Enigma Machine

Eric Oliver

December 5, 2017

```
#Function that establishes the order of the rotors for encryption
def rotor_Order(a, b, c):
    slot[0] = a
    slot[1] = b
    slot[2] = c
    return

#Function that places the rotors and the reverse rotors at their beginning
    positions
def set_rotorPosition():
    for i in range(0,3):
        temp1 = rotors[slot[i]-1][:]
        temp2 = rotorsRev[slot[i] - 1][:]
        for k in range(0,26):
            rotors[slot[i] - 1][k]= temp1[(k + tick[i]) % 26]
            rotorsRev[slot[i] - 1][k] = temp2[(k + tick[i]) % 26]
        #print(rotors[slot[i]-1][:])
    return

#Function that turns the rotors and reverse rotor one position
def rotor_moveOne(i):
    temp1 = rotors[slot[i] - 1][:]
    temp2 = rotorsRev[slot[i] - 1][:]
    for k in range(0,26):
        rotors[slot[i] - 1][k]= temp1[(k + 1) % 26]
        rotorsRev[slot[i] - 1][k] = temp2[(k + 1) % 26]
    #print(rotors[slot[i]-1][:])
    return

#Function that establishes the beginning position of the rotors and reverse
    rotors
def rotor_Set(a, b, c):
    tick[0] = a
    tick[1] = b
    tick[2] = c
    set_rotorPosition()
    return

#Function that sets up the plug board
def plug_board(plugs):
    for i in range(0, plugs):
        letterOne = ord(keyArray[10+(2*i)]) - 97
        letterTwo = ord(keyArray[11+(2*i)]) - 97
        checkSwap[letterOne] = 1
        checkSwap[letterTwo] = 1
        temp = plugboard[letterOne]
        plugboard[letterOne] = plugboard[letterTwo]
        plugboard[letterTwo] = temp
    #print(plugboard)
    return

#Function that swaps code numbers based on the plugboard settings
def plugSwap(codeIndex):
    codeArray[codeIndex] = plugboard[codeArray[codeIndex]]
```

```
    return

#Function that recodes an incoming letter with the current rotor
def rotorCoding(codeIndex, rotorNum):
    codeArray[codeIndex] = (codeArray[codeIndex] + rotors[slot[rotorNum]-1][
    codeArray[codeIndex]])%26
    return

#Function that recodes an incoming letter with the current rotor in the
    reverse direction
def rotorCodingRev(codeIndex, rotorNum):
    codeArray[codeIndex] = (codeArray[codeIndex] + rotorsRev[slot[rotorNum
    ]-1][codeArray[codeIndex]])%26
    return

#Function Setup Enigma
def Enigma_setup():
    rotor_Order(int(keyArray[0]), int(keyArray[1]), int(keyArray[2]))
    rotorP1 = [keyArray[3], keyArray[4]]
    rotorP2 = [keyArray[5], keyArray[6]]
    rotorP3 = [keyArray[7], keyArray[8]]
    rotorP1 = ''.join(rotorP1)
    rotorP2 = ''.join(rotorP2)
    rotorP3 = ''.join(rotorP3)
    rotor_Set(int(rotorP1), int(rotorP2), int(rotorP3))
    plugs = int(keyArray[9])
    #print(plugs)
    plug_board(plugs)
    return

#Function Encrypt/Decrypt message
def encrypt_decrypt():
    # This is the coding section. It calls the functions of rotors and reverse
     rotors
    # The encoding/decoding happens for letter in the array
    for m in range(0, codeLength):
        # Plugswap happens at the beginning based on the plugboard settings
        plugSwap(m)

        # Sends the numbers through all 3 rotors in forward order
        for n in range(0, 3):
            rotorCoding(m, n)

        # Simple Caesar shift for the reflector
        codeArray[m] = (codeArray[m] + 13) % 26

        # Sends the numbers through all 3 rotors in reverse order
        for n in range(2, -1, -1):
            rotorCodingRev(m, n)

        # Once the numbers have exited, they go through the plugboard again
        plugSwap(m)

        # Increments the first rotor by one after each letter is encoded
```

```
        # if the first rotor goes past 25, then the next rotor is incremented by one
        # if not then the range is maxxed to exit the loop
        for p in range(0, 3):
            temp = (tick[p] + 1) % 26
            rotor_moveOne(p)
            if (temp > tick[p]):
                tick[p] = temp
                p = 3
            else:
                tick[p] = temp
    # Converts the 0-25 numbers into ASCII numbers and then back into characters
    # for the textArray String
    for m in range(0, codeLength):
        textArray[m] = chr(codeArray[m] + 97)
    return

# DATA SECTION ------------------------------------------------------------------
#The hard coded setting of the ZERO position of the three rotors
rotors = [ [3, 15, 20, 22, 20, 12, 24, 6, 4, 18, 11, 7, 19, 10, 1, 11, 16, 20, 15, 15, 8, 24, 14, 23, 16, 10],
           [9, 14, 8, 16, 24, 18, 23, 7, 13, 24, 6, 1, 12, 17, 3, 19, 4, 14, 7, 18, 2, 11, 22, 3, 15, 2],
           [16, 24, 12, 14, 11, 13, 20, 12, 5, 3, 17, 9, 16, 24, 7, 14, 20, 5, 12, 16, 3, 10, 12, 1, 8, 8]]
rotorsRev = [ [0 for i in range(26)],[0 for i in range(26)],[0 for i in range(26)]]
plugboard = [i for i in range(26)]
checkSwap = [0 for i in range(26)]
tick = [ 0, 0, 0]
slot = [-1, -1, -1]
rotorPosition = [0,0,0]


#Loop sets up the reverse coded setting of the three rotors
for i in range(0,3):
    for j in range(0,26):
        rotorsRev[i][(j + (rotors[i][j]))%26] = 26-(rotors[i][j])

rotorSafe = rotors
rotorsRevSafe = rotorsRev
plugboardSafe = plugboard
checkSwapSafe = checkSwap
tickSafe = tick
slotSafe = slot
rotorPositionSafe = rotorPosition


#MAIN PROGRAM SECTION----------------------------------------------------------*
#Key Input
#first 3 characters = rotor order (123, 132, 213, 231, 312, 321)
#next 6 characters = 2-digit rotor setting per rotor (00-25)
#next character = number of plugs (0-9)
```

```
#last characters = letters of the plugs to be swapped
print ("Input the key for encryption/decryption.")
someKey = input()
keyArray = list(someKey)

Enigma_setup()

#Enter text to be encrypted or decrypted--------------------------*
print ('Enter text to be encrypted/decrypted (lowercase letters only) -')
someText = input()

#This converts a string into a character array
textArray = list(someText)

#Converts the string array into an (ASCII array - 97)
codeLength = len(textArray)
codeArray = [0 for i in range(codeLength)]
for i in range(0,codeLength):
    codeArray[i] = ord(textArray[i])-97

#Encrypt or Decrypt message
encrypt_decrypt()

#Converts a character array back into a string
newText = ''.join(textArray)

#Ecrypted/Decrypted output----------------------------------------*
print(newText)
```