program for monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from which the cipher sequence can be generated. For example, using the keyword CIPHER, write out the keyword followed by unused letters in normal order and match this against the plaintext letters: plain: 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 cipher: C I P H E R A B D F G J K L M N O Q S T U V W X Y Z

program:

def generate\_cipher\_alphabet(keyword):

# Remove duplicates while preserving order

keyword = "".join(sorted(set(keyword.upper()), key=keyword.upper().index))

alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

# Create cipher alphabet

cipher = keyword + "".join([c for c in alphabet if c not in keyword])

return cipher

def encrypt(plaintext, cipher):

alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

result = ""

for ch in plaintext.upper():

if ch in alphabet:

result += cipher[alphabet.index(ch)]

else:

result += ch

return result

def decrypt(ciphertext, cipher):

alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

result = ""

for ch in ciphertext.upper():

if ch in cipher:

result += alphabet[cipher.index(ch)]

else:

result += ch

return result

# --- MAIN PROGRAM ---

keyword = "CIPHER"

cipher\_alphabet = generate\_cipher\_alphabet(keyword)

print("Cipher Alphabet:", cipher\_alphabet)

plaintext = "HELLO WORLD"

ciphertext = encrypt(plaintext, cipher\_alphabet)

decrypted = decrypt(ciphertext, cipher\_alphabet)

print("\nPlaintext :", plaintext)

print("Encrypted :", ciphertext)

print("Decrypted :", decrypted)

output:

