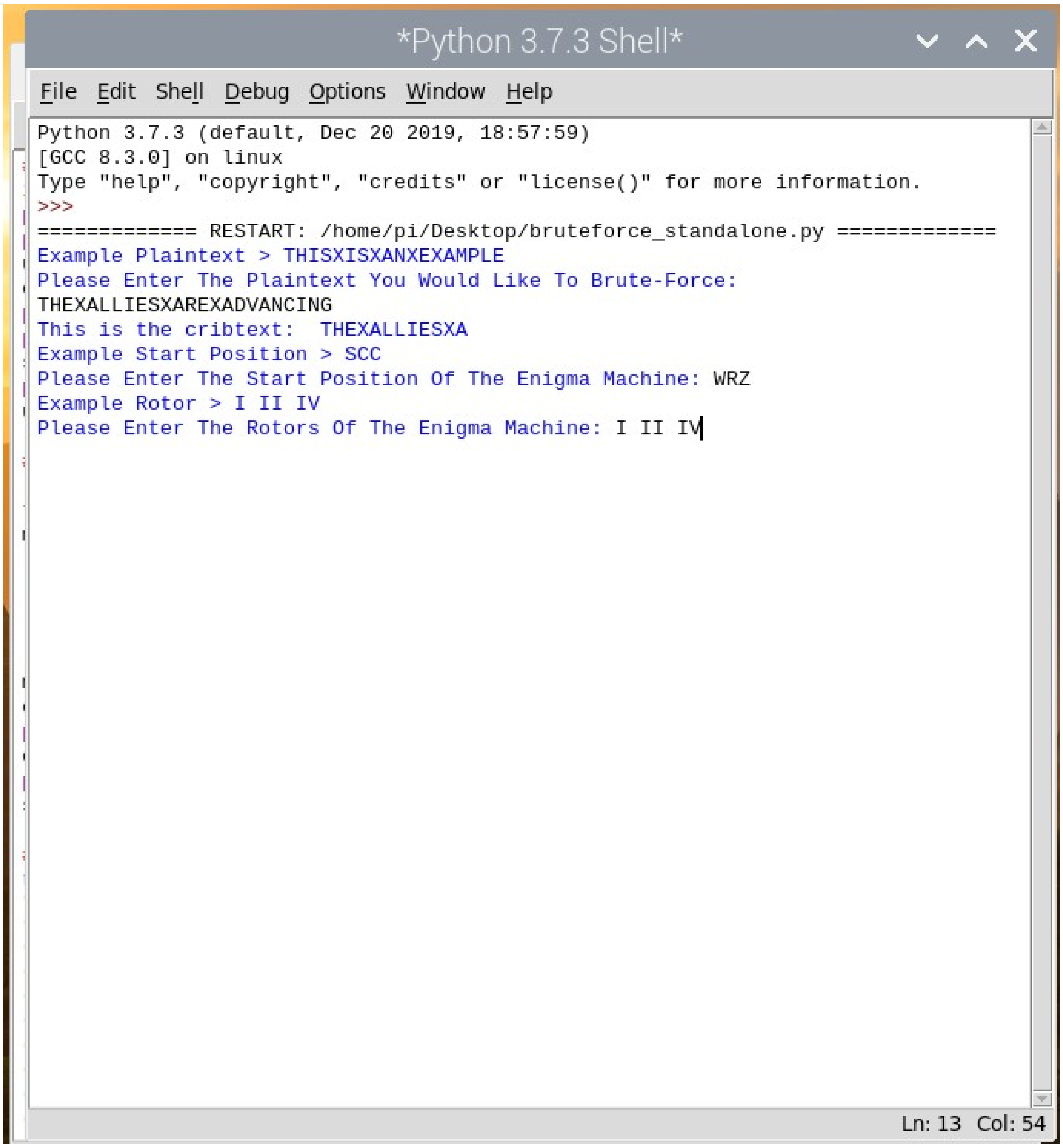
*Figure 2: brute-force section of the program*


bruteforce_standalone.p...e_standalone.py (3.7.3)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```
                start_position = rotor1 + rotor2 + rotor3

                #Set the starting position
                machine.set_display(start_position)

                #Attempt to decrypt the plaintext
                plaintext = machine.process_text(cribCiphertext)
                print(plaintext)

                #Check if decrypted version is the same as crib text
                if plaintext == cribtext:
                    ##Decrypt full text
                    #Set the initial position of the Enigma rotors after knowing
                    machine.set_display(start_position)
                    #Decrypting the ciphertext
                    print("Valid settings found")
                    print ("Rotor Choice: ", rotor_choice, "      ", "Start Posit
                    print("Attempting To Decrypt Full Ciphertext...")
                    Fullplaintext = machine.process_text(ciphertext)
                    print(Fullplaintext)
                    return rotor_choice, start_position


    #If unsuccessful in decrypting message
    return rotor_choice, "Cannot find settings"



#Calling the function
for rotor_setting in rotors:
    rotor_choice, start_position = find_rotor_start(rotor_setting, ciphertext, c
    if start_position != "Cannot find settings":
        end= time.time()
        minutes = end-start
        print("It took", minutes/60, " minutes to brute force the ciphertext")
        break
```

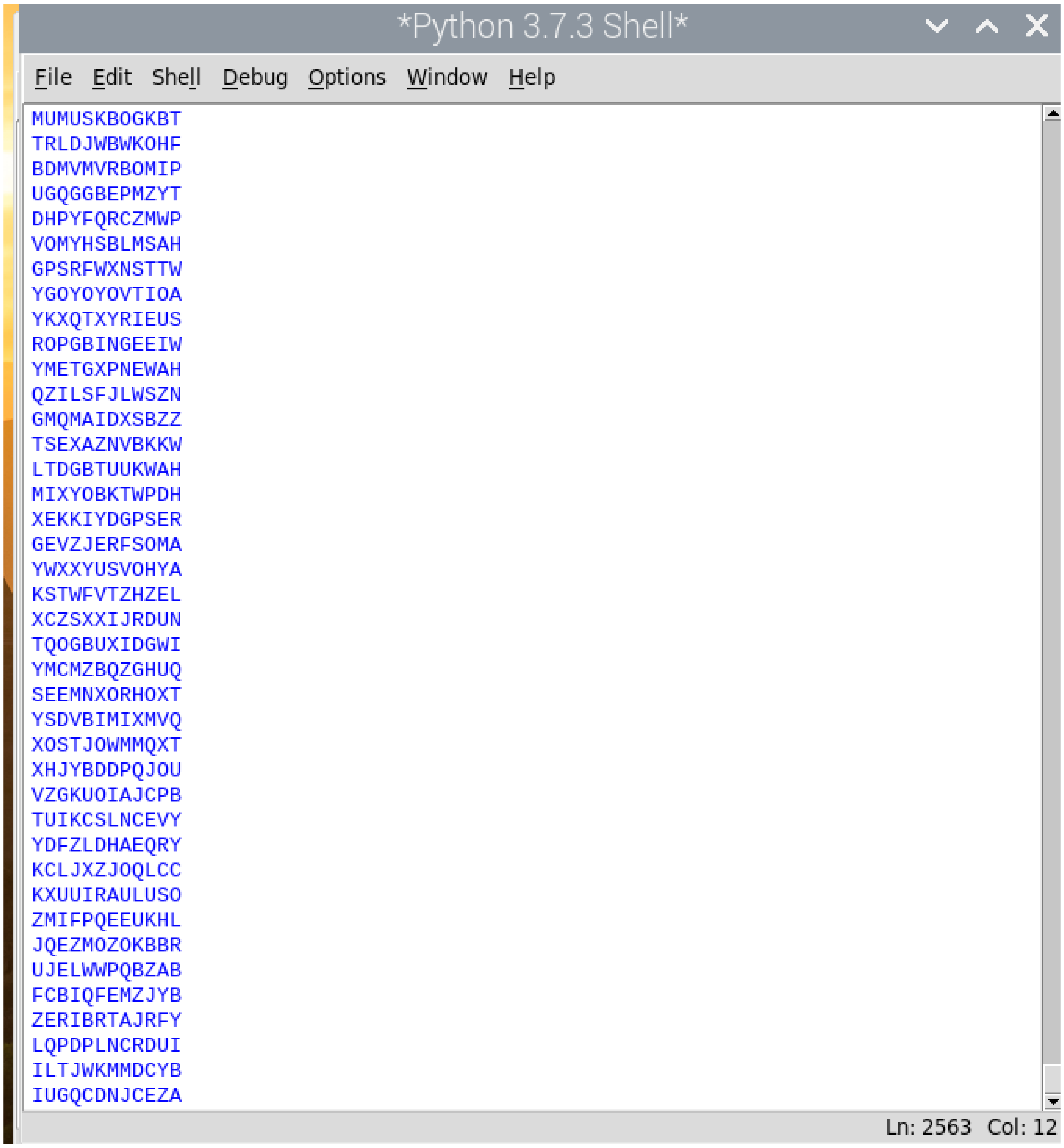
*Figure 3: Paying attention to "checking if decrypted version is the same as the crib text", the enigma machine state is set to the valid settings found. Then the full ciphertext that the user entered is decrypted displaying it in plaintext.*

## Appendix E: Running the program



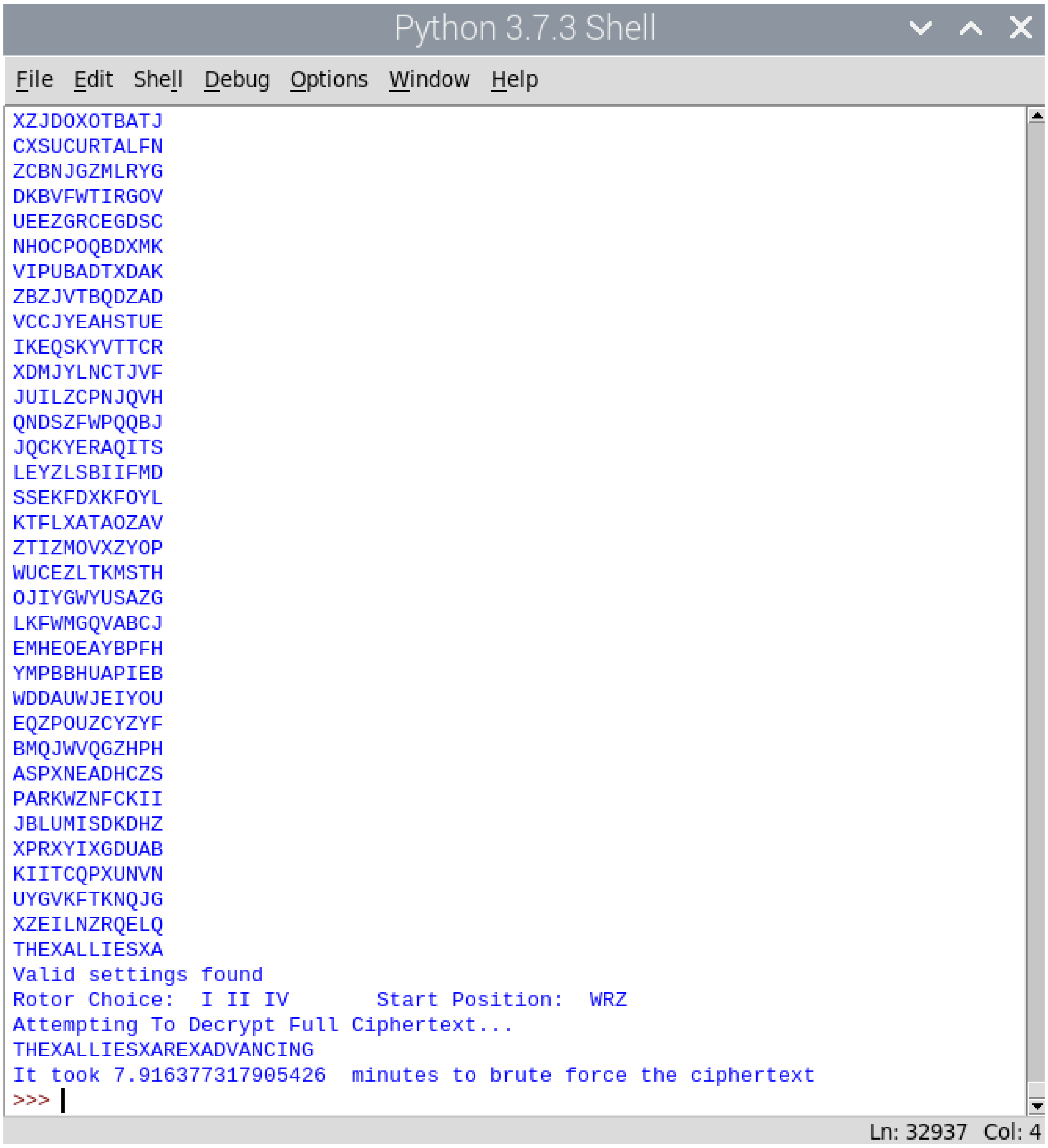
```
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: /home/pi/Desktop/bruteforce_standalone.py =============
Example Plaintext > THISXISXANXEXAMPLE
Please Enter The Plaintext You Would Like To Brute-Force:
THEXALLIESXAREXADVANCING
This is the cribtext:  THEXALLIESXA
Example Start Position > SCC
Please Enter The Start Position Of The Enigma Machine: WRZ
Example Rotor > I II IV
Please Enter The Rotors Of The Enigma Machine: I II IV
```

*Figure 1: User entering text to encrypt and choosing settings of the enigma machine*



```
MUMUSKBOGKBT
TRLDJWBWKOHF
BDMVMVRBOMIP
UGQGGBEPMZYT
DHPYFQRCZMWP
VOMYHSBLMSAH
GPSRFWXNSTTW
YGOYOYOVTIOA
YKXQTXYRIEUS
ROPGBINGEEIW
YMETGXPNEWAH
QZILSFJLWSZN
GMQMAIDXSBZZ
TSEXAZNVBKKW
LTDGBTUUKWAH
MIXYOBKTWPDH
XEKKIYDGPSER
GEVZJERFSOMA
YWXXYUSVOHYA
KSTWFVTZHZEL
XCZSXXIJRDUN
TQOGBUXIDGWI
YMCMZBQZGHUQ
SEEMNXORHOXT
YSDVBIMIXMVQ
XOSTJOWMMQXT
XHJYBDDPQJOU
VZGKUOIAJCPB
TUIKCSLNCEVY
YDFZLDHAEQRY
KCLJXZJOQLCC
KXUUIRAULUSO
ZMIFPQEEUKHL
JQEZMOZOKBBR
UJELWWPQBZAB
FCBIQFEMZJYB
ZERIBRTAJRFY
LQPDPLNCRDUI
ILTJWKMMDCYB
IUGQCDNJCEZA
```

*Figure 2: The attempts to decrypt the Cribs Ciphertext and find valid settings*



Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
XZJDOXOTBATJ
CXSUCURTALFN
ZCBNJGZMLRYG
DKBVFWTIRGOV
UEEZGRCEGDSC
NHOCPOQBDXMK
VIPUBADTXDAK
ZBZJVTBQDZAD
VCCJYEAHSTUE
IKEQSKYVTTCR
XDMJYLNCTJVF
JUILZCPNJQVH
QNDSZFWPQQBJ
JQCKYERAQITS
LEYZLSBIIFMD
SSEKFDXKFOYL
KTFLXATAOZAV
ZTIZMOVXZYOP
WUCEZLTKMSTH
OJIYGWYUSAZG
LKFWMGQVABCJ
EMHEOEAYBPFH
YMPBBHUAPIEB
WDDAUWJEIYOU
EQZPOUZCYZYF
BMQJWVQGZHPH
ASPXNEADHCZS
PARKWZNFCKII
JBLUMISDKDHZ
XPRXYIXGDUAB
KIITCQPXUNVN
UYGVKFTKNQJG
XZEILNZRQELQ
THEXALLIESXA
Valid settings found
Rotor Choice:  I II IV       Start Position:  WRZ
Attempting To Decrypt Full Ciphertext...
THEXALLIESXAREXADVANCING
It took 7.916377317905426  minutes to brute force the ciphertext
>>> 
```

Ln: 32937   Col: 4

*Figure 3: After 7 minutes the settings of the enigma machine, that the user entered, is found. A new enigma machine is created with these settings and the full ciphertext is decrypted, revealing the full message that the user entered.*