All Day Lollipop Productions – Enigma Machine Simulation

*I put this material together for the fun of it. Anyone is free to use it in any manner they wish.
No guarantees are made.*

Expertise comes with study and practice, practice, practice, …

Corporate Headquarters

CTO

Copyright

Table of Contents

Introduction

I put this material together for the fun of it. Anyone is free to use it in any manner they wish. No guarantees are made.

I enjoy programming and have been doing it for over 40 years.

I think everyone should know a little about programming; it is the future required literacy. Of course, not everyone can be an expert. However, everyone should know a little.

I created programming problems over several years. They currently exist as web pages, but I plan to convert them to something more portable.

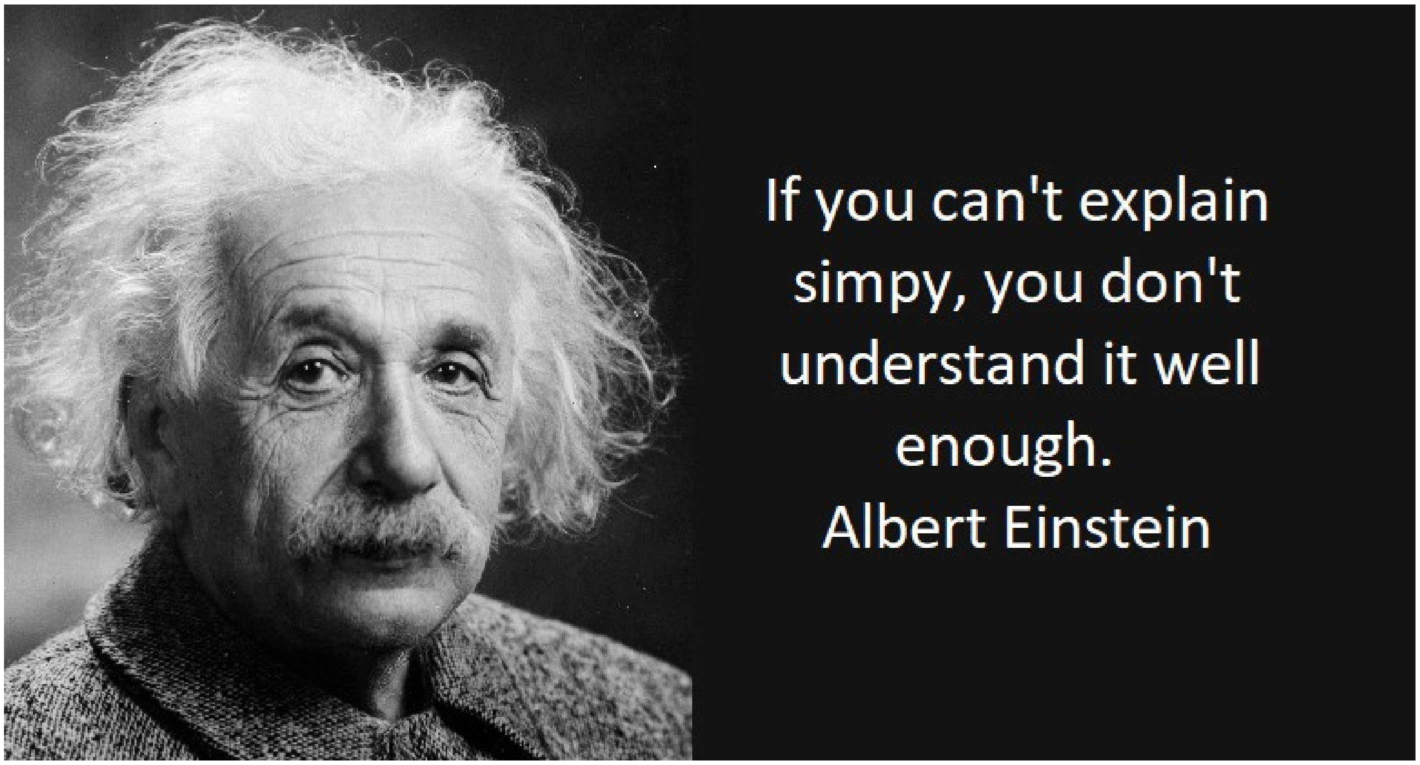
The Enigma Machine Simulation is one of the projects. I admit, one of the more interesting ones.

I apologize in advance for the errors, typos, etc. Just think of them as an exercise for the student or an obstacle to overcome.

Best of luck and I hope you enjoy the projects.
The Author

Why do The Projects

All of the projects collected have been done before. The purpose of the projects is to learn and practice programming. You will do things that has been done before so that, in the future, you know enough to do things no one has done before.


If you can't explain simpy, you don't understand it well enough.
Albert Einstein

A Small Rant

In the past it was said that computers will be able handle anything in the world. They will become as smart as we are. Now we see roads, culture, the way we do things, etc. being modified to fit the computer's need. Remember, general AI is just 50 years away and it has been that way for the last 50 years.

Something You Need To Know

I put this material together for the fun of it. Anyone is free to use it in any manner they wish. No guarantees are made.

"If I had an hour to solve a problem, I'd spend 55 minutes thinking about the problem and five minutes thinking about solutions."
(Albert Einstein)

What you need to know about programming is **"Algorithms + Data Structures = Programs"**. You structure data so it can be efficiently manipulated by code (algorithms). You create code (algorithms) so it can efficiently manipulate data. They are related and should be balanced. One is not more important than the other.

Note: "Algorithms + Data Structures = Programs" is the title of a 1976 book written by Niklaus Wirth. This is where the quote comes from.

Another way to look at it is every program, every chunk of code is **input** → **do-something** → **output**. It is not mysterious.

Baby Code

During my working career I have been accused of writing "baby code" and wasting space and time making it look good. This is a response to those allegations.

But first a story. One day a young programmer came up to me and said she had been given someone else's code to modify. After spending several hours trying to figure out how it worked, she gave up. She then went to the original programmer and asked for help understanding what the code did. He tried for half an hour and said he would have to get back to her. Several hours later he returned to explain his code. (It was very clever.) He then accused her of writing "baby code". She told me that her code was just how I had taught her.

I have several comments on "baby code" and "pretty code".

- A majority of the cost of a piece of software is in maintenance and modification. If "baby code" makes it easier to read and modify, it cost less.
- "Pretty code" is easier to read; Thus easier to understand; Thus easier to maintain.

  If spaces in code, aligning vertically code snippets, blank lines, indentation, etc means the code runs a few milliseconds slower than desired, you are using the **wrong** language.
- And yes, there are times when milliseconds count, but that is not the case for most code.
- It just looks better.
- Too many programmers (and managers) are hung up of the latest language, methodology, framework, ... Sometimes the "old fashion" way is better in the long run.

Simple Design Steps (Think, Design, Code)

1. Brainstorm about what the application/system will do
2. Write down some goals
3. Develop some requirements

   Requirements are very hard to write. Requirements are "what to do" rather than "how to do" something.
4. Develop a preliminary design

   Do not code yet. Design on paper. For example:
   - what data is available; what is its format
   - what data is output; what is its format
   - what data structures to use
   - what web pages might look like
   - ...

5. Prototype to make sure you understand how to do things

   This allows you to test ideas about how to do things, especially things that you have never done before. Prototyping may show that you need to change the design or even the requirements.
6. Code the design

   Do not change the design while coding unless absolutely forced to (You can change the design later). It is better to have something that works rather than something that is constantly changing that never works.

   Do not code the whole thing at once. Break the project down into small tasks that can be coded and tested. This will test your code, and validate your design. This may also cause you to change the design if something is very wrong.

Create a Collection of Code Snippets

Code snippets are code that:

- Help you solve problems on the next project.
- Help you answer the question, "How did I do that last time?".
- Helps you "not reinvent the wheel".
- Reuse code.

As your collection of code snippets grows, it is important to document what it is and how it works for next time.

Small (and sometimes large) programs can be put together like LEGOs with code snippets. Sometimes little or no change is required. Cutting and pasting (good code snippets) can speed up development, especially for common capabilities.

Enigma Machine Simulation

I put this material together for the fun of it. Anyone is free to use it in any manner they wish.
No guarantees are made.

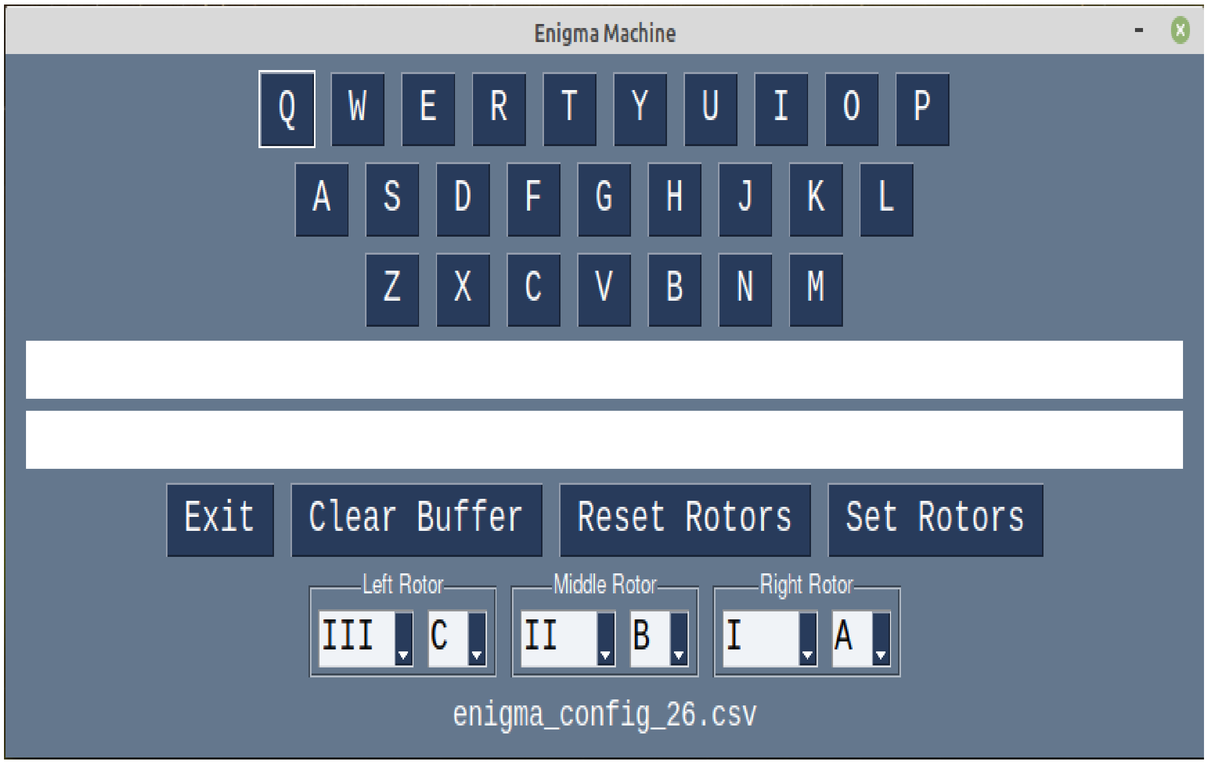
Expertise comes with study and practice, practice, practice, ...

Enigma machine (Wikipedia)
The Enigma machine is a cipher device developed and used in the early- to mid-20th century to protect commercial, diplomatic, and military communication. It was employed extensively by Nazi Germany during World War II, in all branches of the German military.

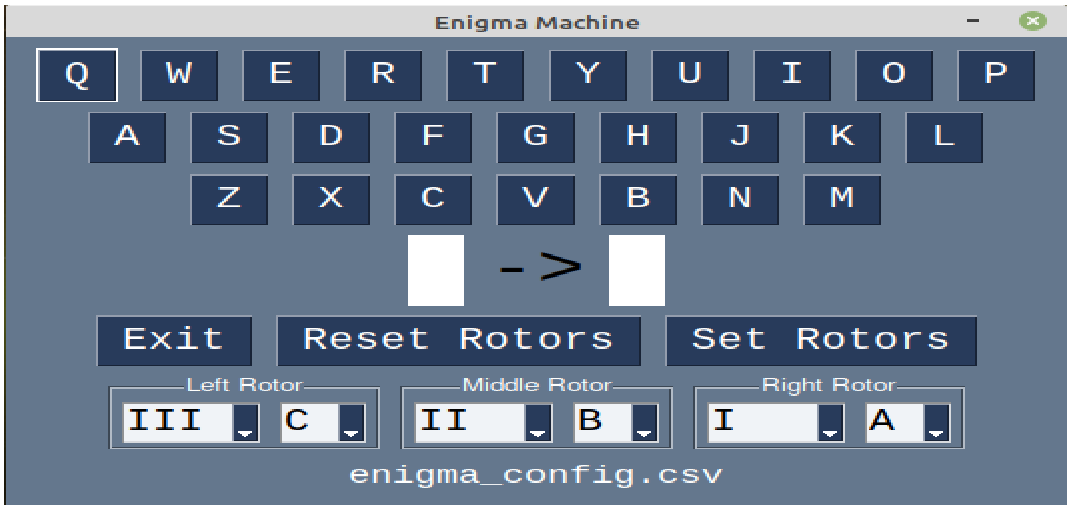
The real Thing

commons.wikimedia.org/wiki/File:Enigma.jpg

Possible Simulation GUI Interface



Another Possible Simulation GUI Interface



Enigma Machine Simulation Project Design

Introduction

This project is to create an Enigma Machine simulation.

The document is my first pass through a design. The as-build (code) is somewhat different. (See the code.)

Program #1

Create rotor configurations and write them to a file. In order to share them, the configuration file format is standardized and will

a. contain eight rotors (I, II, III, IV, V, VI, VII, VIII)
b. each rotor with have 26 letters
c. the internally wiring will be randomly generated
d. The configuration file will contain csv data
e. each line will have the following format: *Rotor,input letter,output letter*
   for example

```
I,A,X
I,B,D
...
...
II,A,Q
II,B,U
...
...
```

Note:

1. *If this is a class project common reflector and rotor configurations should be used. This will allow students to send and receive encrypted messages from other students.*
2. *You can use this program to create a reflectors configuration. (format: R,input letter,output letter)*
3. *The rotor and reflector configurations can be in a single file*
4. *Hint: look up random.shuffle()*

Program #2

Simulate an enigma machine, and only use rotors.

a. read a rotor configuration file
b. allow the user to select which rotors to use
c. allow the user to select which position to place each rotor (left, middle, right)
d. allow the user to select the starting position for each rotor
e. allow the user to entering text to be encrypted
f. allow the user to enter text to be decrypted

g. limit the amount of text to be entered at one time to (60 characters?)
h. on request, display the internal state of the simulation

Program #3

This will be the same as program #1, with the plug board added.

Allow the user to configure the plug board. (interactive?) (add to rotor configuration file?) (separate configuration file?)

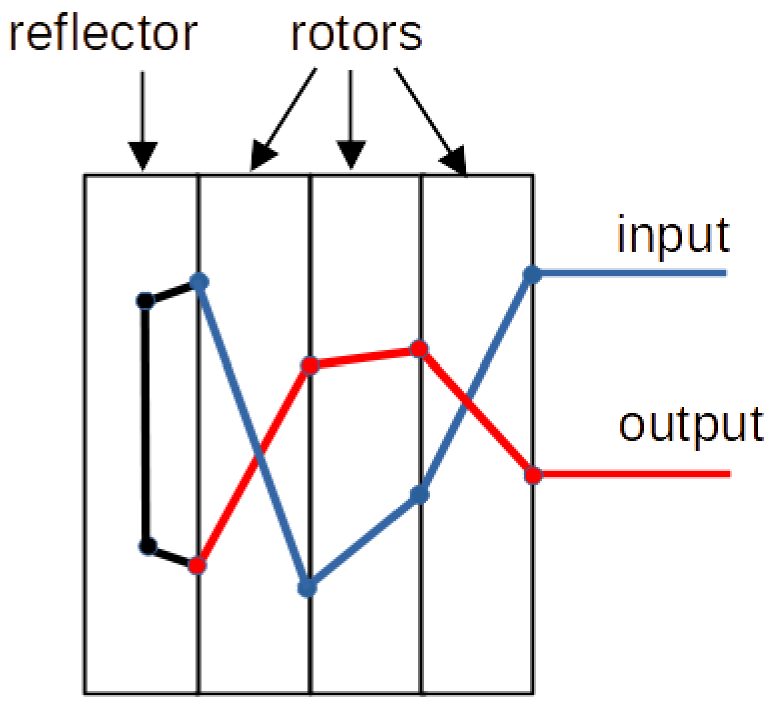
Program #4

Create a GUI for the simulation.

Enigma Machine Simulation Design

Rotor

1. there will be 8 rotors available
2. each rotor has a different wiring
3. each rotor will have 26 letters (all caps)
4. the rotors will be numbered (named) I, II, III, IV, V, VI, VII, VIII
5. the rotor wiring will be in a data file and read by the simulation
6. Each rotor supports starting on any letter
7. The simulation will use 3 rotors
8. every time there is a keypress, the right rotor is advanced one position. After a complete rotation, the middle rotor will be advanced one position. it is the same for the left rotor. when the middle rotor makes a complete rotation, the left rotor is advanced one position.
9. the data input flow is to the
    a. right rotor, then to the
    b. middle rotor, then to the
    c. left rotor, then to the
    d. reflector, and back to the
    e. left rotor, then to the
    f. middle rotor, then to the
    g. right rotor, then to the
    h. data display



Plug Board

The plug board will allow the user to select any two letters to be swapped. This means up to 13 swaps are possible.

Enigma Machine - Design Notes

Introduction

This is an extraction of pertinent information from source material. I have distilled (simplified?) information down to what is needed to create a simulation of an Enigma Machine. There were many version of the Enigma machine. **I used this material for my simulation.**

The Enigma machine only encoded capitol letters (A - Z)
- lowercase letters were not allowed
- numbers were spelled out (one, two, ...)
- unprintable characters were never transmitted
- punctuation, spaces, etc. where replaces by an 'X'

Substitution Cipher

A substitution cipher substitutes one letter for another. Plaintext is thus converted to ciphertext. Knowing the substitution algorithm (process steps), ciphertext can be converted back to plaintext.

The Enigma Machine used a complicated set of mechanical substitution ciphers.

Rotor

Each rotors is wired differently and contains a different fixed substitution cipher. In some Enigma machines three rotors were used, but some used as many as eight. The simulation will use three rotors.

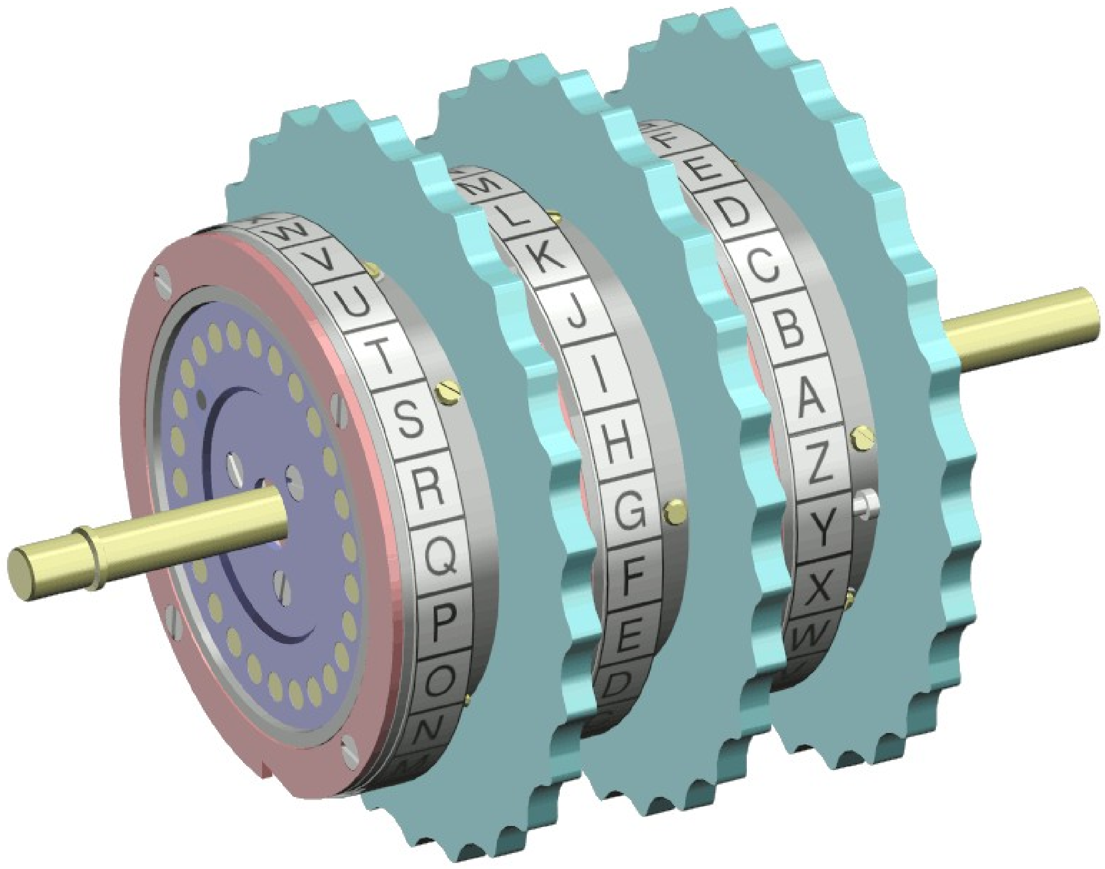
Each rotor was given a name using Roman numerals: I, II, III, IV, V, VI, VII and VIII. To further complicate things the rotors were allowed to be moved around before use. Rotor II might be the left one, with rotor III in the middle and rotor I on the right.

Each rotor also has an attached an alphabet ring that turns with the rotor and was used to set the initial (starting) position of the rotors. The positions were numbered A-Z.

The person doing the encrypting would choose random rotor starting positions and include them at the start of the messages. They sent the rotor positions in the clear followed by the ciphertext.

Which rotors used by the sender and receiver was published in a document. Sometimes it was changed daily. For the simulation the sender and receiver will agree on the order. (Although, if they want to publish a document ...)

Remember, each rotor is named using a Roman numeral.



commons.wikimedia.org/wiki/File:Enigma_rotor_set.png

Reflector

The reflector appears to be nothing but a fixed rotor reflecting each character back thru a different position. A selected character is passed thru each rotor twice. Once in each direction before the results is lit up on the lampboard. Thus passing the character thru the equivalent of six rotors.

Rotation

A keypress caused one or more of the rotors to turn. The first rotor would turn one letter position for each keypress. The second rotor turns one letter after the first rotor made a complete rotation. The third rotor then turned one letter for every complete rotation of the second rotor.

Plug Board

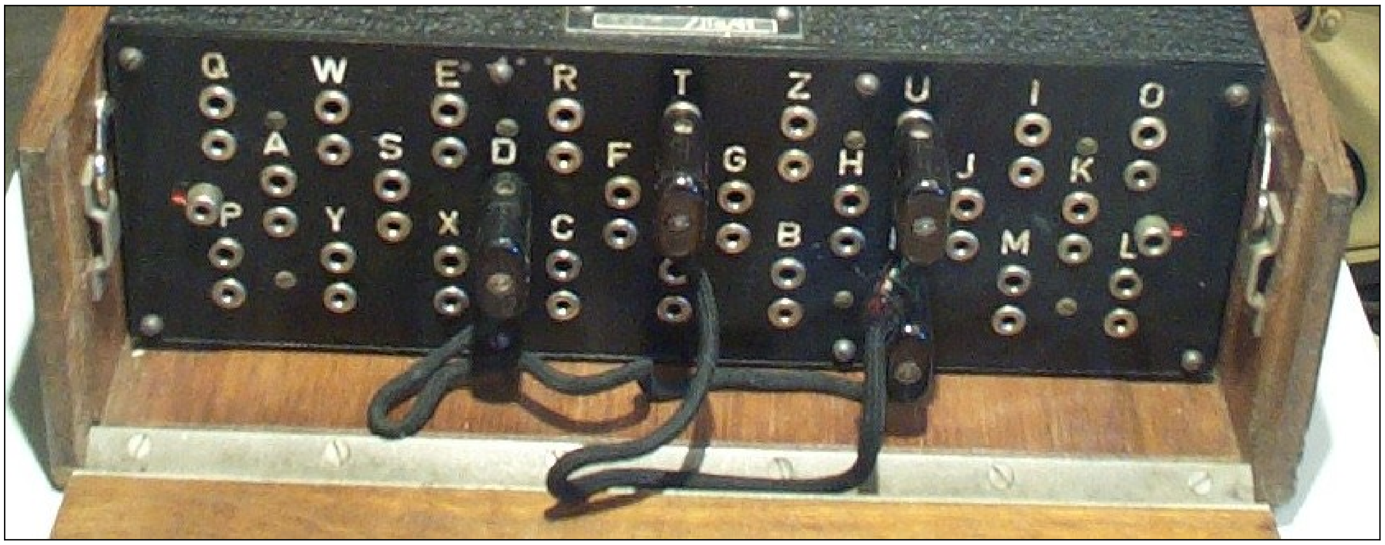
In 1930 the German army versions added a plugboard. This allows letters to be swapped before going to the rotors.

Since there are 26 letters, up to 13 swaps could be done but typically only up to 10 were used.

For example if the plugboard was wired to swapped two letters (A and D)

- If the A key was pressed, D was sent to the rotors.
- If the D key was pressed, A was send to the rotors.

This added another layer of complexity to the encryption.



commons.wikimedia.org/wiki/File:Enigma_Verkehrshaus_Luzern.jpg

Enigma Machine - Interesting Facts

From: A rare 1944 edition of the Kurzsignalheft, or "Enigma Code Book" German, circa 1944

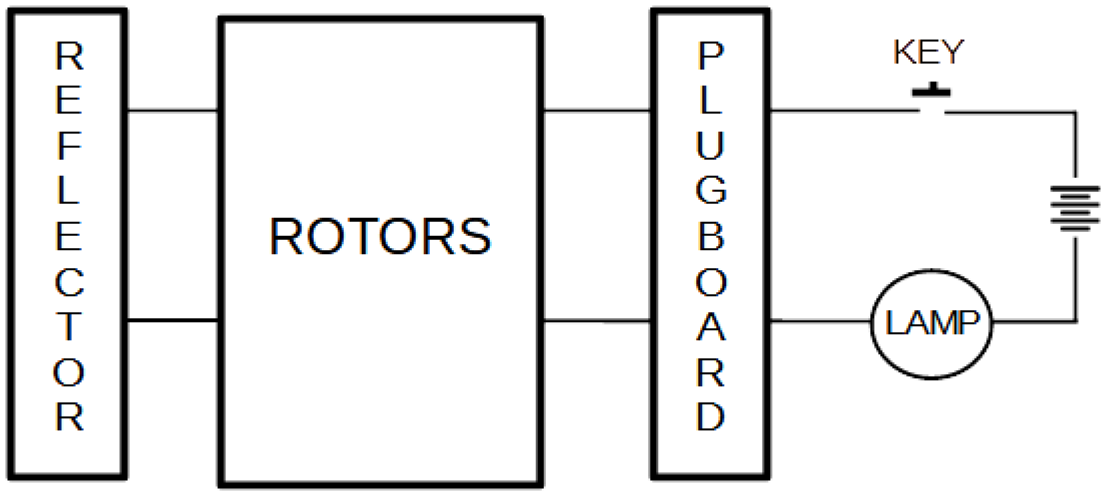
During the Second World War the German U-boats used Kurzsignale or Short Signals to send their messages. The Kurzsignale were an important part of the complex Kriegsmarine communications system. In general, the Kurzsignale were four letter groups representing all kinds of sentences regarding tactical information such as course, enemy reports, position grids or weather reports.

An important reason for the Kriegsmarine to apply these Kurzsignale was the Allied use of High Frequency Direction Finding, also called HFDF or Huff Duff. This system enabled Allied Forces to accurately determine the position of German transmissions. This was an important tactical advantage in the Atlantic, revealing the positions of German ships and U-boats. The use of Kurzsignale decreased the length of the Morse messages enormously, often reducing broadcasting time to less then one minute. This way, the German Navy made it harder to fix positions with Huff Duff.

Enigma Simulation - End to End

Introduction

This diagram shows of an enigma machine electrical circuit.



It shows several things the simulation needs to be careful with

- each character to be encrypted/decrypted has a unique, fixed key. key A can not be key B the next time.
- the character input and output to the rotors, and reflector can not be the same. A (character) can never encrypt/decrypt to itself.
- each key (character) must have a unique and fixed connection to the plugboard.
- any key can light any lamp depending on the encryption/decryption path.
- it may not be obvious, but by reversing the input and output an encrypted character can be decrypted.

Plugboard

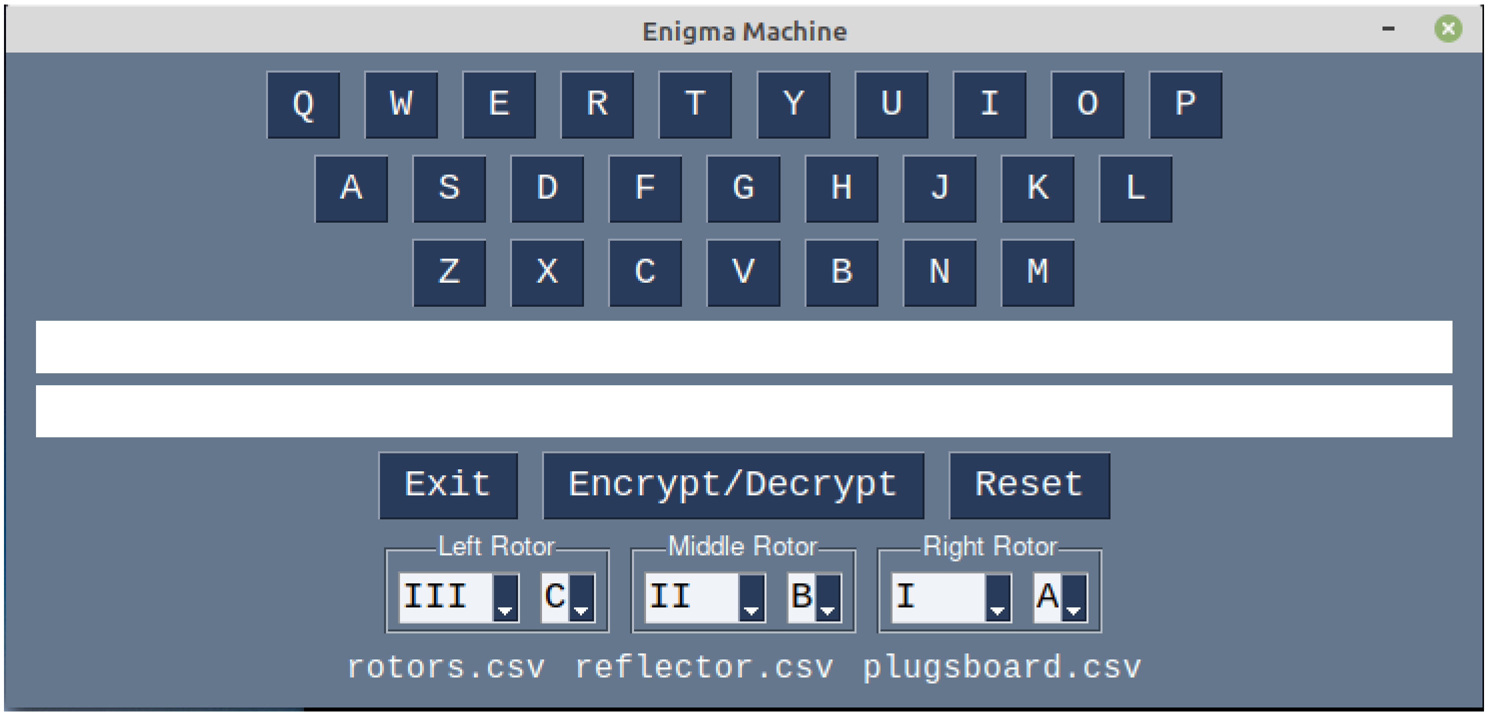
The question is "where in the circuit to place the plugboard"?

I found the explanation in Wikipedia

> *A cable placed onto the plugboard connected letters in pairs; for example, E and Q might be a steckered (plug) pair. The effect was to swap those letters before and after the main rotor scrambling unit. For example, when an operator pressed E, the signal was diverted to Q before entering the rotors.*

It seems to indicate that the plugboard was used before and after the rotor scrambling. That is, twice in a circuit. After the pressed key and after the scrambling. This is how I implemented it in my code. It seems to work.

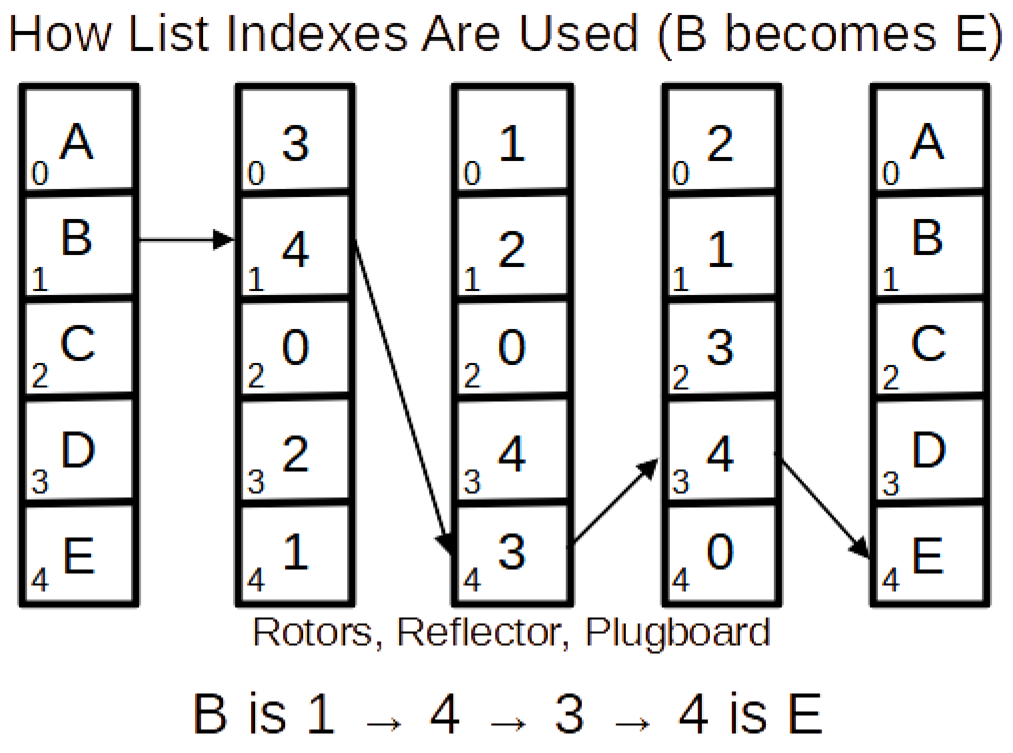
One Possible Enigma GUI



Internally

list indexes are used by the simulation. Alphabetic characters are only used when getting data from or displaying information to the user. For example:

1. When the user presses / clicks on / selects an alphabetic character, the character is used to lookup the list-index of that character.
2. The character's list-index is used internally by the enigma machine simulation. A list-index points to another list-index, that points to another list-index, ...
3. At the end, the final list-index is converted back to an alphabetic character and displayed to the user.



Note: lists and list-indexes are faster and simpler to use in the simulation.

Each rotor, reflector, plugboard can be modeled with a Python list. Each rotor, reflector, plugboard is part of an electric circuit. Each has a "in" pins and an "out" pins which, when connected together with their pins touching form an electric circuit. The list's indexes (0-N) are the "in" pins and the list values (0-N) are the "out" pin. The reflector and plugboard are fixed, but the rotors rotate changing the circuit's path.

Each rotor, reflector, plugboard can also be considered a substitution cipher. The list indexes are the "input" characters and the list's values are the "output" characters.

Python Coding

Python lists and dictionaries can be used to simulate the plugboard, rotors, reflector, and keys.

The GUI was created using the Python PySimpleGUI module.

Enigma Simulation - Randomize Rotor

Introduction

To simulate a "hard wired" rotor substitution cipher, a Python list is used. A list-index is the input (character) and the list-value is the output substituted list-index (character).

For a rotor to be a substitution cipher we create a randomized list from an ordered list. (Note: the lists contain list-indexes and no actual character.)

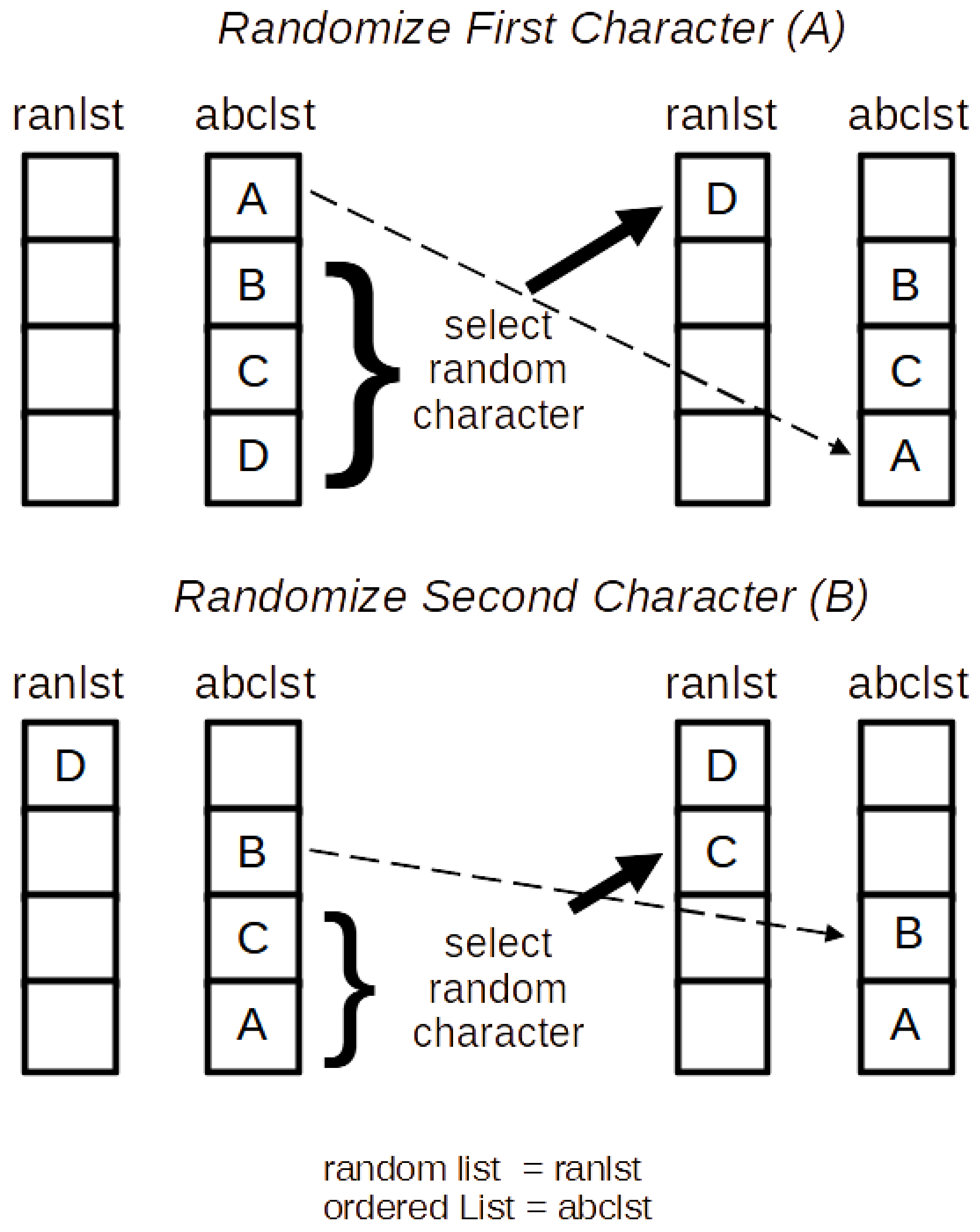
In order to simulate an enigma machine correctly, a character can not be encoded to itself. In other words 'A' can not be substituted for 'A'. It must be a different character.

I created this code to make sure 'A' is not be substituted for 'A'. I searched the web for an hour looking for a existing solution but gave up. So here is my code (such as it is).

I left my print statement in the code. I used them to see what was going on in the code. Comment them out or delete them or use them to see what is going on.

This is not the best/cleanest code. someday I will fix it. For now it is good enough to continue with the enigma machine project.

Randomization Process



The Algorithm

Each character at the top of the ordered list (abclst) is replaced in the randomized list (ranlst) from the remaining characters (in the ordered list abclst).

After being added to the randomized list, the top character replaces the randomly selected character in the ordered list. It can now be selected for inclusion in the random list.

The process continues with the next top character in the ordered list. When there are only two characters left in the ordered list, they are handled differently. See the code.

Enigma Simulation - Internals

Introduction

This documents some of the simulation internals. It is what and sometimes why I did what I did.

Lists

Lists do not contain characters. They contain list indexes which represent characters. When needed they can be converted to a character to display to the user or convert a user's input character to a list index.

Lists are used for character substitutions ciphers. A list index represents an "in" character and the list value at that index represents the "out" character.

A rotor is advanced after a key is pressed. With dictionaries is got very complicated to implement. Lists were simpler.

I think using lists is also more efficient. Perhaps not. I did not perform any tests.

Why lists for rotors, reflector, and plugboard

The encode/decode "circuit" passes through each rotor twice. This means each rotor must encode/decode a character twice. If only one list is used, one direction (right-to-left, left-to-right) must search a list rather than directly convert it.

I played around with using dictionaries and decided it got too complicated. Lists seem more direct. (I could be wrong.)

Why right-to-left rather than left-to-right

In some document I read, when a key is pressed it hits the right rotor first. I kept this convention and reflect it in the data structures. When there is directionality in a list, element 0 is the left rotor, element 1 is the middle rotor, and element 2 is the right rotor. ([left,middle,right])

Why rotors internally have right-to-left and left-to-right lists

Characters (list indexes) are passed thru rotors twice. I use two list for this double pass. I am not sure this is the most efficient way, but this is what I did.

The code could be simplified

I made the code somewhat simple and verbose. I hope it will help the reader better understand what is going on. (Perhaps it just confuses people?)

Enigma Simulation - Rotor Initial Position

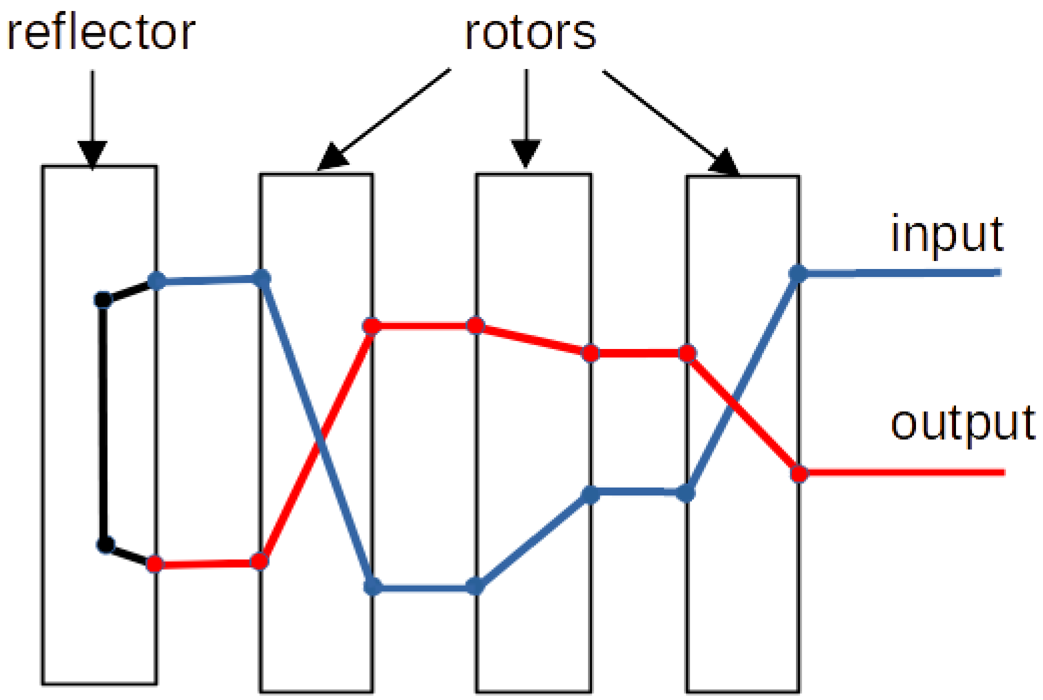
Introduction

Before encrypting/decryption, an enigma machine operator must set the rotors to an initial starting position. (See the operation notes.)

The simulation has an interesting problem. The code must simulate how mechanical rotors and electric circuit works.

Traverse The Rotors and Reflector

Note that the IN pins are not the OUT pins for rotors and reflector. Also note if you switch input and output, an encrypt character can be decrypted.



Keys, Lamps, Pins, Characters, Python Lists

"In the code" keys/lamps/characters are hard coded ...

- A is pin 0 (list-index 0)
- B is pin 1 (list-index 1)
- C is pin 2 (list-index 2)
- ...
- Y is pin 24 (list-index 24)

- Z is pin 25 (list-index 25)

Pins connect characters to rotor pins. And rotor pins eventually connect to lamps. Between the keys and lamps, pins connect to other pins, the other pins connect to yet other pins, ...

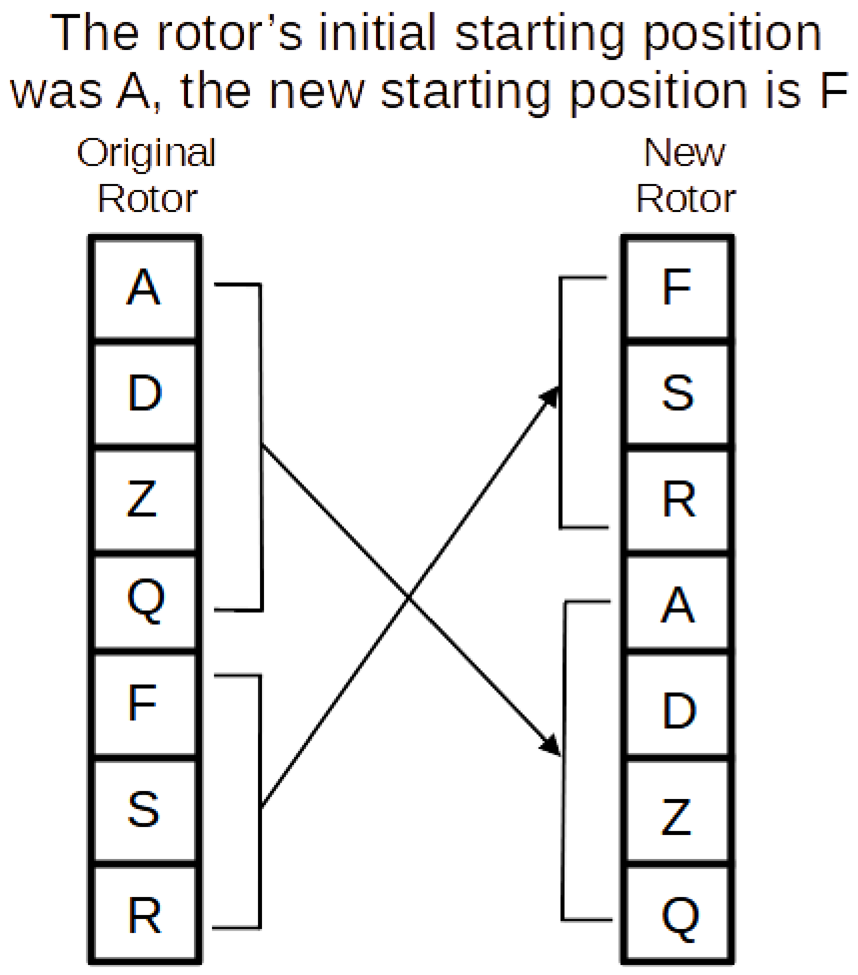
A python list is used to represent a round rotor. When processing reaches the end of the list it start back at the beginning. Thus the list effectively becomes round.

Python lists have an integer index to access elements (values) in the list. The first element's index is zero (0).

Set a Rotor's Starting (Initial) Position

Each rotor's starting position must be set before encryption/decryption. (The default "in the code" is A.)

The diagram shows how the code simulates the rotor being rotated so F is in the starting position. The list order is maintained.

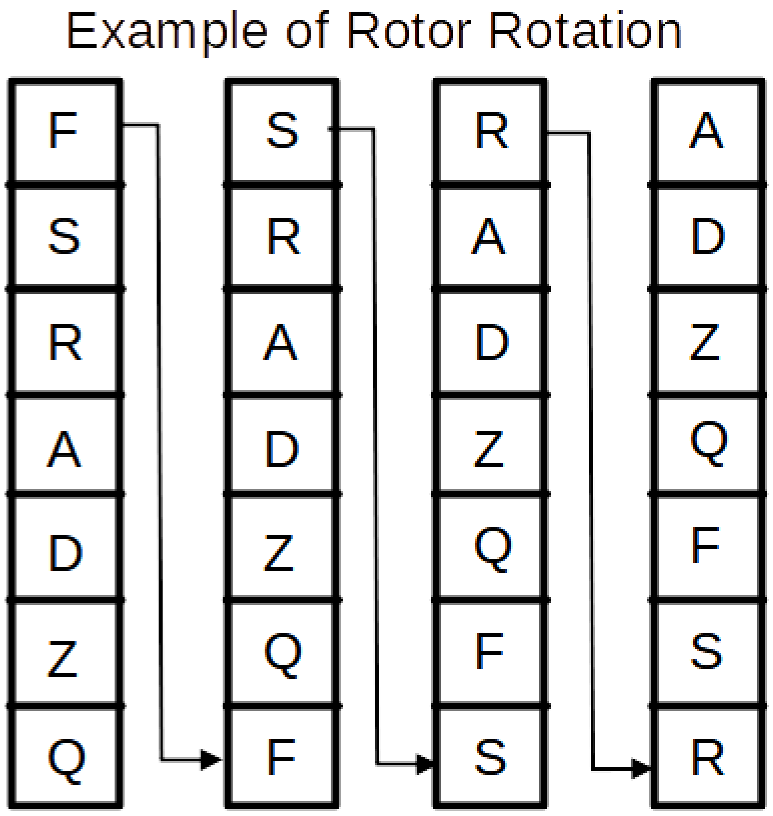

The rotor's initial starting position was A, the new starting position is F
Original Rotor
New Rotor
A
D
Z
Q
F
S
R
F
S
R
A
D
Z
Q

The diagram show characters in a rotor list. In fact, "in the code" it contains list-indexes. The characters are in the diagram to help visualize what is happening. The list is a randomized set of list-indexes (characters).

For example if when a key is pressed, the Key's (character's) list-index (0-25) is used to access the first rotor's list-index (0-25). The list's value at that index is used to access the next rotor, and so on until, at the end, a lamp is lit indicating the encrypted/decrypted character. At each step a new character (list-index) is substituted for the current character (list-index).

Rotate a Rotor

This diagram shows how the code simulates a rotor rotating to the next position to wait for the next key to be pressed. (After each move to the next position the character pins are connected to different set of rotor pins.)



Enigma machine rotors are hard-wired substitution ciphers. Each rotor's IN pin is connected to a different OUT pin. There is a unique IN/OUT pair of pins for each letter of the alphabet. Each letter of the alphabet is associated with a list-index and pin number. For example, the letter A may be pin 0 and list index 0. (See the abc dictionary in the code.)

With a rotor/list, you can think of the list-index as an IN pin/list-index/character. The list's value at the index is an OUT pin/list-index/character. OUT is also the IN pin/list-index/character of the next rotor.

At the start of encryption/decryption, a character is converted to a pin/index. list-indexes are used to traverse the rotors and the reflector and back. At the end, the resulting list-index is converted back to a character and a lamp is lit.

Rotor Loop Code Example

```
#!/usr/bin/python3

lst    = [ 'a','b','c','d' ]   # list
lstlen = len(lst)              # list length
max    = 2                     # max number of time thru the loop
i      = 0                     # list (rotor) element index

while True:

    # ---- display loop information

    print(f'({max}) [{i}] {lst[i]}')

    # --- loop around the list (rotor)

    i += 1
    if not i < lstlen:
        i = 0
        max -= 1

    # ---- make sure the test/demo loop ends

    if max < 0:
        break
```

Enigma Simulation - Rotor List Index Advance Algorithm

Introduction

Hardware rotors are represented in the simulation using python lists. When the physical rotor is advanced one position the python list must be modified (advanced one position). It requires two steps.

1. move the list entries up one

   This rotates the rotor one position. This changes the "IN" points in the list.

   For example:

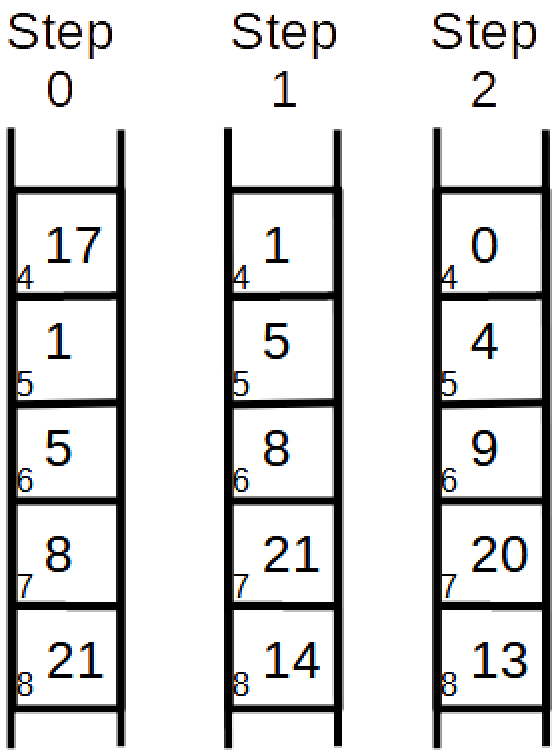
   ```
   (list indexes) 0 -> 26, 1 -> 0, 2 -> 1, 3 -> 2, ...

   (characters)   A -> Z,  B -> A, C -> B, D -> C, ...
   ```

2. reduce the list values by one

   Because the rotor is made of physical wires they don't change lengths. The rotor list "IN and "OUT" points must remain a fixed distance apart. If you move the "IN" point, you must move the "OUT" point.

   This changes the "OUT" points in the list.

Advance Rotor



Enigma Simulation - debug

For debugging or just to see what is going on internally, enter this on the command line

```
./test_EnigmaMachine.py  enigma_config_6.csv | tee log.txt
```

The user sees the normal standard out but it is also captured to a log file.

The Linux script command allows the user to capture terminal sessions.

```
script log.txt
....
....
....
^d
```

ctl-d will exit the script command. It closes the terminal window or ends terminal line input.

Enigma Simulation - Operation Notes

Introduction

This document is sort-of an operation manual for the Enigma simulation. It is not much, but here it is.

To use the Enigma machine simulation to communicate requires a shared rotor, reflector, plugboard configuration.

- The reflector is fixed and never changes.
- The plugboard is changed periodically on an agreed upon schedule. weekly? daily? hourly?
- Which rotors ("I",...,"VII") to use is also part of the published schedule.
- The rotors have initial settings and rotate with each character of a message. (They rotate in a fixed pattern so the sender and receiver can repeat it.)
- The rotor's initial position are part of each message.

It is always a good idea to read the code files for more information. The structure of the code and the comments in the code are another form of documentation.

Enigma Machine Simulation Configuration

The Enigma machine simulation configuration is in a file. It contains rotor, reflector, and plugboard "wiring" and is created by a user. To communicate, the configuration (file) must be shared and used by others.

Which rotors to use and their order (left,middle,right), and the rotor's initial (starting) position must be communicates another way. It could be communicated verbally or written down.

Creating Configuration Files

The program `create_emigma_config_file.py` is available and is used to create configuration files. The file is a simple comma separated value (csv) file. The file format can be understood by reading the file.

The program allows the user to select one of several alphabets and also set plugboard pairs as part of the enigma simulation configuration. It creates the file `enigma_config.csv`.

Multiple Configuration Files

When communicating with a large number of people you may need separate configuration files for different groups. The enigma machine simulation programs have a build in default configuration file (`enigma_config.csv`). If you do not want to use the default, you may include a different configuration file as the first parameter on the command line.

To create separate configuration files:

1. Create a config file using `create_enigma_config_file.py`. This will create a config file having new "wiring" for rotors, and reflector. (Perhaps a new plugboard setup if you enter different character

pairs.)

2. Rename the new config file (`enigma_config.py`).
3. Include the renamed config file on the command line.

Send and Receiving Messages

These steps are from: The Enigma Enigma: How The Enigma Machine Worked

The sender and receiver must use the same configuration of the enigma machine in order to communicate. They must know

- the rotors to be used (I, ... ,VIII)
- the order of the rotors (left,middle,right)
- the plugboard configuration

These were published and changed regularly (hourly? daily?). The sender and receiver looked up this information and configure their enigma machines accordingly.

To set the rotor starting positions, the person doing the encrypting selects random rotor positions and includes them at the start of a message.

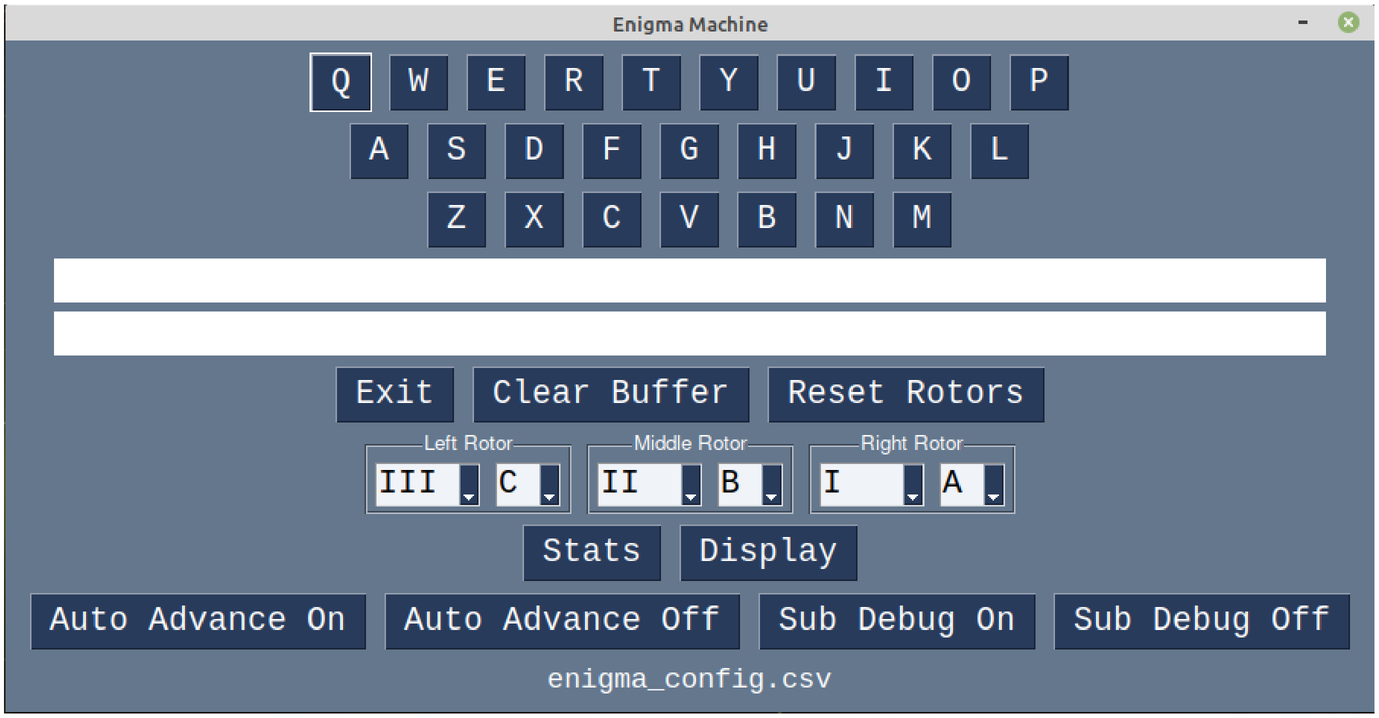
1. Make up an initial start position. For example ABC, and a message key, DEF.
2. Set the rotors to ABC and type in DEF. Let’s say that encrypts to GHI.
3. Next set the rotors to the message key, DEF, and start typing the plaintext message.
4. Record the output, which is the ciphertext.
5. Transmit the initial start position ABC, the encrypted message key GHI, and then the ciphertext.

The person decrypting the message would

1. Set the rotors to the first three letters received, ABC.
2. type in the next three letters, GHI. DEF is output.
3. sets the rotors to DEF.
4. Start typing the ciphertext message. The output is the plaintext message.

Enigma Simulation - GUI Notes

The file `enigma_machine_w_gui.py` combines a GUI with the enigma machine simulation. Below is a screenshot of the GUI.



This is the current version of the enigma machine simulation GUI (as of 11/23/2021). There are "buttons" displayed that are used for debugging. To remove them, edit the file `enigma_gui.py`. Comment out or delete the buttons in the layout. I also suggest leaving the event code for the buttons in the file `enigma_machine_w_gui.py`. This way you can restore them later.

The debug buttons are: Stats, Display, Auto Advance On, Auto Advance Off, Sub Debug On, and Sub Debug Off.

Please note the configuration file used. It is at the bottom of the GUI window.

Enigma Simulation - EnigmaConfig Documentation

EnigmaConfig Class

The class EnigmaConfig implements a name space for enigma machine simulation configuration file information. It also does some verification of the data.

EnigmaConfig Methods

__init__(filename)
initialize and return a enigma simulation configuration object.

filename enigma machine simulation configuration file.

create_plugboard()
create a list that is used as a plugboard from the plugboard information found in the configuration file.

display_config() and **display_stats**
(same function) display configuration information.

Enigma Simulation - EnigmaMachine Documentation

EnigmaMachine Class

The class EnigmaMachine implements an enigma machine. It requires several helper classes (EnigmaConfig and EnigmaRotor). It also uses the function in the file enigma_gui.py.

A configuration file is used to configure the enigma machine when it first starts up. It defines:

- an alphabet
- the number of characters in the alphabet
  (used to verify other entries in the configuration file)
- rotors I through VIII containing randomized indexes (characters)
- reflector randomized list of indexes (characters)
- plugboard character pairs

Configuration files are created using the **create_enigma_config_file.py** program.

EnigmaMachine Methods

__init__(configfilename)
initialize and return an EnigmaMachine object.

display_internal_state()
displays current configuration/state of the rotor's, reflector, plugboard.

display_rotor_status()
display the current state of the rotor's. +Rotor name, start, and count.

char_to_idx(char)
returns a character's list index. If it does not exists in the alphabet, -1 is returned.

idx_to_char(idx)
return the alphabet character associated with the index.Limited to the enigma machine's alphabet. (Hardwired into the enigma machines configuration file.)

char_in_alphabet(char)
return True if the character is in the alphabet; Return False if the character it not in the alphabet.

advance_rotors()
advances the rotors one position. Every time the right rotor makes a complete revolution, the the middle rotor is advances one position. Every the the middle rotor makes a complete revolution, the left rotor is advanced one position. This cascade of rotor advancement continues until the rotors are reset.

advance rotor(rtr)
advances a specified rotor one position. True is returned of the rotor has made a complete revolution; False if it has not.

reset()
sets the simulation rotors to the builtin initial configuration.

- Right rotor: rotor I, starting position 0
- Middle rotor: rotor II starting position 1
- Left rotor: rotor III starting position 2

set(names,starts)
sets the simulation rotors to a new configuration. New rotors (I, II, III, VI, V, VI, VII, VIII) and new starting positions (indexes). The new configuration information is not saved.

Note: the reset command sets the rotors to a builtin configuration. The set command allows the user to set any (legal) configuration.

names is a list of three legal rotor names. Rotor names are limited to I, II, III, IV, V, VI,VII, VIII. Order is important (left,middle,right rotors).

starts is a list of three legal rotor staring positions. Starting positions are list indexes. Starting positions (indexes) are limited to the size of the alphabet (0 to N-1). Order is important (left,middle,right rotors).

auto_advance_on()
auto_advance_off()
auto_advance_state() returns the advancement (rotor rotation) flag. Used for debugging or experimenting.

sub_debug_on()
sub_debug_off()
sub_debug_state() returns the display-character-substitution-detailed-information flag. (Substitution passes through the rotors, the reflector, and the plugboard.) Used for debugging or experimenting.

substitution(idx) performs character substitutions using the rotors, reflector, and plugboard. (In other words, encrypting/decrypting the character using multiple substitution ciphers.)

Enigma Simulation - EnigmaRotor Documentation

EnigmaRotor Class

The class EnigmaRotor implements an enigma machine rotor. It requires the helper classes EnigmaConfig.

EnigmaRotor Methods

__init__(config,rotor_name,rotor_start)
initialize and return an EnigmaRotor object.

config EnigmaConfig object.

rotor_name name of rotor (list) to use. (From EnigmaConfig object.)

rotor_start rotor starting index.

rtol_substitution(idx)
Used for character substitution when passing through the rotor right-to-left. Input is a list index; Output is a different list index.

ltor_substitution(idx)
Used for character substitution when passing through the rotor left-to-right. Input is a list index; Output is a different list index.

reset(rotor_name,rotor_start)

rotor_name name of rotor (list) to use. (From EnigmaConfig object.)

rotor_start rotor starting index.

display(title,all)
Display rotor's internal data.

title (optional) String or None. Title to display.

all (optional) True or False. Display complete internal data.

Enigma Simulation - EnigmaUtil Documentation

EnigmaUtil Class

The class EnigmaUtil implements a namespace for various support functions.

EnigmaUtil Methods

create_randomized_rotor_list(lst,debug)
return a randomized Python list that can be used as a rotor.

lst name of ordered (alphabet) list.

debug (optional) display debug messages

create_randomized_reflector_list(lst,debug)
return a randomized Python list that can be used as a reflector.

list name of ordered (alphabet) list.

debug (optional) display debug messages

Enigma Simulation - GUI Documentation

Introduction

This document answers the question "What do all of those buttons do"?

Buttons

A thru Z
Display a character and its encrypt/decrypt character. The characters are displayed in the text areas. The text areas have a limited size (64 characters I think.).

Clear Buffer
Clears the text areas so the user can start over. The rotors, etc are not changed. They pick-up where they left off.

Reset Rotors
Resets everything (rotors, buffers, etc) to the built-in default configuration.

Set Rotors
Sets the internal configuration using the left-rotor, middle-rotor and right-rotor values displayed.

Exit
Exit the program.

left-rotor, middle-rotor, right-rotor
Allows the user to define the configuration used by the set command.

Stat, Display, Auto Advance On, Auto Advance Off, Sub Debug On, Sub Debug Off
Used for debugging or just watching what goes on internally.

Enigma Simulation - My Journey

I spend a lot of time trying various code solutions before I got to where I am now. I tested various dictionary and list configurations; I played around with algorithms on paper and in code; Some (not all) of those ended up as "demo" programs.

One of the design problems I have not finalized (in my head) is how much functionality to put in the EnigmaMachine vs the EnigmaRotor objects. I keep vacillating back and forth and have not come to any conclusion. Should EnigmaRotor just be a namespace or should it also contain methods.

Enigmas Simulation - What to do Next

What to do Next

After the code is working, what to do next.

- Remove redundant code. There is a lot of redundant/useless code in the python files (*.py). It is there because I did a lot of tinkering with code while figuring out what I wanted to do. As an example, with a little re-structuring multiple global variables could be eliminated.
- Cleanup/re-structure the code. For example,
    - Eliminate the EnigmaRotor class?
    - Move more functionality into the EnigmaRotor class?
    - Add help information to the GUI?
    - Make a better GUI
    - Create documentation. (Use Pydoc for the Python class interface?)
- Create a program to encrypt/decrypt a file. What do you do with characters not in the alphabet? New-line? Carriage-return?
- Create an expanded alphabet. For example, add the characters ' ', '\n', '\r', and others. (perhaps lowercase?)
  (remember there must be a even number of characters in the alphabet.)
- Create a software enigma machine that allow a character to substitute itself. (Bypass the hardware limitation.)
- Output Morse code? Input Morse code? *(What is a Numbers Station?)*
- Create a GUI that somehow simulates "lighting a lamps". (Actual hardware light bulbs? Seven segment display?)
- Create an Enigma Machine that can encrypt/decrypt binary files/data. (split bytes into nibbles? base64 binary-to-text encoding?)

- Currently rotor lists contain absolute index values. Change them to relative index values. (Relative to the current index +N or – N?) Also use pop/append to rotate a rotor (list)? *For example: next_rotor_idx = (current_rotor_idx + list[current_rotor_idx]) % list_length.*

Python3 Files

The enigma machine simulation works, but I am sure there are still bugs in the code. I apologize in advance.

To get the code, copy-and-paste.

The code contains comments which are part of the documentation.

enigma_gui.py

```
# ====================================================================
# Create an enigma machine simple GUI
# ====================================================================

import PySimpleGUI as sg
import sys

def enigma_gui(config):

    #font2     = ('Aerial',12)
    #font4     = ('Aerial',14)
    font2      = ('Courier New',14)
    font4      = ('Courier New',16)

    rotors     = list(config.rotors.keys())
    abc        = config.abc
    abclen     = config.abclen
    tsize      = 64              # GUI text field size

    # ---- GUI rotors ----------------------------------------------

    names  = config.default_rotor_names
    starts = config.default_rotor_starts

    rotor1 = [ [sg.Combo(rotors,
                         default_value=names[2],
                         font=font4,key='-RROTOR-'),
                sg.Combo(abc,
                         default_value=abc[starts[2]],
                         font=font4,key='-RPOSITION-',size=(2,1))]]
    rotor2 = [ [sg.Combo(rotors,
                         default_value=names[1],
                         font=font4,key='-MROTOR-'),
                sg.Combo(abc,
                         default_value=abc[starts[1]],
                         font=font4,key='-MPOSITION-',size=(2,1))]]
    rotor3 = [ [sg.Combo(rotors,
                         default_value=names[0],
                         font=font4,key='-LROTOR-'),
                sg.Combo(abc,
                         default_value=abc[starts[0]],
                         font=font4,key='-LPOSITION-',size=(2,1))]]

    # ---- big alphabet (many characters) --------------------------

    alphabet0 = [ 'A','B','C','D','E','F','G','H','I','J','K','L',
                  'M','N','O','P','Q','R','S','T','U','V','W','X',
                  'Y','Z','1','2','3','4','5','6','7','8','9','0',
                  '.','=','+','-','/','*','(',')' ]
    layout0   = [ [sg.Button('1',font=font4),
                   sg.Button('2',font=font4),
                   sg.Button('3',font=font4),
                   sg.Button('4',font=font4),
                   sg.Button('5',font=font4),
                   sg.Button('6',font=font4),
                   sg.Button('7',font=font4),
                   sg.Button('8',font=font4),
                   sg.Button('9',font=font4),
                   sg.Button('0',font=font4)],

                  [sg.Button('Q',font=font4),
                   sg.Button('W',font=font4),
                   sg.Button('E',font=font4),
                   sg.Button('R',font=font4),
                   sg.Button('T',font=font4),
                   sg.Button('Y',font=font4),
                   sg.Button('U',font=font4),
                   sg.Button('I',font=font4),
                   sg.Button('O',font=font4),
                   sg.Button('P',font=font4)],

                  [sg.Button('A',font=font4),
                   sg.Button('S',font=font4),
                   sg.Button('D',font=font4),
                   sg.Button('F',font=font4),
                   sg.Button('G',font=font4),
                   sg.Button('H',font=font4),
                   sg.Button('J',font=font4),
                   sg.Button('K',font=font4),
                   sg.Button('L',font=font4)],

                  [sg.Button('Z',font=font4),
                   sg.Button('X',font=font4),
                   sg.Button('C',font=font4),
                   sg.Button('V',font=font4),
                   sg.Button('B',font=font4),
                   sg.Button('N',font=font4),
                   sg.Button('M',font=font4),
                   sg.Button('.',font=font4),
                   sg.Button('M',font=font4)],

                  [sg.Button('=',font=font4),
                   sg.Button('+',font=font4),
                   sg.Button('-',font=font4),
                   sg.Button('*',font=font4),
                   sg.Button('(',font=font4),
                   sg.Button(')',font=font4)] ]

    # ---- enigma alphabet (26 characters) -------------------------
    # ---- special key layout for enigma alphabet

    alphabet1 = [ 'A','B','C','D','E','F','G','H','I','J','K','L','M',
                  'N','O','P','Q','R','S','T','U','V','W','X','Y','Z' ]
    layout1   = [ [sg.Button('Q',font=font4),
                   sg.Button('W',font=font4),
                   sg.Button('E',font=font4),
                   sg.Button('R',font=font4),
                   sg.Button('T',font=font4),
                   sg.Button('Y',font=font4),
                   sg.Button('U',font=font4),
                   sg.Button('I',font=font4),
                   sg.Button('O',font=font4),
                   sg.Button('P',font=font4)],

                  [sg.Button('A',font=font4),
                   sg.Button('S',font=font4),
                   sg.Button('D',font=font4),
                   sg.Button('F',font=font4),
                   sg.Button('G',font=font4),
                   sg.Button('H',font=font4),
                   sg.Button('J',font=font4),
                   sg.Button('K',font=font4),
                   sg.Button('L',font=font4)],

                  [sg.Button('Z',font=font4),
                   sg.Button('X',font=font4),
                   sg.Button('C',font=font4),
                   sg.Button('V',font=font4),
                   sg.Button('B',font=font4),
                   sg.Button('N',font=font4),
                   sg.Button('M',font=font4)] ]

    # ---- test alphabet ------------------------------------------

    alphabet2 = [ 'A','B','C','D','E','F' ]
    layout2   = [ [sg.Button('A',font=font4),
                   sg.Button('B',font=font4),
                   sg.Button('C',font=font4),
                   sg.Button('D',font=font4),
                   sg.Button('E',font=font4),
                   sg.Button('F',font=font4)] ]

    # ---- layout without alphabet ---------------------------------

    layoutx = [ [sg.Text(font=font4,text_color='black',key='-IN-',
                         background_color='white',size=(tsize,1))],
                [sg.Text(font=font4,text_color='black',key='-OUT-',
                         background_color='white',size=(tsize,1))],

                [sg.Button('Exit',font=font4),
                 sg.Button('Clear Buffer',font=font4),
                 sg.Button('Reset Rotors',font=font4),
                 sg.Button('Set Rotors',font=font4)],

                [sg.Frame('Left Rotor',rotor3,title_location='n',
                       border_width=2,key='-ROTOR3-'),
                 sg.Frame('Middle Rotor',rotor2,title_location='n',
                          border_width=2,key='-ROTOR2-'),
                  sg.Frame('Right Rotor',rotor1,title_location='n',
                           border_width=2,key='-ROTOR1-')],

                [sg.Button('Stats',font=font4),
                 sg.Button('Display',font=font4)],

                [sg.Button('Auto Advance On',font=font4),
                 sg.Button('Auto Advance Off',font=font4),
                 sg.Button('Sub Debug On',font=font4),
                 sg.Button('Sub Debug Off',font=font4)],

                [sg.Text(config.configfile,font=font2)] ]
```

```
# ---- return a GUI window --------------------------------------

if abclen == 44:                   # big alphabet
    layout = layout0 + layoutx
elif abclen == 26:                 # enigma alphabet
    layout = layout1 + layoutx
elif abclen == 6:                  # test apphabet
    layout = layout2 + layoutx
else:
    print()
    print('internal error - ' +
          'alaphabet size and GUI layout mismatch - ' +
          'end program')
    print()
    sys.exit()

win = sg.Window('Enigma Machine',layout,
                element_justification='c')

return (win,tsize)
```

EnigmaConfig.py

```
# ===================================================================
# Enigma Machine Simulation - read/process configuration file
# ===================================================================

from datetime import date
import user_interface as ui
import copy
import re
import sys

# -------------------------------------------------------------------
# ---- namespace for enigma simulation runtime configuration
# -------------------------------------------------------------------

class EnigmaConfig():

    def __init__(self,filename):

        # ---- rotor dictionary and keys
        self.rotors     = { 'I':[], 'II':[], 'III':[],   'IV':[],
                            'V':[], 'VI':[], 'VII':[], 'VIII':[] }

        self.rotorkeys = list(self.rotors.keys())

        self.abc        = []                        # alphabet list
        self.abclen     = 0                         # abc list length
        self.abcidx     = []                        # list of alphabet
                                                    #   indexes
        self.configfile = filename                  # configuration
                                                    #   file
        self.line_count = 0                         # line count read
                                                    #   from config file
        self.plugs      = []                        # plugboard pairs
        self.pglst      = []                        # plugboard list
        self.reflst     = []                        # reflector list
        self.date       = date.today()              # config file date

        # ---- default enigma machine rotor configuration
        # ---- [left,middle,right]

        self.default_rotor_names = ['III','II','I']
        self.default_rotor_starts= [2,1,0]

        # ---- create an empty configuration

        if not filename:
            return

        # ---- read/process configuration file

        fh = open(filename,'r')

        for line in fh:
```

```
self.line_count += 1

line = line.strip()

if not line:               # empty string (line)
    continue

if re.match('^#',line):         # comment
    continue

x = line.split(',')

# --- date file created

if x[0] =='date':               # date
    self.date = x[1]
    continue

# --- alphabet length?

if x[0] == 'len':               # alphabet length
    tf,i = ui.is_int(x[1])
    if tf:
        if i >= 0:
            self.abclen = i
            continue

if x[0] == 'unk':               # substitute for unknown
                                #   character
    self.abcunk = x[1]
    continue

# --- alphabet?

if x[0] == 'abc':               # alphabet
    self.abc.append(x[2])
    continue

# --- plug?

if x[0] == 'plug':              # plugboard
    self.plugs.append(x[1])
    continue

# ---- reflector?

if x[0] == 'ref':               # reflector
    self.reflst.append(int(x[2]))
    continue

# ---- rotor?

if x[0] in self.rotors.keys():  # rotor
    tf,i = ui.is_int(x[2])
    if tf:
        if i >= 0:
            self.rotors[x[0]].append(int(x[2]))
            continue
        # ---- what?

        print()
        print('unknown line in config file ' +
             f'line={line_count})\n({line})')
        sys.exit()

    fh.close()

    # ---- verify config file data

    for r in self.rotors:
        if len(self.rotors[r]) != self.abclen:
            print()
            print(f'bad rotor list length (r)')
            sys.exit()
    if len(self.abc) != self.abclen:
        print()
        print('bad abc list length\n(alphabet)')
        sys.exit()
    if len(self.reflst) != self.abclen:
        print()
        print('bad reflector list length\n(reflector)')
        sys.exit()
    for i in range(self.abclen):
        j = self.reflst[i]
        if self.reflst[j] != i:
            print(f'error in reflector pair i={i} j={j}')
            sys.exit()

    # ---- now set abcidx

    self.abcidx = list(range(self.abclen))

    # ---- now create plugboard

    self.create_plugboard()

# -----------------------------------------------------------------
# ---- helper function: convert character to index
# -----------------------------------------------------------------

def _char_to_index(self,c):
    for i in range(len(self.abc)):
        if c == self.abc[i]:
            return i
    print()
    print(f'character c not found in alphabet')
    sys.exit()

# -----------------------------------------------------------------
# ---- create plugboard (list)
# -----------------------------------------------------------------

def create_plugboard(self):

    used_chars = []
```

```
        self.pglst = list(range(self.abclen))

        for cc in self.plugs:

            # --- error check

            if len(cc) != 2 or cc[0] == cc[1] or \
                cc[0] in used_chars or \
                cc[1] in used_chars or \
                cc[0] not in self.abc or \
                cc[1] not in self.abc:
                print()
                print(f' error in plugboard pair ({cc})')
                sys.exit()

            used_chars.append(cc[0])
            used_chars.append(cc[1])

            idx0 = self._char_to_index(cc[0])
            idx1 = self._char_to_index(cc[1])

            self.pglst[idx0] = idx1
            self.pglst[idx1] = idx0

    # -----------------------------------------------------------------
    # ---- display EnigmaConfig internals
    # -----------------------------------------------------------------

    def display_config(self):
        self.display_stats()

    def display_stats(self):
        print()
        print(f'config   file: {self.configfile}')
        print(f'date         : {self.date}')
        print(f'lines read   : {self.line_count}')
        print(f'alphabet  len: {len(self.abc)}')
        print(f'  abclen     : {self.abclen}')
        print(f'  abcunk     : {self.abcunk}')
        print(f'rotor   count: {len(self.rotors)}')
        for r in self.rotors.keys():
            print(f'  {r:5}   len: {len(self.rotors[r])}')
        print(f'reflector len: {len(self.reflst)}')
        print(f'plugs        : {len(self.plugs)}')
        for p in self.plugs:
            print(f'  ({p})')
        ##for idx in range(len(self.pglst)):
        ##    print(f'  {idx:>2} --> {self.pglst[idx]:<2}  i' +
        ##          f'({self.abc[idx]})')
        print()
```

EnigmaMachine.py

```
# ==================================================================
# test the enigma machine simulation with and without rotor advance
# ==================================================================
# the user can advance the rotors with the "advance" command.
# ------------------------------------------------------------------
# Note: because this is software, we do some special "stuff" with
# characters not in the enigma machine's alphabet [A-Z]. see the
# code/comments.
# ==================================================================

from EnigmaConfig    import EnigmaConfig
from EnigmaRotor     import EnigmaRotor
import EnigmaUtils
import user_interface as ui
import copy
import re
import sys


class EnigmaMachine:

    # --------------------------------------------------------------
    # ---- initialize object
    # --------------------------------------------------------------
    def __init__(self,configfile):

        # ---- enigma machine configuration

        self.config = EnigmaConfig(configfile)

        # ---- enigma machine alphabet

        self.abc    = self.config.abc
        self.abcidx = self.config.abcidx
        self.abclen = self.config.abclen
        self.abcunk = self.config.abcunk

        self.rotorkeys = self.config.rotorkeys

        # ---- left rotor, middle rotor, right rotor,
        # ---- reflector, plugboard

        self.lr = EnigmaRotor(self.config,
                self.config.default_rotor_names[0],
                self.config.default_rotor_starts[0])
        self.mr = EnigmaRotor(self.config,
                self.config.default_rotor_names[1],
                self.config.default_rotor_starts[1])
        self.rr = EnigmaRotor(self.config,
                self.config.default_rotor_names[2],
                self.config.default_rotor_starts[2])
        self.rf = copy.deepcopy(self.config.reflst)
        self.pg = copy.deepcopy(self.config.pglst)
        # ---- automatic advance rotors

        self.auto_advance_rotors = True

        # ---- substitution debug messages

        self.substitution_debug = False

    # -----------------------------------------------------------------
    # ---- display internal state of enigma machine
    # -----------------------------------------------------------------
    def display_internal_state(self):
        print()
        print('                    -- Left Rotor --- ' +
              '-- Middle Rotor - -- Right Rotor --')
        print('plugboard reflector  ' +
              'l-to-r   r-to-l   l-to-r   r-to-l   l-to-r   r-to-l')
        print('--------- --------- -------- -------- ' +
              '-------- -------- -------- --------')
        for i in range(self.abclen):
             print(f'{i:>2} -> {self.pg[i]:<2}  ' +
                   f'{i:>2} -> {self.rf[i]:<2}  ' +
                   f'{i:>2} -> {self.lr.ltor[i]:<2} ' +
                   f'{self.lr.rtol[i]:>2} <- {i:<2} ' +
                   f'{i:>2} -> {self.mr.ltor[i]:<2} ' +
                   f'{self.mr.rtol[i]:>2} <- {i:<2} ' +
                   f'{i:>2} -> {self.rr.ltor[i]:<2} ' +
                   f'{self.rr.rtol[i]:>2} <- {i:<2}')
        print(f'---- auto advance rotors ({self.auto_advance_rotors})')
        print(f'---- right  start {self.rr.rotor_start:<2} ' +
              f'rotor {self.rr.rotor_name:<4} ' +
              f'count {self.rr.rotor_count}')
        print(f'---- middle start {self.mr.rotor_start:<2} ' +
              f'rotor {self.mr.rotor_name:<4} ' +
              f'count {self.mr.rotor_count}')
        print(f'---  left   start {self.lr.rotor_start:<2} ' +
              f'rotor {self.lr.rotor_name:<4} ' +
              f'count {self.lr.rotor_count}')
        return

    # -----------------------------------------------------------------
    # ---- display rotor stats of enigma machine
    # -----------------------------------------------------------------
    def display_rotor_stats(self):
        print()
        print(f'auto advance rotors ({self.auto_advance_rotors})')
        print(f'right  start {self.rr.rotor_start:<2} ' +
              f'rotor {self.rr.rotor_name:<4} ' +
              f'count {self.rr.rotor_count}')
        print(f'middle start {self.mr.rotor_start:<2} ' +
              f'rotor {self.mr.rotor_name:<4} ' +
              f'count {self.mr.rotor_count}')
        print(f'left   start {self.lr.rotor_start:<2} ' +
              f'rotor {self.lr.rotor_name:<4} ' +
              f'count {self.lr.rotor_count}')
        return

    # -----------------------------------------------------------------
    # ---- convert character to index
    # ----------------------------------------------------------------
    def char_to_idx(self,c):
        if c in self.abc:
            for i in range(self.abclen):
                if self.abc[i] == c:
                    return i
        return -1

    # ----------------------------------------------------------------
    # ---- convert index to character
    # ----------------------------------------------------------------
    def idx_to_char(self,idx):
         return self.abc[idx]

    # ----------------------------------------------------------------
    # ---- advance the rotors one position
    # ---- advance each rotor until it makes a complete revolution
    # ----   then start over again
    # ---- note: tf is a true/false flag
    # ----       True  have completed a full rotation
    # ----       False have not completed a complete full rotation
    # ----------------------------------------------------------------
    def advance_rotors(self):

        tf = self.advance(self.rr)           # right rotor
        if tf:
            tf = self.advance(self.mr)       # middle rotor
            if tf:
                tf = self.advance(self.lr)   # left rotor
        return

    # ----------------------------------------------------------------
    # ---- advance a rotor
    # ---- return:
    # ----     True  have completed a full rotation
    # ----     False have not completed a complete full rotation
    # ----------------------------------------------------------------
    def advance(self,r):

        # ---- advance rotor's rtol list

        rtollen = len(r.rtol)

        tmp = r.rtol[0]

        for i in range(rtollen-1):
            r.rtol[i] = (r.rtol[i+1] - 1) % rtollen

        r.rtol[-1] = (tmp-1) % rtollen

        # ---- sync rotor's ltor list with its rotl list

        for i in range(rtollen):
            r.ltor[r.rtol[i]] = i

        # ---- check for a complete rotation of the rotor
        # ---- (end of list)
        r.rotor_count += 1
        if not r.rotor_count < rtollen:
            r.rotor_count = 0
            return True
        return False

    # -----------------------------------------------------------------
    # ---- reset the internal state to the initial configuration
    # -----------------------------------------------------------------
    def reset(self):

        self.lr = EnigmaRotor(self.config,
                self.config.default_rotor_names[0],
                self.config.default_rotor_starts[0])
        self.mr = EnigmaRotor(self.config,
                self.config.default_rotor_names[1],
                self.config.default_rotor_starts[1])
        self.rr = EnigmaRotor(self.config,
                self.config.default_rotor_names[2],
                self.config.default_rotor_starts[2])

        self.lr.rotor_count = 0
        self.mr.rotor_count = 0
        self.rr.rotor_count = 0

        return

    # -----------------------------------------------------------------
    # ---- set rotor configuration
    # ---- the initial rotor configuration is still available
    # ---- using reset_initial_state function
    # -----------------------------------------------------------------
    def set(self,names,starts):

        # --- verify input lists

        if len(names)  != 3 or len(starts) != 3:
               print()
               print('internal error - bad set rotor comfiguration ' +
                     'names or starts list size')
               print(f'({names}) ({starts})')
               print()
               sys.exit()

        for i in range(3):
            if names[i] not in self.rotorkeys:
               print()
               print('internal error - bad set rotor comfiguration ' +
                     'rotor name')
               print(f'({names})')
               print()
               sys.exit()

        for s in range(3):
            if s< 0 or s >self.config.abclen-1:
               print()
               print('internal error - bad set rotor comfiguration ' +
                     'start position')
               print(f'({starts})')
               print()
               sys.exit()

        # ---- set rotor configuration

        self.lr = EnigmaRotor(self.config,names[0],starts[0])
        self.mr = EnigmaRotor(self.config,names[1],starts[1])
        self.rr = EnigmaRotor(self.config,names[2],starts[2])

        self.lr.rotor_count = 0
        self.mr.rotor_count = 0
        self.rr.rotor_count = 0

        return


    # ----------------------------------------------------------------
    # ---- change auto advance
    # ----------------------------------------------------------------

    def auto_advance_on(self):
        self.auto_advance_rotors = True
        return True

    def auto_advance_off(self):
        self.auto_advance_rotors = False
        return False

    def auto_advance_state(self):
        return auto_advance_rotors

    # ----------------------------------------------------------------
    # ---- change substitution debug
    # ----------------------------------------------------------------

    def sub_debug_on(self):
        self.substitution_debug = True
        return True

    def sub_debug_off(self):
        self.substitution_debug = False
        return False

    def sub_debug_state(self):
        return self.substitution_debug

    # ----------------------------------------------------------------
    # ---- substitute character
    # ----------------------------------------------------------------
    def substitution(self,idx):

        idx1 = self.pg[idx]                     # plugboard
        idx2 = self.rr.rtol_substitution(idx1)
        idx3 = self.mr.rtol_substitution(idx2)
        idx4 = self.lr.rtol_substitution(idx3)
        idx5 = self.rf[idx4]                    # reflector
```

```
idx6 = self.lr.ltor_substitution(idx5)
idx7 = self.mr.ltor_substitution(idx6)
idx8 = self.rr.ltor_substitution(idx7)
idx9 = self.pg[idx8]                      # plugboard

if self.substitution_debug:
    print(f'{idx} --> ',end='')
    print(f'{idx1} --> ',end='')
    print(f'{idx2} --> ',end='')
    print(f'{idx3} --> ',end='')
    print(f'{idx4} --> ',end='')
    print(f'{idx5} --> ',end='')
    print(f'{idx6} --> ',end='')
    print(f'{idx7} --> ',end='')
    print(f'{idx8} --> ',end='')
    print(f'{idx9}')

    print(f'{self.abc[idx]} --> ',end='')
    print(f'{self.abc[idx1]} --> ',end='')
    print(f'{self.abc[idx2]} --> ',end='')
    print(f'{self.abc[idx3]} --> ',end='')
    print(f'{self.abc[idx4]} --> ',end='')
    print(f'{self.abc[idx5]} --> ',end='')
    print(f'{self.abc[idx6]} --> ',end='')
    print(f'{self.abc[idx7]} --> ',end='')
    print(f'{self.abc[idx8]} --> ',end='')
    print(f'{self.abc[idx9]}')
    print(f'[{idx} ({self.abc[idx]})]  -->  ' +
          f'[{idx8} ({self.abc[idx9]})]')

if self.auto_advance_rotors:
    self.advance_rotors()

return idx9
```

EnigmaRotor.py

```
# ====================================================================
# Enigma Machine Simulation - rotor
# ====================================================================

from EnigmaConfig import EnigmaConfig
import EnigmaUtils as util
import copy

class EnigmaRotor():

    # ----------------------------------------------------------------
    # ---- initialize rotor object
    # ----------------------------------------------------------------
    # ---- abc             alphabet list
    # ---- abclen          alphabet list length
    # ---- abcunk          unknown character substitute
    # ---- rotors          configuration file dictionary of
    # ----                     rotor lists (I,II,...,VIII)
    # ---- rotor_name      rotor name (from rotors)
    # ---- rotor_start     rotor start position (idx=0)
    # ---- rotor_count     rotation/advancement count
    # ---- lst             randomized list of indexes from
    # ----                   configuration file
    # ----                 (associated with the alphabet)
    # ---- lstlen          length, lst
    # ---- ltor            left-to-right substitution list
    # ---- rtol            right-to-left substitution list
    #-----------------------------------------------------------------
    def __init__(self,config,rotor_name,rotor_start):

        self.abc         = config.abc
        self.abcidx      = config.abcidx
        self.abclen      = config.abclen
        self.abcunk      = config.abcunk
        self.rotors      = config.rotors
        self.rotor_name  = rotor_name
        self.rotor_start = rotor_start
        self.rotor_count = 0
        self.lst         = []
        self.lstlen      = []
        self.rtol        = []
        self.ltor        = []

        self.reset(rotor_name,rotor_start)

    # ----------------------------------------------------------------
    # ---- character substitution right-to-left
    # ----------------------------------------------------------------
    def rtol_substitution(self,idx):
        return self.rtol[idx]

    # ----------------------------------------------------------------
    # ---- character substitution left-to-right
    # ----------------------------------------------------------------
```

```
def ltor_substitution(self,idx):
    return self.ltor[idx]

# -----------------------------------------------------------------
# ---- reset rotor starting position
# -----------------------------------------------------------------
def reset(self,rotor_name,rotor_start):

    self.rotor_name  = rotor_name
    self.rotor_start = rotor_start
    self.rotor_count = 0

    # ---- fill rotor randomized substitution list

    self.lst    = self.rotors[rotor_name]  # indexes list
    self.lstlen = len(self.lst)            # indexes list length

    # ---- fill in the rtol randomized substitution list

    self.rtol = [None] * self.lstlen

    for i in range(self.lstlen):
        self.rtol[i] = self.lst[i]

    # --- shift rtol list so rotor_start is index 0

    if rotor_start >= 0:
       self.rtol = self._shift_rotor(rotor_start,self.rtol)

    # ---- sync rotor's ltor list with it's rotl list

    self.ltor = [None] * self.lstlen

    for i in range(self.abclen):
        self.ltor[self.rtol[i]] = i

# -----------------------------------------------------------------
# ---- helper function: set rotor configuration
# -----------------------------------------------------------------
def _shift_rotor(self,idx,lst):

    # ---- create new list

    new_lst = [-1] * self.lstlen  # set everything to -1

    # --- copy lower half of original list to new list

    i = 0                         # new list index

    for j in range(idx,self.lstlen):
        new_lst[i] = lst[j]   # copy original list to new list
        i += 1                # increment new list index

    # ---- copy upper part of original list to new list

    for j in range(0,idx):
        new_lst[i] = lst[j]   # copy original list to new list
        i += 1                # increment new list index
    ##print(f'new lst: {self.rotor_name} {self.rotor_start}')
    ##print(f'old={lst} new={new_lst}')

    return new_lst

# ---------------------------------------------------------------
# ---- display rotor internals
# ----------------------------------------------------------------
def display(self,title=None,all=True):
    print()
    for k in self.rotors.keys():
        print(f'{k:>4} {self.rotors[k]}')
    print()
    if title:
        print(title)
    if all:
        print('-lst-   l-to-r    r-to-l')
        for i in range(self.lstlen):
            print(f'{i:2} {self.lst[i]:<2}  ' +
                  f'{i:>2} -> {self.ltor[i]:<2}  ' +
                  f'{self.rtol[i]:>2} <- {i:<2}')
    print(f'rotor_name  = {self.rotor_name}')
    print(f'rotor_start = {self.rotor_start}')
    print(f'rotor_count = {self.rotor_count}')
```

EnigmaUtils.py

```
# ===================================================================
# Enigma Machine Simulation - support functions
# ===================================================================

import copy
from   datetime import date
import re
import sys
import random
import user_interface as ui

# -------------------------------------------------------------------
# ---- create a randomize rotor list
# ---- 1. input list must contain a least two values
# ---- 2. input list must contain an even number of values
# ---- 3. this function ensures every list-value will end up in a
# ----    different location (list-index) in the randomized list.
# ----    different locations are required by the enigma machine.
# ----    we are simulating electric circuits. IN can not be the
# ----    same wire as OUT. See the electric circuit diagram
# ----    elsewhere.
# ---- 4. no list-value in the list is allowed to be randomized
# ----    to itself (ie. lst[10] can not be 10)
# -------------------------------------------------------------------

def create_randomized_rotor_list(lst, debug=False):

    ranlst = copy.deepcopy(lst)

    i = 0                     # ranlst list index
    s = 1                     # start of random range
    e = len(ranlst)-1         # end   of random range

    if debug:
        print(f'ranlst: {ranlst}  (original list)')

    while s < e:

        j = random.randint(s,e)

        tmp = ranlst[i]
        ranlst[i] = ranlst[j]
        ranlst[j] = tmp

        if debug:
            print(f'ranlst: {ranlst}')

        i += 1
        s += 1

    # ---- make sure the last two don't encode themselves

    tmp        = ranlst[-1]
    ranlst[-1] = ranlst[-2]
    ranlst[-2] = tmp

    if debug:
        print(f'ranlst: {ranlst}  (last two swapped list)')

    # ---- return the randomized list

    return ranlst

# -------------------------------------------------------------------
# ---- create a randomize reflector list
# ---- 1. input list must contain a least two values
# ---- 2. input list must contain an even number of values
# ---- 3. this function ensures every list element is paired with
# ----    another list element. Each pair is unique and reference
# ----    each other (i.e. lst[6] = 10 and lst[10] = 6)
# -------------------------------------------------------------------

def create_randomized_reflector_list(lst, debug=False):

    def _pop_random(lst):
        idx = random.randrange(0,len(lst))
        return lst.pop(idx)

    cpylst = copy.deepcopy(lst)

    reflst = [None] * len(lst)

    while cpylst:
        idx1 = _pop_random(cpylst)
        idx2 = _pop_random(cpylst)
        reflst[idx1] = idx2
        reflst[idx2] = idx1

    return reflst
```

user_interface.py

```
#! /usr/bin/python3
# ====================================================================
# command line user interface functions
# (see the example code at the end of this file)
# ====================================================================

import os
import sys
import platform

# --------------------------------------------------------------------
# ----running Python3?
# --------------------------------------------------------------------

def running_python3():
    if sys.version_info[0] == 3:
        return True
    return False

# --------------------------------------------------------------------
# ---- prompt the user for input
# --------------------------------------------------------------------

def get_user_input(prompt):
    return input(prompt).strip()

# --------------------------------------------------------------------
# ---- pause program
# --------------------------------------------------------------------

def pause():
    print()
    get_user_input('Press enter to continue ')

# --------------------------------------------------------------------
# ---- clear the terminal screen (window)
# --------------------------------------------------------------------

def clear_screen():
    if platform.system() == 'Linux':
        os.system('clear')
    elif platform.system() == 'Windows':
        os.system('clear')
    else:
        os.system('cls')

# --------------------------------------------------------------------
# ---- Function: convert a string to a float
# --------------------------------------------------------------------

def is_float(s):
    try:
        n = float(s)
        return (True,n)
    except:
        return (False,0.0)

# ------------------------------------------------------------------
# ---- Function: convert a string to an int
# ------------------------------------------------------------------

def is_int(s):
    try:
        n = int(s)
        return (True,n)
    except:
        return (False,0)

def is_integer(s):
    try:
        n = int(s)
        return (True,n)
    except:
        return (False,0)

# ------------------------------------------------------------------
# ---- is a number (int, float, scientific notation)
# ------------------------------------------------------------------

def is_a_number(s):

    x,n = is_int(s)
    if x:
        return True

    x,n = is_float(s)
    if x:
        return True

    return False

# ------------------------------------------------------------------
# ---- main
# ------------------------------------------------------------------

if __name__ == '__main__':

    if not running_python3():
        print()
        print('Must run Python3 - exit program')
        print()
        sys.exit()

    while True:                # loop

        clear_screen()
        print()
        s = get_user_input('Enter data: ')
        if not s:              # empty string?
            break

        (x,n) = is_int(s)
        if x:
            print()
            print(f'You entered the integer {n}')
            pause()
            continue

        (x,n) = is_float(s)
        if x:
            print()
            print(f'You entered the float {n}')
            pause()
            continue

        print()
        print(f'You entered the string "{s}"')
        pause()

    print()
```

create_enigma_config_file.py

```
#!/usr/bin/python3
# ===================================================================
# Enigma Machine Simulation - create configuration file
# ===================================================================

from EnigmaUtils import *

import copy
import re
import sys
import random
import user_interface as ui
from datetime import date

rotors = {'I':[], 'II':[], 'III':[],   'IV':[],
          'V':[], 'VI':[], 'VII':[], 'VIII':[] }


# ---------------------------------------------------------------
# ---- select alphabet
# ----
# ---- NOTES:
# ---- 1. AN ALPHABET MUST CONTAIN AN EVEN NUMBER OF CHARACTERS
# ---- 2. IT MUST NOT CONTAIN A COMMA
# ---------------------------------------------------------------

def select_alphabet():

    # ---- big alphabet (44 characters)
    alphabet0 = [ 'A','B','C','D','E','F','G','H','I','J','K','L',
                  'M','N','O','P','Q','R','S','T','U','V','W','X',
                  'Y','Z','1','2','3','4','5','6','7','8','9','0',
                  '.','=','+','-','/','*','(',')' ]

    # ---- enigma alphabet
    alphabet1 = [ 'A','B','C','D','E','F','G','H','I','J','K','L',
                  'M','N','O','P','Q','R','S','T','U','V','W','X',
                  'Y','Z' ]

    # ---- test alphabet
    alphabet2 = [ 'A','B','C','D','E','F' ]

    # ---- big alphabet (48 characters)
    alphabet3 = [ 'A','B','C','D','E','F','G','H','I','J','K','L',
                  'M','N','O','P','Q','R','S','T','U','V','W','X',
                  'Y','Z','1','2','3','4','5','6','7','8','9','0',
                  '.','=','+','-','/','*','(',')',' ','\\','"','\'' ]


    # ---- alphabet menu
    print('''select an alphabet for enigma configuration file

   option  description
   ------  ------------------------------------------------
```

```
     0     big    alphabet [ A-Z,0-9, ... ] (44 characters)
     1     enigma alphabet [ A-Z ]
     2     test   alphabet [ A-F ]
     3     bigger alphabet [ A-Z,0-9, ... ] (48 characters)
     9     exit program''')

    # ---- make sure the alphabets have an even number of characters

    if len(alphabet0)%2 != 0 or \
       len(alphabet1)%2 != 0 or \
       len(alphabet2)%2 != 0 or \
       len(alphabet3)%2 != 0:
           print()
           print('Internal error: An alphabet contains an odd ')
           print('number of characters - end program')
           sys.exit()

    while True:

        print()
        s =ui.get_user_input('Select alphabet: ')

        if not s:
            sys.exit()

        tf,i = ui.is_int(s)

        if not tf:
            continue

        if i == 0:
            return alphabet0
        if i == 1:
            return alphabet1
        if i == 2:
            return alphabet2
        if i == 3:
            return alphabet3
        if i == 9:
            print()
            sys.exit()

        print()
        print(f'Unlnown alphabet selection (i)')
        continue

# -----------------------------------------------------------------
# ---- create plugboard connections (two unique characters)
# -----------------------------------------------------------------

def add_plugboard_connections(abc):

    plubs = []

    char_in_use = []

    # ---- display the current alphabet
```

```
for i in range(len(abc)):
    if i % 12 == 0:
        print()
        print('Alphabet:',end='')
    print(f' {abc[i]}',end='')
print()

# ---- process user input

while True:

    print()
    s = ui.get_user_input('Enter plugboard connection (ab): ')

    if not s:
        break

    # ---- error check the user's input

    s = s.upper()

    if len(s) != 2 or s[0] == s[1]:
        print()
        print(f'Bad connection entered ({s})')
        continue

    c0 = s[0]
    c1 = s[1]

    if c0 not in abc:
        print()
        print(f'Character {c0} not in the alphabet')
        continue

    if c1 not in abc:
        print()
        print(f'Character {c1} not in the alphabet')
        continue

    if c0 in char_in_use:
        print()
        print(f'Character {c0} already in use')
        continue

    if c1 in char_in_use:
        print()
        print(f'Character {c1} already in use')
        continue

    # ---- save the user's input

    char_in_use.append(c0)
    char_in_use.append(c1)

    plugs.append(s)

return plugs
```

```
# -------------------------------------------------------------------
# ---- main
# -------------------------------------------------------------------

# ---- select an alphabet

abc = select_alphabet()            # alphabet

abclen     = len(abc)              # abc list length
abcidx     = list(range(abclen))   # abc ordered list of indexes
configfile = 'enigma_config.csv'   # configuration file
plugs      = []                    # plugboard pairs list
reflst     = []                    # reflector list
today      = date.today()          # today's date

# ---- write enigma config file

line_count = 0                     # count, lines           written
plug_count = 0                     # count, plugs           written
refl_count = 0                     # count, reflector lists written
roto_count = 0                     # count, rotor lists     written

with open(configfile,"w") as f:

    # ---- add header lines to output file

    f.write(f'# Enigma Machine Simulation Configuration\n')
    line_count += 1

    f.write(f'date,{today}\n')
    line_count += 1

    # ---- add an unknown character substitution to the output file
    # ---- if an input character is not in the alphabet substitute
    # ---- a known character

    if len(abc) < 25:
        f.write(f'unk,{abc[-1]}\n')
    else:
        f.write('unk,X\n')
    line_count += 1

    # ---- output alphabet length

    f.write(f'len,{abclen}\n')
    line_count += 1

    # ---- add alphabet to output file

    f.write(f'# alphabet,list-idx,char\n')
    line_count += 1

    for i in range(abclen):
        f.write(f'abc,{i},{abc[i]}\n')
        line_count += 1

    # ---- create rotor lists and add them to the output file
```

```
for r in rotors:

    ranlst = create_randomized_rotor_list(abcidx)

    rotors[r] = ranlst     # save it for later counting

    f.write(f'# rotor-name,in-list-idx,out-list-idx,in-char,out-char\n')
    line_count += 1

    for i in range(len(ranlst)):
        j = ranlst[i]
        f.write(f'{r},{i},{j},{abc[i]},{abc[j]}\n')
        line_count += 1

    roto_count += 1

# ---- create reflector list and add it to the output file

reflst = create_randomized_reflector_list(abcidx)

f.write(f'# reflector,in-list-idx,out-list-idx,in-char,out-char\n')
line_count += 1

for i in range(abclen):
    j = reflst[i]
    f.write(f'ref,{i},{j},{abc[i]},{abc[j]}\n')
    line_count += 1

# ---- verify reflector pairs

for i in range(abclen):
    j = reflst[i]
    if reflst[j] != i:
        print(f'Error in reflector list i={i} j={j}')
        print(f'reflst[{i}] = {reflst[i]}')
        print(f'reflst[{j}] = {reflst[j]}')
        sys.exit()

refl_count += 1

# ---- add plugboard connections to the output file

plubs = add_plugboard_connections(abc)

l = len(plugs)

print()
if l == 0:
    print('No plugboard connections')
else:
    print(f'{l} plugboard connections')
    f.write(f'# plugboard,char-pair\n')
    line_count += 1
    for i in range(l):
        f.write(f'plug,{plugs[i]}\n')
        line_count += 1
```

```
# ---- display stats

print()
print(f'date         : {today}')
print(f'file     name: {configfile}')
print(f'abc       len: {abclen}')
print(f'alphabet  len: {len(abc)}')
print(f'rotor   count: {len(rotors)}')
for r in rotors.keys():
    print(f'  {r:5}   len: {len(rotors[r])}')
print(f'reflector len: {len(reflst)}')
print(f'plugs        : {len(plugs)}')
print(f'lines written: {line_count}')
print()
```

read_enigma_config_file.py

```
#!/usr/bin/python3
# ==================================================================
# Enigma Machine Simulation - read configuration file
# ==================================================================

import re
import os
import sys
import user_interface as ui

abc        = []
abclen     = 0
abcunk     = None
configfile = 'enigma_config.csv'
date       = None
line_count = 0
plugs      = []
reflst     = []
rotors     = {  'I':[], 'II':[], 'III':[],   'IV':[],
                'V':[], 'VI':[], 'VII':[], 'VIII':[] }

# ------------------------------------------------------------------
# ---- read enigma machine configuration file
# ------------------------------------------------------------------

def EnigmaReadConfigiFile(filename):

    global abc
    global abclen
    global abcunk
    global date
    global line_count
    global plugs
    global reflst
    global rotors

    infile = open(filename,'r')

    for line in infile:

        line_count += 1

        line = line.strip()

        if not line:                # empty string (line)
            continue

        if re.match('^#',line):     # comment?
            continue

        x = line.split(',')

        # --- date file created
        if x[0] =='date':
            date = x[1]
            continue

        # --- alphabet length?

        if x[0] == 'len':          # length
            tf,i = ui.is_int(x[1])
            if tf:
                if i >= 0:
                    abclen = i
                    continue

        # --- unknown character?

        if x[0] == 'unk':          # unknown character
            abcunk = x[1]
            continue

        # --- alphabet?

        if x[0] == 'abc':          # alphabet
            abc.append(x[2])
            continue

        # --- plug?

        if x[0] == 'plug':         # plugboard
            plugs.append(x[1])
            continue

        # ---- reflector?

        if x[0] == 'ref':          # reflector
            reflst.append(int(x[1]))
            continue

        # ---- rotor?

        if x[0] in rotors.keys():  # rotor
            rotors[x[0]].append(x[2])
            continue

        # ---- what?

        print()
        print(f'unknow line in config file (line={line_count})\n({line})')
        sys.exit()

    infile.close()

# -------------------------------------------------------------------
# ---- main
# -------------------------------------------------------------------

# ---- configuration file on command line?

if len(sys.argv) > 1:
    configfile = sys.argv[1]

if not os.path.exists(configfile):
    print()
    print(f'Can not find enigma configuration file ({configfile})')
    print()
    sys.exit()

# ---- read config file

EnigmaReadConfigiFile(configfile)

# ---- now set abcidx

abcidx = list(range(abclen))

# ---- verify config file data

for r in rotors:
    if len(rotors[r]) != abclen:
        print()
        print('bad rotor list length (r)')
        sys.exit()
if len(abc) != abclen:
    print()
    print('bad abc list length\n(alphabet)')
    sys.exit()
if len(reflst) != abclen:
    print()
    print('bad reflector list length\n(reflector)')
    sys.exit()
for i in range(abclen):
    j = reflst[i]
    if reflst[j] != i:
        print(f'error in reflector pair i={i} j={j}')
        sys.exit()

# ---- print stats

print()
print(f'input    file: {configfile}')
print(f'date         : {date}')
print(f'alphabet  len: {len(abc)}')
print(f'  abclen     : {abclen}')
print(f'  abcunk     : {abcunk}')
print(f'rotor   count: {len(rotors)}')
for r in rotors.keys():
    print(f'  {r:5}   len: {len(rotors[r])}')
print(f'reflector len: {len(reflst)}')
print(f'plugs        : {len(plugs)}')
print(f'lines read   : {line_count}')
print()
```

test_enigma_gui.py

```
#!/usr/bin/python3
# ====================================================================
# test Enigma Machine simple GUI
# ====================================================================

from enigma_gui import enigma_gui
from EnigmaConfig import EnigmaConfig
from EnigmaMachine import EnigmaMachine
import PySimpleGUI as sg
import os
import re
import sys

# ---- configuration file

configfile = 'enigma_config.csv'

if len(sys.argv) > 1:
     configfile = sys.argv[1]

if not os.path.exists(configfile):
    print()
    print(f'Can not find enigma configuration file ({configfile})')
    print()
    sys.exit()

# ---- create enigma machine simulation

em = EnigmaMachine(configfile)

# ---- create a GUI window

win,tsize = enigma_gui(em.config)

# ---- alphabet to index

def _char_to_idx(c,abc):
    for i in range(len(abc)):
        if abc[i] == c:
            return i
    return 0

# ---- event loop
x = ''

while(True):

    event,values = win.read()

    ##print('-----------------------------------------')
    ##print(f'event={event}')
    ##print(values)
    ##print('-----------------------------------------')
    # ---- keypress

    if event in em.config.abc:

        if len(x) < tsize:
            x = x + event
            win['-IN-'].update(x)
        continue

    # ---- exit program

    if event in ('Exit',sg.WIN_CLOSED):
        break

    # ---- reset rotors

    if event == 'Reset Rotors':
        x = ''
        win['-IN-'].update(x)
        win['-OUT-'].update(x)
        em.reset()
        continue

    # ---- set rotors

    if event == 'Set Rotors':

        win['-IN-'].update('')
        win['-OUT-'].update('')

        rnam = values['-RROTOR-']
        rpos = values['-RPOSITION-']
        ridx = _char_to_idx(rpos,em.abc)

        mnam = values['-MROTOR-']
        mpos = values['-MPOSITION-']
        midx = _char_to_idx(mpos,em.abc)

        lnam = values['-LROTOR-']
        lpos = values['-LPOSITION-']
        lidx = _char_to_idx(lpos,em.abc)

        nam = [lnam,mnam,rnam]
        pos = [lidx,midx,ridx]
        print(f'names: {nam}  starts: {pos}')
        continue

    # ---- encrypt/decrypt input data

    if event == 'Encrypt/Decrypt':
        continue

    # ---- enigma machine display function

    if event == 'Display':
        em.display_internal_state()
        continue
    # ---- enigma machine display rotor stats

    if event == 'Stats':
        em.display_rotor_stats()
        continue


win.close()
print()
```

test_EnigmaConfig.py

```
#!/usr/bin/python3
# ===================================================================
#
# ===================================================================

from EnigmaConfig import EnigmaConfig
import os
import sys

# ---- configuration file name on the command line?

configfile = 'enigma_config.csv'

if len(sys.argv) > 1:
    configfile = sys.argv[1]

if not os.path.exists(configfile):
    print()
    print(f'Can not find enigma configuration file ({configfile})')
    print()
    sys.exit()

# ---- get configuration

config = EnigmaConfig('enigma_config.csv')

# ---- display configuration

config.display_stats()
```

test_EnigmaMachine.py (command line interface)

```
#!/usr/bin/python3
# ===================================================================
# test the rotors, reflector and plugboard with and without
# rotor advance
# ===================================================================
# the user must advance to rotors with the "advance" command.
# they do not automatically advance.
# -------------------------------------------------------------------
# Note: because this is software and not hardware, we do some special
# "stuff" with characters not in the enigma machine's alphabet [A-Z].
# see the code/comments.
# ===================================================================

from EnigmaMachine import EnigmaMachine
import user_interface as ui
import re
import os
import sys

# -------------------------------------------------------------------
# ---- main
# -------------------------------------------------------------------

if __name__ == '__main__':

    # ---- start enigma machine simulation

    configfile = 'enigma_config.csv'

    if len(sys.argv) > 1:
        configfile = sys.argv[1]

    if not os.path.exists(configfile):
        print()
        print(f'Can not find enigma configuration file ({configfile})')
        print()
        sys.exit()

    em = EnigmaMachine(configfile)

    # ---- user commands

    while True:

        print()
        print('Cmd: text,   advance, reset, set, display, stats')
        print('     autoon, autooff, subdebugon, subdebugoff')

        print()
        s = ui.get_user_input('Enter text: ')
        if not s:
           break

        # ---- display command
```

```
if re.match('^display$',s):
    em.display_internal_state()
    continue

# ---- display rotor stats command

if re.match('^stats$',s):
    em.display_rotor_stats()
    continue

# ---- advance the rotors

if re.match('^advance$',s):
    em.advance_rotors()
    continue

# ---- auto advance

if re.match('^auto$',s):
    print()
    print(f'auto advance ({em.auto_advance_rotors})')
    continue

if re.match('^auton$',s) or re.match('^autoon$',s):
    em.auto_advance_rotors = True
    continue

if re.match('^autoff$',s) or re.match('^autooff$',s):
    em.auto_advance_rotors = False
    continue

# ---- substitution debug

if re.match('^subdebug$',s):
    print()
    print(f'substitution debug ({em.substitution_debug})')
    continue

if re.match('^subdebugon$',s) or re.match('^subon$',s):
    em.substitution_debug = True
    continue

if re.match('^subdebugoff$',s) or re.match('^suboff$',s):
    em.substitution_debug = False
    continue

# ---- reset the rotors to the initial configuration

if re.match('^reset$',s):
    em.reset()
    continue

# ---- set rotor configuration support function

def _get_set_rotor_configuration(r):

    while True:
        print()
        prompt = f'Enter {r} rotor\'s new config (name,start): '
        s = ui.get_user_input(prompt)
        if not s:
            return None

        nn = s.split(',')
        if len(nn) != 2:
            print()
            print('bad input - try again')
            continue

        nam = nn[0].strip().upper()
        num = nn[1].strip()

        if nam not in em.rotorkeys:
            print()
            print('bad input - try again')
            continue

        tf,nnum = ui.is_int(num)
        if not tf or nnum < 0 or nnum > em.abclen-1:
            print()
            print('bad input - try again')
            continue

        break

    return (nam,nnum)

# ---- set the rotors to a specified configuration

if re.match('^set$',s):

    names  = []
    starts = []

    x0 = _get_set_rotor_configuration('left')
    if x0 == None:
        print()
        print('No changees made')
        continue

    x1 = _get_set_rotor_configuration('middle')
    if x1 == None:
        print()
        print('No changees made')
        continue

    x2 = _get_set_rotor_configuration('right')
    if x2 == None:
        print()
        print('No changees made')
        continue

    names.append(x0[0])
    names.append(x1[0])
    names.append(x2[0])

    starts.append(x0[1])
    starts.append(x1[1])
    starts.append(x2[1])

    em.set(names,starts)

    continue


# ---- convert the input text

ss  = s.upper()              # convert input text to uppercase
sss = []

for c in ss:

    if c not in em.abc:
        print(f'char {c} not in alphabet ' +
              f'- substituting {em.abcunk}')
        c = em.abcunk

    n = em.char_to_idx(c)    # convert a character to an index
    ##if n < 0:
    ##    continue

    newidx = em.substitution(n) # substitution cypher

    sss.append(em.abc[newidx])  # collect new characters into a list

ssss = ''.join(sss)          # convert the list to a string

print()
print(f'{s} --> {ssss}')
```

test_EnigmaRotor.py

```
#!/usr/bin/python3
# ====================================================================
# Enigma Machine Simulation - test enigma rotor
# ====================================================================

from EnigmaRotor import EnigmaRotor
from EnigmaConfig import EnigmaConfig
import user_interface as ui
import os
import sys

# --------------------------------------------------------------------
# ---- main - test enigma rotor
# --------------------------------------------------------------------

# ---- get configuration file

configfile = 'enigma_config.csv'

if len(sys.argv) > 1:
    configfile = sys.argv[1]

if not os.path.exists(configfile):
    print()
    print(f'Can not find enigma configuration file ({configfile})')
    print()
    sys.exit()

ec = EnigmaConfig('enigma_config.csv')

keys = list(ec.rotors.keys())

rotor = EnigmaRotor(ec,'I',0)

while True:

    print()
    rotor.display('------Initial configuration ---------',True)

    # ---- get rotor name

    while True:
        print()
        s1 = ui.get_user_input('Enter new rotor name: ')
        if not s1:
            sys.exit()
        if s1 not in keys:
            print()
            print(f'illegal rotor name ({s1})')
            print(f'legal names are {keys}')
            continue
        break

    # ---- get rotor start
```

```
while True:
    print()
    s2 = ui.get_user_input('Enter new rotor start: ')
    if not s2:
        sys.exit()
    tf,x = ui.is_integer(s2)
    if not tf:
        sys.exit()
    if x < 0 or x >= ec.abclen:
        print()
        print(f'illegal rotor start ({s2})')
        print(f'start must be less than {ec.abclen}')
        continue
    break

# ---- adjust rotor

rotor.reset(s1,x)

rotor.display('-----new rotor configurtion -----',True)
```

enigma_machine_w_gui.py (GUI interface)

```
#!/usr/bin/python3
# ===================================================================
# Enigma machine simulation and a GUI
# ===================================================================

from enigma_gui import enigma_gui
from EnigmaConfig import EnigmaConfig
from EnigmaMachine import EnigmaMachine
import PySimpleGUI as sg
import os
import re
import sys

# -------------------------------------------------------------------
# ---- configuration
# -------------------------------------------------------------------

# ---- get configuration file

configfile = 'enigma_config.csv'

if len(sys.argv) > 1:
     configfile = sys.argv[1]

if not os.path.exists(configfile):
    print()
    print(f'Can not find enigma configuration file ({configfile})')
    print()
    sys.exit()

# ---- create an enigma machine object

em = EnigmaMachine(configfile)

# ---- create a GUI window

win,tsize = enigma_gui(em.config)

# -------------------------------------------------------------------
# ----event loop
# -------------------------------------------------------------------

# ---- alphabet to index support function

def _char_to_idx(c,abc):
    for i in range(len(abc)):
        if abc[i] == c:
            return i
    return 0


lst_in  = ''
lst_out = ''
```

```
while(True):

    # ---- wait on an event

    event,values = win.read()

    ##print()
    ##print(f'event = {event}')
    ##print(f'values= {values}')
    ##print()

    # ---- event: exit program

    if event in ('Exit',sg.WIN_CLOSED):
        break

    # ---- event: single character

    if len(event) == 1:
        if event not in em.config.abc:
            print()
            print('internal error - do not recognize event ' +
                 f'({event}) - end program')
            print()
            sys.exit()

        lst_in = lst_in + event
        if len(lst_in) > tsize-1:
            win['-IN-'].update('text buffer overflow')
            win['-OUT-'].update('text buffer overflow')
            continue
        win['-IN-'].update(lst_in)

        idx_in = em.char_to_idx(event)
        idx_out = em.substitution(idx_in)
        lst_out = lst_out + em.idx_to_char(idx_out)
        win['-OUT-'].update(lst_out)

        continue

    # ---- event: reset rotors

    if event == 'Reset Rotors':
        lst_in  = ''
        lst_out = ''
        win['-IN-'].update('')
        win['-OUT-'].update('')
        em.reset()
        continue

    # ---- set rotors

    if event == 'Set Rotors':

        lst_in  = ''
        lst_out = ''
        win['-IN-'].update('')
        win['-OUT-'].update('')
        rnam = values['-RROTOR-']
        rpos = values['-RPOSITION-'].upper()
        ridx = _char_to_idx(rpos,em.abc)

        mnam = values['-MROTOR-']
        mpos = values['-MPOSITION-'].upper()
        midx = _char_to_idx(mpos,em.abc)

        lnam = values['-LROTOR-']
        lpos = values['-LPOSITION-'].upper()
        lidx = _char_to_idx(lpos,em.abc)

        nam = [lnam,mnam,rnam]
        pos = [lidx,midx,ridx]
        em.set(nam,pos)
        continue


    # ---- event: clear encrypt/decrypt buffers

    if event == 'Clear Buffer':
        lst_in  = ''
        lst_out = ''
        win['-IN-'].update('')
        win['-OUT-'].update('')
        continue

    # ---- event: display (enigma machine internals)

    if event == 'Display':
        em.display_internal_state()
        continue

    # ---- event: stats (enigma machine rotor info)

    if event == 'Stats':
        em.display_rotor_stats()
        continue

    # ---- event: auto advance on

    if event == 'Auto Advance On':
        em.auto_advance_on()
        continue

    # ---- event: auto advance off

    if event == 'Auto Advance Off':
        em.auto_advance_off()
        continue

    # ---- event: substitution debug on

    if event == 'Sub Debug On':
        em.sub_debug_on()
        continue
    # ---- event: substitution debug off

    if event == 'Sub Debug Off':
        em.sub_debug_off()
        continue

win.close()
print()
```

enigma_encrypt_decrypt_a_file.py

```
#!/usr/bin/python3
# ==================================================================
# Use the enigma machine encrypt/decrypt a file
#
# Note: for simplicity sake, for this first program
#       a. any characters not in the enigma machine simulation's
#          alphabet, they will pass thru unchanged to the
#          output file.
#       b. the exception to rule "a" is that spaces will be
#          converted to an 'X'.
#       c. all characters in the input file will be converted to
#          uppercase before processing.
# ==================================================================

from EnigmaConfig import EnigmaConfig
from EnigmaMachine import EnigmaMachine
import copy
import os
import sys

inputfile  = 'xx.txt'
outputfile = 'yy.txt'
configfile = 'enigma_config_26.csv'

# ------------------------------------------------------------------
# ---- encrypt/decrypt character
# ------------------------------------------------------------------

def char_to_idx(c,abc):

    for i in range(len(abc)):
        if c == abc[i]:
            return i

    return -1

def convert_char(c,abc,abcunk):

    if c == ' ':
        c = abcunk

    c = c.upper()

    if c in em.abc:
        idx  = char_to_idx(c,abc)
        idxx = em.substitution(idx)
        ##print(f'c={c}  idx={idx}  idxx={idxx}')
        return (True,abc[idxx])

    return (False,c)

# ------------------------------------------------------------------
# ---- main
# ------------------------------------------------------------------
# ---- input/out file on command line?

l = len(sys.argv)

if l != 1:
    if l == 3:
        inputfile  = sys.argv[1]
        outputfile = sys.argv[2]
    else:
       print()
       print(f'wrong number ({l}) of command line arguments')
       print()
       sys.exit()

# ---- create an enigma machine object

em = EnigmaMachine(configfile)

# ---- process input/output files

total_chars     = 0
line_count      = 0
chars_converted = 0

inFile = open(inputfile,'r')
outfile= open(outputfile,'w')

for line in inFile:

    line_count += 1

    total_chars += len(line)

    lst_in = list(line)

    lst_out = copy.copy(lst_in)

    ##print(f'lst_in = {lst_in}')
    ##print(f'lst_out= {lst_out}')

    for i in range(len(lst_in)):
        c = lst_in[i]
        tf,cc = convert_char(c,em.abc,em.abcunk)
        lst_out[i] = cc

        if tf:
            chars_converted += 1

    str_out = ''.join(lst_out)

    outfile.write(str_out)

inFile.close()
outfile.close()

# ---- display processing stats
```

```
print()
print(f'In file        : {inputfile}')
print(f'Out file       : {outputfile}')
print(f'lines read     : {line_count}')
print(f'total chars    : {total_chars}')
print(f'chars converted: {chars_converted}')
print()
```

enigma_config_6.csv

```
# Enigma Machine Simulation Configuration
date,2021-11-23
unk,F
len,6
abc,0,A
abc,1,B
abc,2,C
abc,3,D
abc,4,E
abc,5,F
I,0,4,A,E
I,1,5,B,F
I,2,0,C,A
I,3,2,D,C
I,4,1,E,B
I,5,3,F,D
II,0,5,A,F
II,1,0,B,A
II,2,3,C,D
II,3,1,D,B
II,4,2,E,C
II,5,4,F,E
III,0,4,A,E
III,1,2,B,C
III,2,5,C,F
III,3,0,D,A
III,4,1,E,B
III,5,3,F,D
IV,0,4,A,E
IV,1,5,B,F
IV,2,3,C,D
IV,3,0,D,A
IV,4,1,E,B
IV,5,2,F,C
V,0,5,A,F
V,1,4,B,E
V,2,1,C,B
V,3,2,D,C
V,4,0,E,A
V,5,3,F,D
VI,0,3,A,D
VI,1,4,B,E
VI,2,0,C,A
VI,3,5,D,F
VI,4,2,E,C
VI,5,1,F,B
VII,0,2,A,C
VII,1,5,B,F
VII,2,1,C,B
VII,3,4,D,E
VII,4,0,E,A
VII,5,3,F,D
VIII,0,4,A,E
VIII,1,0,B,A
```

```
VIII,2,5,C,F
VIII,3,1,D,B
VIII,4,2,E,C
VIII,5,3,F,D
ref,0,4,A,E
ref,1,2,B,C
ref,2,1,C,B
ref,3,5,D,F
ref,4,0,E,A
ref,5,3,F,D
plug,AC
plug,BD
```

enigma_config_26.csv

```
# Enigma Machine Simulation Configuration
date,2021-11-23
unk,X
len,26
abc,0,A
abc,1,B
abc,2,C
abc,3,D
abc,4,E
abc,5,F
abc,6,G
abc,7,H
abc,8,I
abc,9,J
abc,10,K
abc,11,L
abc,12,M
abc,13,N
abc,14,O
abc,15,P
abc,16,Q
abc,17,R
abc,18,S
abc,19,T
abc,20,U
abc,21,V
abc,22,W
abc,23,X
abc,24,Y
abc,25,Z
I,0,13,A,N
I,1,2,B,C
I,2,18,C,S
I,3,15,D,P
I,4,20,E,U
I,5,7,F,H
I,6,14,G,O
I,7,4,H,E
I,8,3,I,D
I,9,24,J,Y
I,10,19,K,T
I,11,16,L,Q
I,12,25,M,Z
I,13,1,N,B
I,14,21,O,V
I,15,11,P,L
I,16,9,Q,J
I,17,0,R,A
I,18,8,S,I
I,19,12,T,M
I,20,6,U,G
I,21,23,V,X
I,22,5,W,F
I,23,10,X,K
```

```
I,24,22,Y,W
I,25,17,Z,R
II,0,20,A,U
II,1,15,B,P
II,2,7,C,H
II,3,18,D,S
II,4,11,E,L
II,5,23,F,X
II,6,9,G,J
II,7,21,H,V
II,8,17,I,R
II,9,22,J,W
II,10,12,K,M
II,11,13,L,N
II,12,2,M,C
II,13,24,N,Y
II,14,3,O,D
II,15,19,P,T
II,16,8,Q,I
II,17,4,R,E
II,18,5,S,F
II,19,10,T,K
II,20,25,U,Z
II,21,14,V,O
II,22,16,W,Q
II,23,6,X,G
II,24,0,Y,A
II,25,1,Z,B
III,0,10,A,K
III,1,17,B,R
III,2,20,C,U
III,3,19,D,T
III,4,18,E,S
III,5,25,F,Z
III,6,8,G,I
III,7,5,H,F
III,8,22,I,W
III,9,12,J,M
III,10,16,K,Q
III,11,23,L,X
III,12,0,M,A
III,13,2,N,C
III,14,6,O,G
III,15,7,P,H
III,16,24,Q,Y
III,17,11,R,L
III,18,14,S,O
III,19,1,T,B
III,20,9,U,J
III,21,15,V,P
III,22,21,W,V
III,23,4,X,E
III,24,3,Y,D
III,25,13,Z,N
IV,0,6,A,G
IV,1,11,B,L
IV,2,15,C,P
IV,3,4,D,E
```

```
IV,4,10,E,K
IV,5,24,F,Y
IV,6,20,G,U
IV,7,5,H,F
IV,8,1,I,B
IV,9,23,J,X
IV,10,18,K,S
IV,11,16,L,Q
IV,12,21,M,V
IV,13,25,N,Z
IV,14,0,O,A
IV,15,17,P,R
IV,16,2,Q,C
IV,17,22,R,W
IV,18,19,S,T
IV,19,8,T,I
IV,20,3,U,D
IV,21,13,V,N
IV,22,7,W,H
IV,23,14,X,O
IV,24,12,Y,M
IV,25,9,Z,J
V,0,14,A,O
V,1,21,B,V
V,2,10,C,K
V,3,1,D,B
V,4,15,E,P
V,5,13,F,N
V,6,24,G,Y
V,7,22,H,W
V,8,18,I,S
V,9,6,J,G
V,10,8,K,I
V,11,2,L,C
V,12,4,M,E
V,13,19,N,T
V,14,25,O,Z
V,15,5,P,F
V,16,17,Q,R
V,17,20,R,U
V,18,16,S,Q
V,19,23,T,X
V,20,7,U,H
V,21,9,V,J
V,22,0,W,A
V,23,11,X,L
V,24,12,Y,M
V,25,3,Z,D
VI,0,12,A,M
VI,1,9,B,J
VI,2,22,C,W
VI,3,23,D,X
VI,4,11,E,L
VI,5,13,F,N
VI,6,25,G,Z
VI,7,1,H,B
VI,8,19,I,T
VI,9,14,J,O
```

```
VI,10,0,K,A
VI,11,8,L,I
VI,12,21,M,V
VI,13,24,N,Y
VI,14,10,O,K
VI,15,4,P,E
VI,16,3,Q,D
VI,17,18,R,S
VI,18,6,S,G
VI,19,5,T,F
VI,20,15,U,P
VI,21,16,V,Q
VI,22,7,W,H
VI,23,17,X,R
VI,24,2,Y,C
VI,25,20,Z,U
VII,0,24,A,Y
VII,1,19,B,T
VII,2,16,C,Q
VII,3,17,D,R
VII,4,1,E,B
VII,5,6,F,G
VII,6,0,G,A
VII,7,5,H,F
VII,8,3,I,D
VII,9,7,J,H
VII,10,22,K,W
VII,11,21,L,V
VII,12,15,M,P
VII,13,12,N,M
VII,14,23,O,X
VII,15,8,P,I
VII,16,13,Q,N
VII,17,14,R,O
VII,18,10,S,K
VII,19,25,T,Z
VII,20,11,U,L
VII,21,9,V,J
VII,22,20,W,U
VII,23,18,X,S
VII,24,4,Y,E
VII,25,2,Z,C
VIII,0,24,A,Y
VIII,1,6,B,G
VIII,2,3,C,D
VIII,3,8,D,I
VIII,4,7,E,H
VIII,5,19,F,T
VIII,6,18,G,S
VIII,7,13,H,N
VIII,8,11,I,L
VIII,9,4,J,E
VIII,10,25,K,Z
VIII,11,0,L,A
VIII,12,21,M,V
VIII,13,16,N,Q
VIII,14,20,O,U
VIII,15,23,P,X
```

```
VIII,16,22,Q,W
VIII,17,2,R,C
VIII,18,9,S,J
VIII,19,17,T,R
VIII,20,15,U,P
VIII,21,5,V,F
VIII,22,10,W,K
VIII,23,1,X,B
VIII,24,14,Y,O
VIII,25,12,Z,M
ref,0,8,A,I
ref,1,21,B,V
ref,2,14,C,O
ref,3,23,D,X
ref,4,7,E,H
ref,5,12,F,M
ref,6,18,G,S
ref,7,4,H,E
ref,8,0,I,A
ref,9,19,J,T
ref,10,20,K,U
ref,11,13,L,N
ref,12,5,M,F
ref,13,11,N,L
ref,14,2,O,C
ref,15,16,P,Q
ref,16,15,Q,P
ref,17,24,R,Y
ref,18,6,S,G
ref,19,9,T,J
ref,20,10,U,K
ref,21,1,V,B
ref,22,25,W,Z
ref,23,3,X,D
ref,24,17,Y,R
ref,25,22,Z,W
plug,AC
plug,BD
```

enigma_config_44.csv

```
# Enigma Machine Simulation Configuration
date,2021-11-23
unk,X
len,44
abc,0,A
abc,1,B
abc,2,C
abc,3,D
abc,4,E
abc,5,F
abc,6,G
abc,7,H
abc,8,I
abc,9,J
abc,10,K
abc,11,L
abc,12,M
abc,13,N
abc,14,O
abc,15,P
abc,16,Q
abc,17,R
abc,18,S
abc,19,T
abc,20,U
abc,21,V
abc,22,W
abc,23,X
abc,24,Y
abc,25,Z
abc,26,1
abc,27,2
abc,28,3
abc,29,4
abc,30,5
abc,31,6
abc,32,7
abc,33,8
abc,34,9
abc,35,0
abc,36,.
abc,37,=
abc,38,+
abc,39,-
abc,40,/
abc,41,*
abc,42,(
abc,43,)
I,0,8,A,I
I,1,10,B,K
I,2,3,C,D
I,3,0,D,A
I,4,42,E,(
I,5,21,F,V
I,6,14,G,O
I,7,34,H,9
I,8,13,I,N
I,9,29,J,4
I,10,41,K,*
I,11,33,L,8
I,12,19,M,T
I,13,1,N,B
I,14,16,O,Q
I,15,30,P,5
I,16,37,Q,=
I,17,24,R,Y
I,18,12,S,M
I,19,6,T,G
I,20,7,U,H
I,21,2,V,C
I,22,15,W,P
I,23,4,X,E
I,24,5,Y,F
I,25,20,Z,U
I,26,27,1,2
I,27,11,2,L
I,28,25,3,Z
I,29,32,4,7
I,30,35,5,0
I,31,43,6,)
I,32,17,7,R
I,33,28,8,3
I,34,31,9,6
I,35,36,0,.
I,36,18,.,S
I,37,40,=,/
I,38,23,+,X
I,39,38,-,+
I,40,9,/,J
I,41,26,*,1
I,42,22,(,W
I,43,39,),-
II,0,33,A,8
II,1,21,B,V
II,2,3,C,D
II,3,19,D,T
II,4,31,E,6
II,5,37,F,=
II,6,7,G,H
II,7,17,H,R
II,8,4,I,E
II,9,12,J,M
II,10,22,K,W
II,11,38,L,+
II,12,32,M,7
II,13,8,N,I
II,14,40,O,/
II,15,43,P,)
II,16,23,Q,X
II,17,18,R,S
II,18,28,S,3
II,19,16,T,Q
```

```
II,20,11,U,L
II,21,36,V,.
II,22,25,W,Z
II,23,10,X,K
II,24,27,Y,2
II,25,6,Z,G
II,26,0,1,A
II,27,39,2,-
II,28,42,3,(
II,29,41,4,*
II,30,15,5,P
II,31,35,6,0
II,32,20,7,U
II,33,1,8,B
II,34,29,9,4
II,35,30,0,5
II,36,9,.,J
II,37,14,=,O
II,38,13,+,N
II,39,2,-,C
II,40,26,/,1
II,41,24,*,Y
II,42,5,(,F
II,43,34,),9
III,0,26,A,1
III,1,29,B,4
III,2,5,C,F
III,3,43,D,)
III,4,11,E,L
III,5,12,F,M
III,6,14,G,O
III,7,20,H,U
III,8,33,I,8
III,9,27,J,2
III,10,22,K,W
III,11,32,L,7
III,12,13,M,N
III,13,4,N,E
III,14,42,O,(
III,15,8,P,I
III,16,31,Q,6
III,17,35,R,0
III,18,10,S,K
III,19,1,T,B
III,20,36,U,.
III,21,7,V,H
III,22,17,W,R
III,23,18,X,S
III,24,40,Y,/
III,25,39,Z,-
III,26,16,1,Q
III,27,28,2,3
III,28,30,3,5
III,29,23,4,X
III,30,21,5,V
III,31,37,6,=
III,32,25,7,Z
III,33,6,8,G
```

```
III,34,9,9,J
III,35,41,0,*
III,36,15,.,P
III,37,19,=,T
III,38,3,+,D
III,39,0,-,A
III,40,38,/,+
III,41,34,*,9
III,42,24,(,Y
III,43,2,),C
IV,0,26,A,1
IV,1,25,B,Z
IV,2,31,C,6
IV,3,33,D,8
IV,4,21,E,V
IV,5,28,F,3
IV,6,38,G,+
IV,7,23,H,X
IV,8,1,I,B
IV,9,40,J,/
IV,10,35,K,0
IV,11,12,L,M
IV,12,34,M,9
IV,13,37,N,=
IV,14,17,O,R
IV,15,36,P,.
IV,16,39,Q,-
IV,17,19,R,T
IV,18,9,S,J
IV,19,42,T,(
IV,20,41,U,*
IV,21,20,V,U
IV,22,43,W,)
IV,23,18,X,S
IV,24,13,Y,N
IV,25,11,Z,L
IV,26,10,1,K
IV,27,7,2,H
IV,28,27,3,2
IV,29,6,4,G
IV,30,22,5,W
IV,31,3,6,D
IV,32,5,7,F
IV,33,4,8,E
IV,34,0,9,A
IV,35,29,0,4
IV,36,8,.,I
IV,37,32,=,7
IV,38,30,+,5
IV,39,24,-,Y
IV,40,2,/,C
IV,41,14,*,O
IV,42,15,(,P
IV,43,16,),Q
V,0,26,A,1
V,1,2,B,C
V,2,33,C,8
V,3,24,D,Y
```

```
V,4,43,E,)
V,5,34,F,9
V,6,7,G,H
V,7,36,H,.
V,8,6,I,G
V,9,21,J,V
V,10,41,K,*
V,11,37,L,=
V,12,14,M,O
V,13,1,N,B
V,14,5,O,F
V,15,38,P,+
V,16,15,Q,P
V,17,28,R,3
V,18,25,S,Z
V,19,30,T,5
V,20,3,U,D
V,21,18,V,S
V,22,35,W,0
V,23,16,X,Q
V,24,0,Y,A
V,25,10,Z,K
V,26,40,1,/
V,27,42,2,(
V,28,4,3,E
V,29,11,4,L
V,30,8,5,I
V,31,27,6,2
V,32,19,7,T
V,33,23,8,X
V,34,17,9,R
V,35,20,0,U
V,36,39,.,-
V,37,13,=,N
V,38,9,+,J
V,39,29,-,4
V,40,32,/,7
V,41,12,*,M
V,42,22,(,W
V,43,31,),6
VI,0,33,A,8
VI,1,37,B,=
VI,2,26,C,1
VI,3,4,D,E
VI,4,31,E,6
VI,5,39,F,-
VI,6,18,G,S
VI,7,14,H,O
VI,8,13,I,N
VI,9,0,J,A
VI,10,15,K,P
VI,11,38,L,+
VI,12,7,M,H
VI,13,30,N,5
VI,14,36,O,.
VI,15,17,P,R
VI,16,29,Q,4
VI,17,12,R,M
```

```
VI,18,20,S,U
VI,19,32,T,7
VI,20,16,U,Q
VI,21,34,V,9
VI,22,8,W,I
VI,23,27,X,2
VI,24,9,Y,J
VI,25,2,Z,C
VI,26,22,1,W
VI,27,24,2,Y
VI,28,10,3,K
VI,29,5,4,F
VI,30,42,5,(
VI,31,19,6,T
VI,32,6,7,G
VI,33,43,8,)
VI,34,23,9,X
VI,35,1,0,B
VI,36,35,.,0
VI,37,25,=,Z
VI,38,21,+,V
VI,39,40,-,/
VI,40,41,/,*
VI,41,28,*,3
VI,42,11,(,L
VI,43,3,),D
VII,0,18,A,S
VII,1,26,B,1
VII,2,36,C,.
VII,3,33,D,8
VII,4,16,E,Q
VII,5,0,F,A
VII,6,14,G,O
VII,7,38,H,+
VII,8,1,I,B
VII,9,8,J,I
VII,10,22,K,W
VII,11,39,L,-
VII,12,37,M,=
VII,13,32,N,7
VII,14,34,O,9
VII,15,4,P,E
VII,16,19,Q,T
VII,17,6,R,G
VII,18,25,S,Z
VII,19,10,T,K
VII,20,9,U,J
VII,21,7,V,H
VII,22,3,W,D
VII,23,15,X,P
VII,24,13,Y,N
VII,25,41,Z,*
VII,26,17,1,R
VII,27,42,2,(
VII,28,20,3,U
VII,29,28,4,3
VII,30,27,5,2
VII,31,2,6,C
VII,32,43,7,)
VII,33,35,8,0
VII,34,30,9,5
VII,35,29,0,4
VII,36,23,.,X
VII,37,40,=,/
VII,38,5,+,F
VII,39,24,-,Y
VII,40,21,/,V
VII,41,11,*,L
VII,42,12,(,M
VII,43,31,),6
VIII,0,15,A,P
VIII,1,24,B,Y
VIII,2,31,C,6
VIII,3,20,D,U
VIII,4,39,E,-
VIII,5,11,F,L
VIII,6,8,G,I
VIII,7,43,H,)
VIII,8,1,I,B
VIII,9,36,J,.
VIII,10,13,K,N
VIII,11,40,L,/
VIII,12,10,M,K
VIII,13,38,N,+
VIII,14,25,O,Z
VIII,15,3,P,D
VIII,16,5,Q,F
VIII,17,33,R,8
VIII,18,14,S,O
VIII,19,42,T,(
VIII,20,17,U,R
VIII,21,22,V,W
VIII,22,7,W,H
VIII,23,16,X,Q
VIII,24,2,Y,C
VIII,25,12,Z,M
VIII,26,9,1,J
VIII,27,28,2,3
VIII,28,4,3,E
VIII,29,21,4,V
VIII,30,18,5,S
VIII,31,29,6,4
VIII,32,6,7,G
VIII,33,32,8,7
VIII,34,26,9,1
VIII,35,30,0,5
VIII,36,23,.,X
VIII,37,35,=,0
VIII,38,0,+,A
VIII,39,34,-,9
VIII,40,19,/,T
VIII,41,27,*,2
VIII,42,37,(,=
VIII,43,41,),*
ref,0,28,A,3
ref,1,14,B,O
```

```
ref,2,21,C,V
ref,3,23,D,X
ref,4,38,E,+
ref,5,37,F,=
ref,6,12,G,M
ref,7,13,H,N
ref,8,34,I,9
ref,9,11,J,L
ref,10,42,K,(
ref,11,9,L,J
ref,12,6,M,G
ref,13,7,N,H
ref,14,1,O,B
ref,15,27,P,2
ref,16,35,Q,0
ref,17,32,R,7
ref,18,25,S,Z
ref,19,30,T,5
ref,20,31,U,6
ref,21,2,V,C
ref,22,36,W,.
ref,23,3,X,D
ref,24,33,Y,8
ref,25,18,Z,S
ref,26,39,1,-
ref,27,15,2,P
ref,28,0,3,A
ref,29,40,4,/
ref,30,19,5,T
ref,31,20,6,U
ref,32,17,7,R
ref,33,24,8,Y
ref,34,8,9,I
ref,35,16,0,Q
ref,36,22,.,W
ref,37,5,=,F
ref,38,4,+,E
ref,39,26,-,1
ref,40,29,/,4
ref,41,43,*,)
ref,42,10,(,K
ref,43,41,),*
plug,AB
```

demo_list_copy_1.py (a=b)

```
#!/usr/bin/python3
# ====================================================================
# test: copy deep list
#       list1 = copy.copy(list1) and copy.deepcopy(list1)
# ====================================================================

import copy

print('\n---- original')
list1 = [[0,1],2,3,4,5]
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')

print('\n---- copy')
list2 = copy.copy(list1)
print('---- modify element [0][0]')
list2[0][0] = 'A'
print(f'list2       id = {id(list2)}  list={list2}')
print(f'list2[0][0] id = {id(list2[0][0])}')
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')

print('\n---- deepcopy')
list3 = copy.deepcopy(list1)
print('---- modify element [0][0]')
list3[0][0] = 'B'
print(f'list3       id = {id(list3)}  list={list3}')
print(f'list3[0][0] id = {id(list3[0][0])}')
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')
```

demo_list_copy_2.py (copy)

```
#!/usr/bin/python3
# ====================================================================
# test: copy deep list
#       list1 = copy.copy(list1) and copy.deepcopy(list1)
# ====================================================================

import copy

print('\n---- original')
list1 = [[0,1],2,3,4,5]
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')

print('\n---- copy')
list2 = copy.copy(list1)
print('---- modify element [0][0]')
list2[0][0] = 'A'
print(f'list2       id = {id(list2)}  list={list2}')
print(f'list2[0][0] id = {id(list2[0][0])}')
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')

print('\n---- deepcopy')
list3 = copy.deepcopy(list1)
print('---- modify element [0][0]')
list3[0][0] = 'B'
print(f'list3       id = {id(list3)}  list={list3}')
print(f'list3[0][0] id = {id(list3[0][0])}')
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')
```

demo_list_copy_3.py (deepcopy)

```
#!/usr/bin/python3
# ====================================================================
#
# ====================================================================

import copy

print('\n---- original')
list1 = [[0,1],2,3,4,5]
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')

print('\n---- copy')
list2 = copy.copy(list1)
print('---- modify element [0][0]')
list2[0][0] = 'A'
print(f'list2       id = {id(list2)}  list={list2}')
print(f'list2[0][0] id = {id(list2[0][0])}')
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')

print('\n---- deepcopy')
list3 = copy.deepcopy(list1)
print('---- modify element [0][0]')
list3[0][0] = 'B'
print(f'list3       id = {id(list3)}  list={list3}')
print(f'list3[0][0] id = {id(list3[0][0])}')
print(f'list1       id = {id(list1)}  list={list1}')
print(f'list1[0][0] id = {id(list1[0][0])}')
```

demo_pysimplegui_combo.py

```
#!/usr/bin/python3
# ====================================================================
# example from: stackoverflow.com/questions/75088895/
#               getting-the-value-or-index-of-combobox-itm
# ====================================================================

import PySimpleGUI as sg

#font2     = ('Aerial',12)
#font4     = ('Aerial',14)
font4      = ('Courier New',14)
font6      = ('Courier New',16)
font8      = ('Courier New',18)

clst = ['choice1','choice2','choice3']

layout = [ [sg.Combo(clst,
                    enable_events=True,
                    default_value=' do it ',
                    font=font8,
                    key='combo')],
           [sg.Button('Test',font=font8),
            sg.Exit(font=font8)] ]

win = sg.Window('combo test',layout)

while True:

    event,values = win.read()
    if event is None or event == 'Exit':
        break

    print()
    print(f'event = {event}')
    print(f'values= {values}')

    if event == 'Test':
        combo = values['combo']  # use the conbo key
        print()
        print(f'combo = {combo}')

win.Close()
print()
```

demo_rotate_list_01.py (good but insufficient for enigma rotor)

```
#!/usr/bin/python3
# ====================================================================
# rotate a python list one position
# note: this code is insufficient for enigma machine rotor rotation
#       remember, the list index and its list value represents
#       an electronic circuit (wire). the wire starts and an index
#       and ends a specific number of indexes away.
#       the wire's end point (list values) must also be moved.
# ====================================================================

lst    = ['a','b','c','d','e','f']  # list
lstlen = len(lst)                   # list length

print()
print(lst)

for i in range(lstlen):

    print(f'[{i:>2}]  {lst})   # display initial list

    # ---- move list values up one position

    tmp = lst[0]

    for i in range(lstlen-1):
        lst[i] = lst[i+1]

    lst[-1] = tmp

    print(f'[{i:>2}]  {lst})   # display rotated list
```

demo_rotate_list_02.py (good but insufficient for enigma rotor)

```
#!/usr/bin/python3
# ===================================================================
# rotate a python list one position
# (save the rotated lists and then dislay them at the end)
#
# can you think of a way to rotate a list with the mod operator?
# ===================================================================

import copy

lst    = ['a','b','c','d','e','f']  # list
lsts   = []                         # list of lists

# ---- return a new rotated (one position) list
def rotate_list(lst):
    newlst = copy.copy(lst)
    tmp = newlst[0]
    for i in range(len(newlst)-1):
        newlst[i] = newlst[i+1]
    newlst[-1] = tmp
    return newlst

# ---- create list of lists

lsts.append(lst)    # starting list

print(lsts)
print(lsts[-1])

for _ in range(10):
    rotlst = rotate_list(lsts[-1])
    lsts.append(rotlst)

# ---- display lists

print()
print('Rotate a list one position at a time')

print()
for i in range(len(lst)):
    for j in range(len(lsts)):
        print(f'{lsts[j][i]}  ',end='')
    print()
print()
```

demo_rotate_enigma_rotor_list.py

```
#!/usr/bin/python3
# ======================================================================
# use a python list to simulate rotating an enigma rotor
# the rotor is a hard wired substitution cipher and rotates "upward"
# or "forward" one position (a -> b -> c -> ...)
#
# a list index is an index into a alphabet list used to translate
# between indexes and alphabetic letters
#
# the list index is the IN character and the list value is the index
# of the OUT character
#
# ---- pattern the code is trying to duplicate ----------------------
#
#    [0]     [1]     [2]     [3]     [4]
#   IN OUT  IN OUT  IN OUT  IN OUT  IN OUT
#   0 -> 2  0 -> 2  0 -> 3  0 -> 1  0 -> 2
#   1 -> 3  1 -> 0  1 -> 2  1 -> 3  1 -> 3
#   2 -> 1  2 -> 3  2 -> 0  2 -> 0  2 -> 1
#   3 -> 0  3 -> 1  3 -> 1  3 -> 2  3 -> 0
#
# ======================================================================

abc    = ['A','B','C','D']     # alphabet
lst    = [2,3,1,0]             # randomized list
lstlen = len(lst)              # list length

# ---- display the current index list

def display_list(n,lst,lstlen,abc):
    print(f'[{n}] {lst}  [',end='')
    for i in range(lstlen):
        if i != 0: print(',',end='')
        print(f' {abc[lst[i]]}',end='')
    print(' ]')


# ---- main

print()
display_list(0,lst,lstlen,abc)

for j in range(1,8):           # rotate and display rotor list

    # ---- rotate the list (enigma rotor)
    # ---- move the list values up one position (a->b)
    # ---- modify the list value to be the index of the OUT
    # ---- character

    tmp = lst[0]

    for i in range(lstlen-1):
       lst[i] = (lst[i+1] - 1) % lstlen
```

```
lst[-1] = (tmp-1) % lstlen

if j % lstlen == 0:
    print()
display_list(j,lst,lstlen,abc)
```

demo_substitute_char.py

```
#!/usr/bin/python3
# ====================================================================
#
# ====================================================================


abc    = [ 'A','B','C','D','E','F','G','H' ]

abcunk = 'X'
txt    = 'abcz87d'             # test
txtlst = list(txt)

print()
print(f'txt    = {txt}')
print(f'txtlst = {txtlst}')

# ---- substitute (abcunk) for characters not in alphabet (abc)

for i in range(len(txtlst)):

    if txtlst[i].upper() not in abc:
        txtlst[i] = abcunk

newtxt = ''.join(txtlst)       # change list to a string

print(f'newtxt = {newtxt}')
```

demo_sys_argv.py

```
#!/usr/bin/python3
# ===================================================================
# Test command line arguments
# ===================================================================

import sys

# ---- remove first argument

print()
print(f'before: {sys.argv}')
sys.argv.pop(0)
print(f'after : {sys.argv}')

# ---- print remaining argument list

print()
for i in range(len(sys.argv)):
    print(f'[{i}]  {sys.argv[i]}')
print()

# ---- print remaining argument list length

print(f'argument length is {len(sys.argv)}')
```