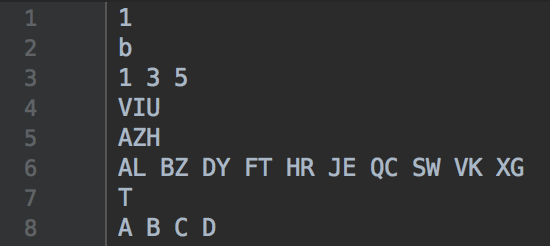
# 1. How To Set Up The Text File

The layout of the text file should be (each on its own line):

1. **Day**: The Day Number  
   ---Type: Must be a whole number
2. **Reflector**: The reflector letter, either “b” or “c”  
   ---Type: Must be a single letter “b” or “c”
3. **Wheels**: The rotor wheel order  
   ---Type: Must be numbers(1-8), number of rotors must be 3.
4. **Ring**: Offsets for each rotor  
   ---NOT USED YET
5. **Ground Starting Point**: Where each Rotor would start  
   ---NOT USED YET
6. **Plugs**: Plug board settings  
   ---Type: Must be letters  
   --Info: Groups in pairs. The first letter of the pair will be replaced by the second letter
7. **Transmit/Receive**: Either Encrypt or Decrypt  
   ---Type: Must be a single letter “T” or “F”
8. **Message**: The message that is decrypted or encrypted  
   ---Type: Must be letters, but character count must not exceed 250.

Example:

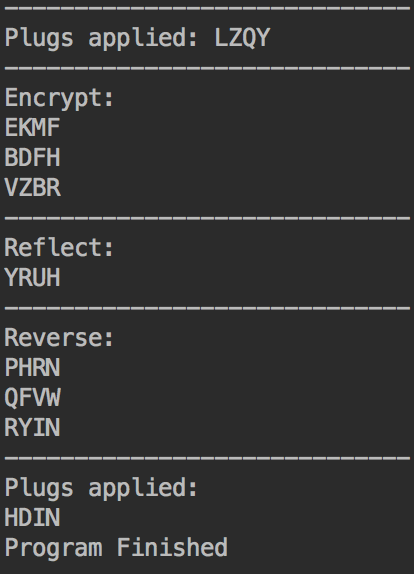


# 2. How To Run The Program

### A. After setting up the User Text File correctly save it as “EnigmaFile”.

## B. Run the program.

Example Of Run Code With Above Settings:



# 3. What Any Messages To The User Might Mean (Errors)

Error #000 **= DAY Input Error, Check your input for DAY, as it might be not entered or in the wrong spot.**

Error #001 **=REFLECTOR Input Error, Check your input for REFLECTOR, as it might be not entered or in the wrong spot.**

Error #002 **= WHEELS Input Error, Check your input for WHEELS, as it might be not entered or in the wrong spot.**

Error #003 **= RING Input Error Check your input for RING, as it might be not entered or in the wrong spot.**

Error #004 **= GROUND Input Error, Check your input for GROUND, as it might be not entered or in the wrong spot.**

Error #005 **= PLUGS Input Error, Check your input for PLUGS, as it might be not entered or in the wrong spot.**

Error #006 **= TRANSMIT/RECEIVE Input Error, Check your input for TRANSMIT/RECEIVE, as it might be not entered or in the wrong spot.**

Error #007 **= MESSAGE Input Error, Check your input for MESSAGE, as it might be not entered or in the wrong spot.**

Error #111 **= Input Error, check your input values and make sure they replicate the example above.**

Error #201 **= Neither B or C Reflector. Your User Input Text is not “B” or “C”.**

Error #202 **= Neither Receiver nor Transceiver. Your User Input Text is not “T” or “R”.**

Error #203 **= Plugs must be letters. Your User Input Text is not all letters.**

Error #204 **= Plugs are not even. Your User Input Text must have pairs of plugs.**

Error #205 **= Wheels are not 1-8. Your User Input Text must reflect this.**

Error #206 **= Number of rotors is not equal to 3. Your User Input Text must have 3 rotors.**

Error #999 **= Your message is too large, the limit for a message is 250 character limits.**

# 4. Additional Information on the Program

**Functions**

1. **def encryptByRotor(curRotor, nextRotor, inString) Inferred type: (curRotor: {index}, nextRotor: {\_\_getitem\_\_}, inString: Any) -> str**

**Purpose: Encrypts by given rotor   
:param curRotor: Has current rotor   
:param nextRotor: Has next rotor   
:param inString: String to be encrypted  
 :return: The result of inString being encrypted**

1. **def textedit() Inferred type: () -> List[str]**

**Purpose: Grabs from file, reads it, splits string by new line.   
:param NONE:   
:return: The result of what is in the read file with the new lines stripped.**

1. **def getMasterRotorList() Inferred type: () -> List[list]**

**Purpose: contains rotors I to VIII   
:param NONE:  
:return: the rotors**

1. **def getReflectorList() Inferred type: () -> List[list]**

**Purpose: contains two reflectors. The official system calls them 'b' and 'c'   
:param NONE:  
:return: the reflector list**

1. **def plugboard(trans, message) Inferred type: (trans: Any, message: {replace}) -> {replace}**

**Purpose: replaces the selected letters with what is to be replaced  
:param trans: a list of plugboard pairs   
:param message: the incoming message to be encoded   
:return: the message with the letters swapped**

1. **def encrypt(plugs, startingString, numberOfRotors, dailyScheduledRotors, reflectorRequest, TR) Inferred type: (plugs: Any, startingString: Any, numberOfRotors: Any, dailyScheduledRotors: Any, reflectorRequest: Any, TR: {\_\_eq\_\_}) -> str**

**Purpose: Encrypts or Decrypts the given string depending on Receiveing or Sending :param plugs: the plugboard settings**

**:param startingString: The message to be translated**

**:param numberOfRotors: How many rotors, including the zeroth one**

**:param dailyScheduledRotors:Today's rotor list**

**:param reflectorRequest: which reflector**

**:param TR: a boolean for transmit/receive**

**:return: the encrypted and decrypted message**

1. **def main() Inferred type: () -> None**

**Purpose: Calls textedit, organizes and cleans information from User Text File to certain variables, then calls encrypt  
 :param: NONE  
 :return NONE:**

1. **def lengthCheck(message) Inferred type: (message: Any) -> Any**

**Purpose: Checks the length of the message   
:param message: The message to be checked   
:return: NONE**

1. **def missingCheck(textlist) Inferred type: (textlist: {\_\_getitem\_\_}) -> None**

**Purpose: Checks if Data is missing   
:param textlist: list to be checked  
:return: NONE**

1. **def checkReflector(reflector) Inferred type: (reflector: {upper, \_\_eq\_\_}) -> None**

**Purpose: Checks if Relfector is a a valid input**

**:param reflector: Reflector to check**

**:return: NONE**

1. **def checkPlugs(plugs) Inferred type: (plugs: Any) -> None**

**Purpose: Checks if plugs are all letters and if they are pairs**

**:param plugs: What is to be tested   
:return: NONE**

1. **def wheelCheck(wheels) Inferred type: (wheels: {\_\_getitem\_\_}) -> None**

**Purpose: Checks if wheels given are between 1 and 8   
:param wheels: What is to be tested   
:return: NONE**

1. **def numberOfRotorsCheck(amount) Inferred type: (amount: {\_\_lt\_\_, \_\_gt\_\_}) -> None**

**Purpose: Counts the amount of rotors and makes sure it is in between 1 and 4. :param amount: What is to be tested.   
:return: NONE**

1. **def numberOfRotorsCheck(amount) Inferred type: (amount: {\_\_lt\_\_, \_\_gt\_\_}) -> None**

**Purpose: Counts the amount of rotors and makes sure it is in between 1 and 4. :param amount: What is to be tested.   
:return: NONE**