GLA University (Mathura)

Cryptography Lab Assignment

Submitted to -

Mr. Ashish tiwari
(Assistant professor)

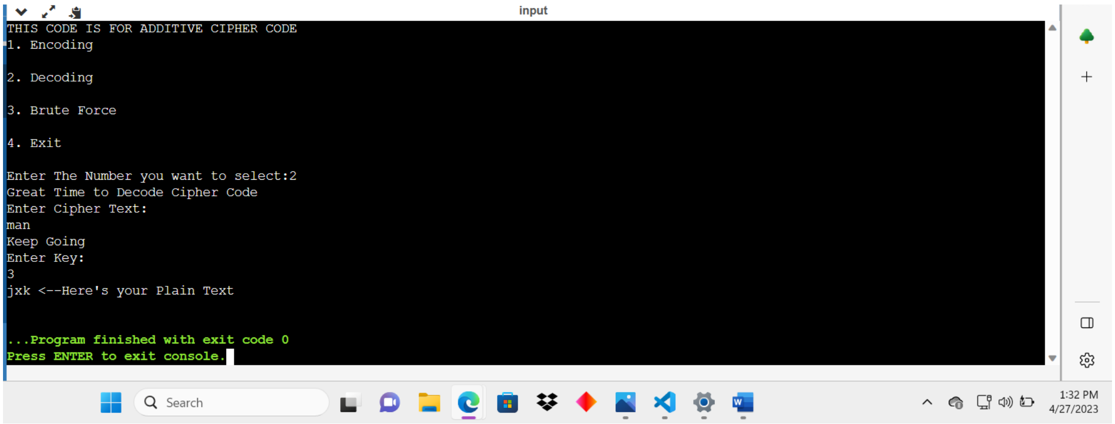
Submitted By -

Name - Manish sahu
Roll No - 41
Section - G
Uni.roll - 191500439

1. Write a program to implement Additive Cipher (Z26) with the following conditions: • Plaintext should be in lowercase. • Ciphertext should be uppercase. • Brute force attack.

```
import sys
opt = True
while opt:
    print("THIS CODE IS FOR ADDITIVE CIPHER CODE")
    print("1. Encoding\n")
    print("2. Decoding\n")
    print("3. Brute Force\n")
    print("4. Exit\n")
    break

selected_optio= int(input("Enter The Number you want to select:"))

if selected_optio== 1:
    while True:
        try:
            cor = input("Enter Plain Text: \n")
            if cor.isupper():
                print(cor, "is uppercase value plz Enter Lower Case Value")
            elif cor.isdigit():
                print(cor, "is an integer value please Enter Valid Input")
            else:
                print("Valid Input please Continue...")
                break;
        except ValueError:
            print("Provide Valid input")
    key = int(input("Enter Key Value: \n"))
    ans = ""
    for i in range(len(cor)):
        ch = cor[i]
        if ch==" ":
            ans+=" "
        elif (ch.isupper()):
            ans += chr((ord(ch) + key-65) % 26+65)
        else:
            ans += chr((ord(ch) + key - 97) % 26 + 97)
    print(ans,"<--Here's your Cipher Code")
elif selected_optio== 2:
    print("Great Time to Decode Cipher Code")
    while True:
        try:
            cor = input("Enter Cipher Text: \n")
            if cor.isupper():
                print(cor, "is uppercase value plz Enter Lower Case Value")
            elif cor.isdigit():
                print(cor, "is an integer value please Enter Valid Input")
            else:
                print("Keep Going")
                break;
        except ValueError:
            print("Provide Valid input")
    key = int(input("Enter Key: \n"))
    ans = ""
    for i in range(len(cor)):
        ch = cor[i]
        if ch==" ":
            ans+=" "
        elif (ch.isupper()):
            ans += chr((ord(ch) - key-65) % 26+65)
        else:
            ans += chr((ord(ch) - key - 97) % 26 + 97)
    print(ans,"<--Here's your Plain Text")
elif selected_optio== 3:
    ciphertext = input("Enter Cipher Text: \n")
    for key in range(1, 26):
        plaintext = ""
        for char in ciphertext:
            if char.isalpha():
                plain_char = chr((ord(char) - 65 + key) % 26 + 65)
                plaintext += plain_char
            else:
                plaintext += char
        print("Key: {} - Plaintext: {}".format(key, plaintext))
elif selected_optio== 4:
    print("Program Terminated...")
    sys.exit()
```

Output -

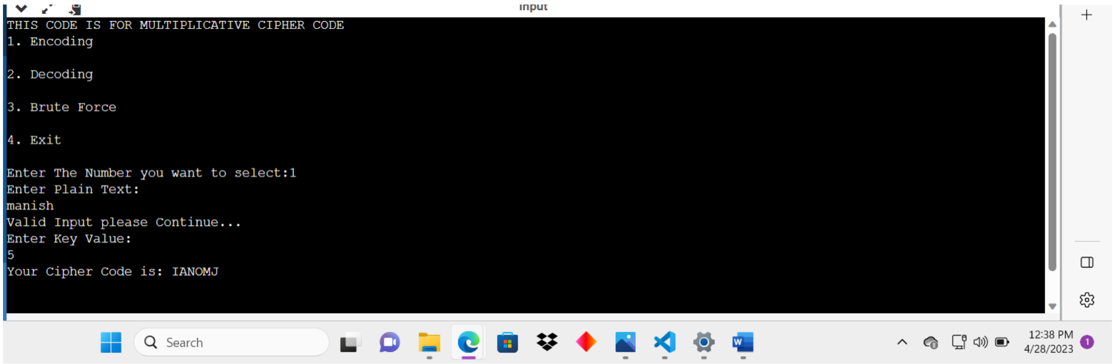


```
THIS CODE IS FOR ADDITIVE CIPHER CODE
1. Encoding

2. Decoding

3. Brute Force

4. Exit

Enter The Number you want to select:2
Great Time to Decode Cipher Code
Enter Cipher Text:
man
Keep Going
Enter Key:
3
jxk <--Here's your Plain Text


...Program finished with exit code 0
Press ENTER to exit console.
```

2. Write a program to implement Multiplicative Cipher. • Plaintext should be in lowercase. • Ciphertext should be uppercase. • Brute force attack.

```
import sys
opt = True
while opt:
   print("THIS CODE IS FOR MULTIPLICATIVE CIPHER CODE")
   print("1. Encoding\n")
   print("2. Decoding\n")
   print("3. Brute Force\n")
   print("4. Exit\n")
   break
choice = int(input("Enter The Number you want to select:"))
if choice == 1:
   while True:
      try:
         cor = input("Enter Plain Text: \n")
         if cor.isupper():
            print(cor, "is uppercase value plz Enter Lower Case Value")
         elif cor.isdigit():
            print(cor, "is an integer value please Enter Valid Input")
         else:
            print("Valid Input please Continue...")
            break;
      except ValueError:
         print("Provide Valid input")
   key = int(input("Enter Key Value:\n"))
   ciphertext = ""
   for char in cor:
      if char.isalpha():
         shift = ord(char.upper()) - 65
         shift = (shift * key) % 26
         ciphertext += chr(shift + 65)
      else:
         ciphertext += char
   print("Your Cipher Code is: " + ciphertext)
elif choice == 2:
   print("Great Time to Decode Cipher Code")
   while True:
      try:
            cor = input("Enter Cipher Text: \n")
            if cor.isupper():
                print(cor, "is uppercase value plz Enter Lower Case Value")
            elif cor.isdigit():
                print(cor, "is an integer value please Enter Valid Input")
            else:
                print("Keep Going")
                break;
        except ValueError:
            print("Provide Valid input")
    key = int(input("Enter Key Value in Integer only: "))
    plain_text = ""
    for char in cor:
        if char.isupper():
            plain_text += chr((ord(char) - key - 65) % 26 + 65)
        else:
            plain_text += chr((ord(char) - key - 97) % 26 + 97)
    print(plain_text)
elif choice == 3:
    ciphertext = input("Enter Cipher Text: \n")
    for key in range(1, 26):
        plaintext = ""
        for char in ciphertext:
            if char.isalpha():
                plain_char = chr((ord(char) - 65) * key % 26 + 65)
                plaintext += plain_char
            else:
                plaintext += char
        print("Key: {} - Plaintext: {}".format(key, plaintext))
elif choice == 4:
    print("Program Terminated...")
    sys.exit()
```

Output -

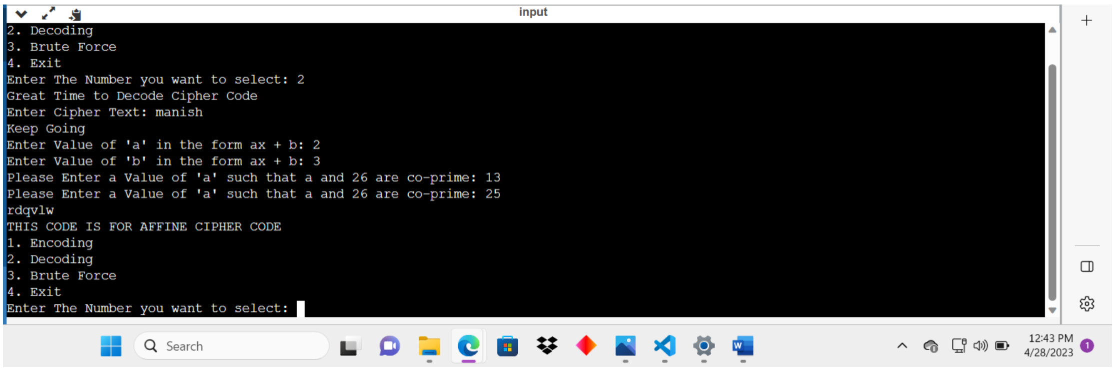


```
THIS CODE IS FOR MULTIPLICATIVE CIPHER CODE
1. Encoding

2. Decoding

3. Brute Force

4. Exit

Enter The Number you want to select:1
Enter Plain Text:
manish
Valid Input please Continue...
Enter Key Value:
5
Your Cipher Code is: IANOMJ
```

3.Write a program to implement Affine Cipher. • Plaintext should be in lowercase. • Ciphertext should be uppercase. • Brute force attack

```
import sys
import math
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
def mod_inverse(a, m):
    for x in range(1, m):
        if (a*x) % m == 1:
            return x
    return -1
opt = True
while opt:
    print("THIS CODE IS FOR AFFINE CIPHER CODE")
    print("1. Encoding")
    print("2. Decoding")
    print("3. Brute Force")
    print("4. Exit")
    choice = int(input("Enter The Number you want to select: ")
    if choice == 1:
        while True:
            try:
                plain_text = input("Enter Plain Text: ")
                if plain_text.isupper():
                    print(plain_text, "is uppercase value please Enter Lower Case Value")
                elif plain_text.isdigit():
                    print(plain_text, "is an integer value please Enter Valid Input")
                else:
                    print("Valid Input please Continue...")
                    break;
            except ValueError:
                print("Provide Valid input")
        a = int(input("Enter Value of 'a' in the form ax + b: "))
        b = int(input("Enter Value of 'b' in the form ax + b: "))
        while gcd(a, 26) != 1:
            a = int(input("Please Enter a Value of 'a' such that a and 26 are co-prime: "))
        ciphertext = ""
        for char in plain_text:
            if char.isalpha():
                shift = ord(char.lower()) - 97
                shift = ((a*shift) + b) % 26
                ciphertext += chr(shift + 97)
            else:
                ciphertext += char
        print("Your Cipher Code is: " + ciphertext
    elif choice == 2:
        print("Great Time to Decode Cipher Code")
        while True:
            try:
                ciphertext = input("Enter Cipher Text: ")
                if ciphertext.isupper():
                    print(ciphertext, "is uppercase value please Enter Lower Case Value")
                elif ciphertext.isdigit():
                    print(ciphertext, "is an integer value please Enter Valid Input")
                else:
                    print("Keep Going")
                    break;
            except ValueError:
                print("Provide Valid input")
        a = int(input("Enter Value of 'a' in the form ax + b: "))
        b = int(input("Enter Value of 'b' in the form ax + b: "))
        while gcd(a, 26) != 1:
            a = int(input("Please Enter a Value of 'a' such that a and 26 are co-prime: "))
        a_inverse = mod_inverse(a, 26)
        plain_text = ""
        for char in ciphertext:
            if char.isalpha():
                shift = ord(char.lower()) - 97
                shift = (a_inverse * (shift - b)) % 26
                plain_text += chr(shift + 97)
            else:
                plain_text += char
        print(plain_text)
    elif choice == 3:
        ciphertext = input("Enter Cipher Text: ")
        for a in range(1, 26):
            if gcd(a, 26) == 1:
                for b in range(26):
                    plain_text = ""
                    for char in ciphertext:
                        if char.isalpha():
                            shift = ord(char.lower()) - 97
                            shift = (a_inverse * (shift - b)) % 26
                            plain_text += chr(shift + 97)
                        else:
                            plain_text += char
```

Output -

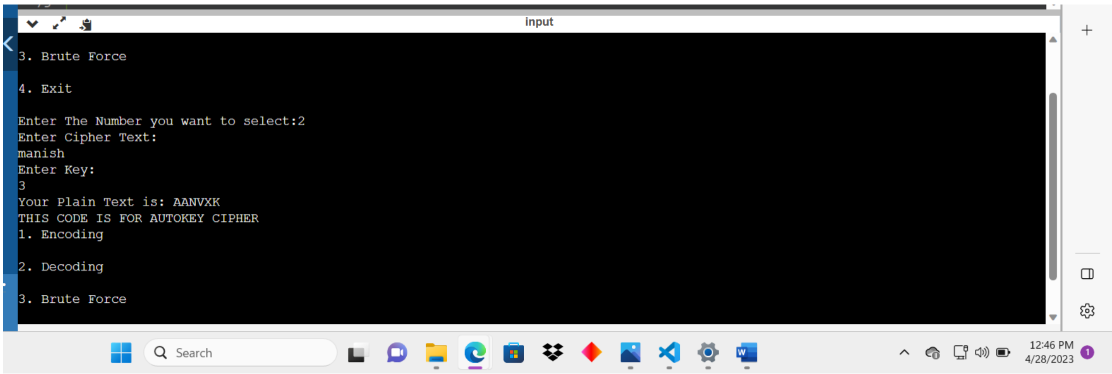


```
2. Decoding
3. Brute Force
4. Exit
Enter The Number you want to select: 2
Great Time to Decode Cipher Code
Enter Cipher Text: manish
Keep Going
Enter Value of 'a' in the form ax + b: 2
Enter Value of 'b' in the form ax + b: 3
Please Enter a Value of 'a' such that a and 26 are co-prime: 13
Please Enter a Value of 'a' such that a and 26 are co-prime: 25
rdqvlw
THIS CODE IS FOR AFFINE CIPHER CODE
1. Encoding
2. Decoding
3. Brute Force
4. Exit
Enter The Number you want to select: 
```

4.Write a program in to implement Autokey Cipher. • Plaintext should be in lowercase. • Ciphertext should be uppercase. • Brute force attack.

```
import sys
def encode_autokey(plaintext, key):
    """
    Encodes plaintext using Autokey Cipher with given key
    """
    key = key.upper()
    ciphertext = ""
    for i, char in enumerate(plaintext):
        if char.isalpha():
            shift = ord(key[i % len(key)].upper()) - 65
            shift = (shift + ord(char.upper()) - 65) % 26
            ciphertext += chr(shift + 65)
            key += char.upper()
        else:
            ciphertext += char
    return ciphertext

def decode_autokey(ciphertext, key):
    """
    Decodes ciphertext encoded using Autokey Cipher with given key
    """
    key = key.upper()
    plaintext = ""
    for i, char in enumerate(ciphertext):
        if char.isalpha():
            shift = ord(key[i % len(key)].upper()) - 65
            shift = (ord(char.upper()) - 65 - shift) % 26
            plaintext += chr(shift + 65)
            key += chr(shift + 65)
        else:
            plaintext += char
    return plaintext
def brute_force_autokey(ciphertext)
    for key_len in range(1, len(ciphertext)):
        for start in range(len(ciphertext) - key_len):
            possible_key = ciphertext[start : start + key_len]
            possible_plaintext = decode_autokey(ciphertext, possible_key)
            print("Key: {} - Plaintext: {}".format(possible_key,
possible_plaintext))
opt = True
while opt:
    print("THIS CODE IS FOR AUTOKEY CIPHER")
    print("1. Encoding\n")
    print("2. Decoding\n")
    print("3. Brute Force\n")
    print("4. Exit\n")
    choice = int(input("Enter The Number you want to select:"))
    if choice == 1:
        plaintext = input("Enter Plain Text: \n").upper()
        key = input("Enter Key: \n")
        ciphertext = encode_autokey(plaintext, key)
        print("Your Cipher Code is: " + ciphertext)
    elif choice == 2:
        ciphertext = input("Enter Cipher Text: \n").upper()
        key = input("Enter Key: \n")
        plaintext = decode_autokey(ciphertext, key)
        print("Your Plain Text is: " + plaintext)
    elif choice == 3:
        ciphertext = input("Enter Cipher Text: \n").upper()
        brute_force_autokey(ciphertext)
    elif choice == 4:
        print("Program Terminated...")
        sys.exit()
    else:
        print("Invalid choice. Please select a number from 1 to 4.")
```

Output -

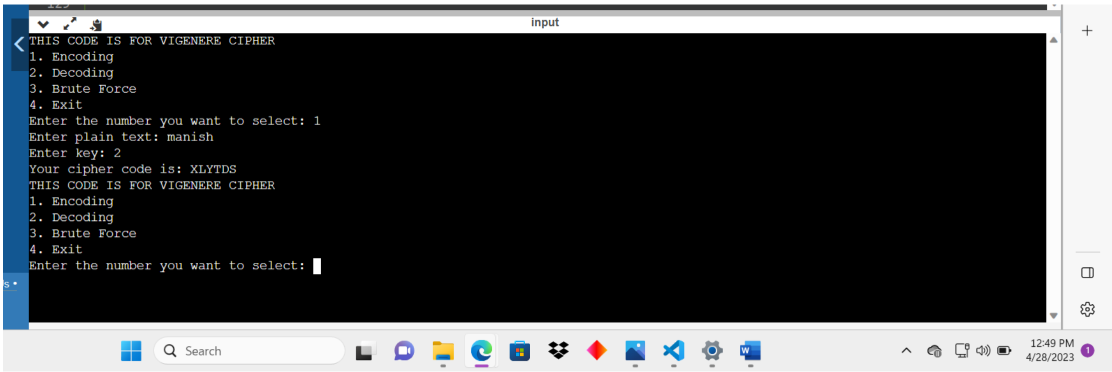


```
3. Brute Force

4. Exit

Enter The Number you want to select:2
Enter Cipher Text:
manish
Enter Key:
3
Your Plain Text is: AANVXK
THIS CODE IS FOR AUTOKEY CIPHER
1. Encoding

2. Decoding

3. Brute Force
```

5. Write a program to implement Vignere Cipher • Plaintext should be in lowercase. • Ciphertext should be uppercase. • Brute force attack

```
import sys

def vigenere_encrypt(plain_text, key):
    plain_text = plain_text.upper()
    key = key.upper()
    encrypted_text = ""
    for i in range(len(plain_text)):
        if plain_text[i].isalpha():
            shift = ord(key[i % len(key)]) - 65
            shifted_char = chr(((ord(plain_text[i]) - 65) + shift) % 26 + 65)
            encrypted_text += shifted_char
        else:
            encrypted_text += plain_text[i]
    return encrypted_text

def vigenere_decrypt(cipher_text, key):
    cipher_text = cipher_text.upper()
    key = key.upper()
    decrypted_text = ""
    for i in range(len(cipher_text)):
        if cipher_text[i].isalpha():
            shift = ord(key[i % len(key)]) - 65
            shifted_char = chr(((ord(cipher_text[i]) - 65) - shift) % 26 + 65)
            decrypted_text += shifted_char
        else:
            decrypted_text += cipher_text[i]
    return decrypted_text

def vigenere_brute_force(cipher_text):
    cipher_text = cipher_text.upper()
    for key_length in range(1, len(cipher_text) + 1):
        print(f"Brute forcing with key length: {key_length}")
        for i in range(26 ** key_length):
            key = ""
            for j in range(key_length):
                key += chr((i // (26 ** j)) % 26 + 65)
            decrypted_text = vigenere_decrypt(cipher_text, key)
            print(f"Key: {key} - Decrypted text: {decrypted_text}")

opt = True
while opt:
    print("THIS CODE IS FOR VIGENERE CIPHER")
    print("1. Encoding")
    print("2. Decoding")
    print("3. Brute Force")
    print("4. Exit")
    choice = int(input("Enter the number you want to select: "))

    if choice == 1:
        plain_text = input("Enter plain text: ")
        key = input("Enter key: ")
        cipher_text = vigenere_encrypt(plain_text, key)
        print(f"Your cipher code is: {cipher_text}")
    elif choice == 2:
        cipher_text = input("Enter cipher text: ")
        key = input("Enter key: ")
        plain_text = vigenere_decrypt(cipher_text, key)
        print(f"Your plain text is: {plain_text}")
    elif choice == 3:
        cipher_text = input("Enter cipher text: ")
        vigenere_brute_force(cipher_text)
    elif choice == 4:
        print("Program terminated...")
        sys.exit()
import sys

def vigenere_encrypt(plain_text, key):
    plain_text = plain_text.upper()
    key = key.upper()
    encrypted_text = ""
    for i in range(len(plain_text)):
        if plain_text[i].isalpha():
            shift = ord(key[i % len(key)]) - 65
            shifted_char = chr(((ord(plain_text[i]) - 65) + shift) % 26 + 65)
            encrypted_text += shifted_char
        else:
            encrypted_text += plain_text[i]
    return encrypted_text
def vigenere_decrypt(cipher_text, key):
    cipher_text = cipher_text.upper()
    key = key.upper()
    decrypted_text = ""
    for i in range(len(cipher_text)):
        if cipher_text[i].isalpha():
            shift = ord(key[i % len(key)]) - 65
            shifted_char = chr(((ord(cipher_text[i]) - 65) - shift) % 26 + 65)
            decrypted_text += shifted_char
        else:
            decrypted_text += cipher_text[i]
    return decrypted_text
def vigenere_brute_force(cipher_text):
    cipher_text = cipher_text.upper()
    for key_length in range(1, len(cipher_text) + 1):
        print(f"Brute forcing with key length: {key_length}")
        for i in range(26 ** key_length):
            key = ""
            for j in range(key_length):
                key += chr((i // (26 ** j)) % 26 + 65)
            decrypted_text = vigenere_decrypt(cipher_text, key)
            print(f"Key: {key} - Decrypted text: {decrypted_text}")
opt = True
while opt:
    print("THIS CODE IS FOR VIGENERE CIPHER")
    print("1. Encoding")
    print("2. Decoding")
    print("3. Brute Force")
    print("4. Exit")
    choice = int(input("Enter the number you want to select: "))
    if choice == 1:
        plain_text = input("Enter plain text: ")
        key = input("Enter key: ")
        cipher_text = vigenere_encrypt(plain_text, key)
        print(f"Your cipher code is: {cipher_text}")
    elif choice == 2:
        cipher_text = input("Enter cipher text: ")
        key = input("Enter key: ")
        plain_text = vigenere_decrypt(cipher_text, key)
        print(f"Your plain text is: {plain_text}")
    elif choice == 3:
        cipher_text = input("Enter cipher text: ")
        vigenere_brute_force(cipher_text)
    elif choice == 4:
        print("Program terminated...")
        sys.exit()
```

Output -

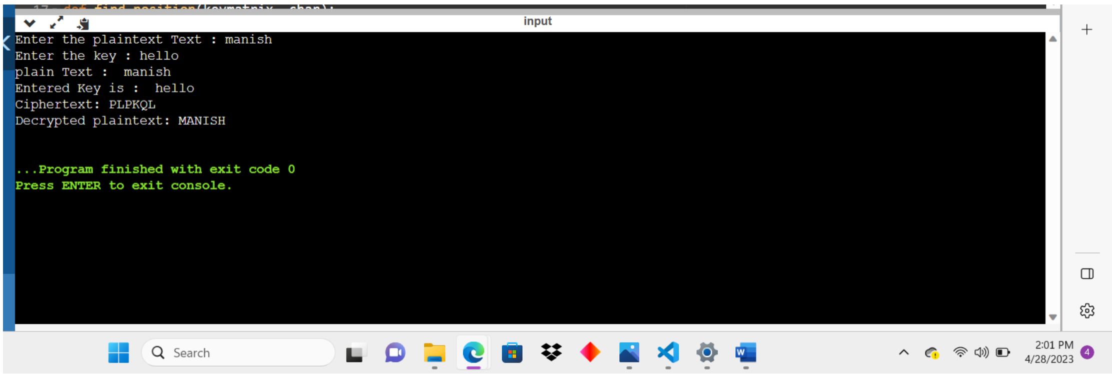


```
THIS CODE IS FOR VIGENERE CIPHER
1. Encoding
2. Decoding
3. Brute Force
4. Exit
Enter the number you want to select: 1
Enter plain text: manish
Enter key: 2
Your cipher code is: XLYTDS
THIS CODE IS FOR VIGENERE CIPHER
1. Encoding
2. Decoding
3. Brute Force
4. Exit
Enter the number you want to select: 
```

6. Write a program to implement Playfair Cipher to encrypt & decrypt the given message where the key matrix can be formed by using a given keyword

```
import re

def generate_keymatrix(key):
    # create key matrix from given keyword
    key = key.replace(' ', '').upper()
    key = re.sub(r'J', 'I', key)
    keymatrix = []
    for char in key:
        if char not in keymatrix:
            keymatrix.append(char)
    alphabet = 'ABCDEFGHIKLMNOPQRSTUVWXYZ'
    for char in alphabet:
        if char not in keymatrix:
            keymatrix.append(char)
    keymatrix = [keymatrix[i:i+5] for i in range(0, 25, 5)]
    return keymatrix

def find_position(keymatrix, char):
    # find position of character in key matrix
    row = col = 0
    for i in range(5):
        for j in range(5):
            if keymatrix[i][j] == char:
                row, col = i, j
                break
    return row, col
def encrypt(plaintext, key):
    # encrypt plaintext using Playfair Cipher
    keymatrix = generate_keymatrix(key)
    plaintext = plaintext.replace(' ', '').upper()
    plaintext = re.sub(r'J', 'I', plaintext)
    if len(plaintext) % 2 == 1:
        plaintext += 'X'
    ciphertext = ''
    for i in range(0, len(plaintext), 2):
        char1 = plaintext[i]
        char2 = plaintext[i+1]
        row1, col1 = find_position(keymatrix, char1)
        row2, col2 = find_position(keymatrix, char2)
        if row1 == row2:
            ciphertext += keymatrix[row1][(col1+1)%5]
            ciphertext += keymatrix[row2][(col2+1)%5]
        elif col1 == col2:
            ciphertext += keymatrix[(row1+1)%5][col1]
            ciphertext += keymatrix[(row2+1)%5][col2]
        else:
            ciphertext += keymatrix[row1][col2]
            ciphertext += keymatrix[row2][col1]
    return ciphertext
def decrypt(ciphertext, key):
    # decrypt ciphertext using Playfair Cipher
    keymatrix = generate_keymatrix(key)
    plaintext = ''
    for i in range(0, len(ciphertext), 2):
        char1 = ciphertext[i]
        char2 = ciphertext[i+1]
        row1, col1 = find_position(keymatrix, char1)
        row2, col2 = find_position(keymatrix, char2)
        if row1 == row2:
            plaintext += keymatrix[row1][(col1-1)%5]
            plaintext += keymatrix[row2][(col2-1)%5]
        elif col1 == col2:
            plaintext += keymatrix[(row1-1)%5][col1]
            plaintext += keymatrix[(row2-1)%5][col2]
        else:
            plaintext += keymatrix[row1][col2]
            plaintext += keymatrix[row2][col1]
    plaintext = re.sub(r'X$', '', plaintext)
    return plaintext
plaintext =input("Enter the plaintext Text : ")
key = input("Enter the key : ")
print("plain Text : ",plaintext)
print("Entered Key is : ",key)
ciphertext = encrypt(plaintext, key)

print('Ciphertext:', ciphertext)
decrypted_plaintext = decrypt(ciphertext, key)
print('Decrypted plaintext:', decrypted_plaintext)
```

Output -

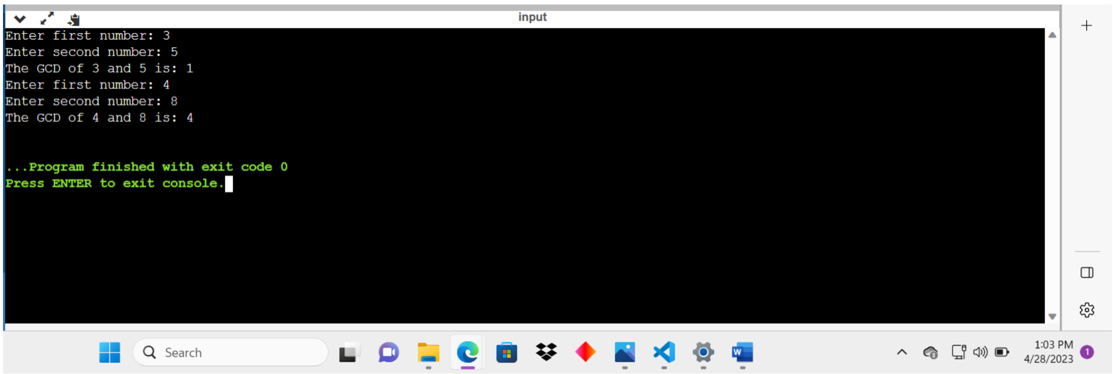


```
Enter the plaintext Text : manish
Enter the key : hello
plain Text :  manish
Entered Key is :  hello
Ciphertext: PLPKQL
Decrypted plaintext: MANISH


...Program finished with exit code 0
Press ENTER to exit console.
```

7.W.A.P. to implement Euclidean Algorithm to find the GCD of given numbers

```
def gcd(a, b):
    if a == 0:
        return b
    else:
        return gcd(b % a, a)

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
gcd_value = gcd(num1, num2)
print("The GCD of", num1, "and", num2, "is:", gcd_value)
def gcd(a, b):
    if a == 0:
        return b
    else:
        return gcd(b % a, a)

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

gcd_value = gcd(num1, num2)

print("The GCD of", num1, "and", num2, "is:", gcd_value)
```

Output -

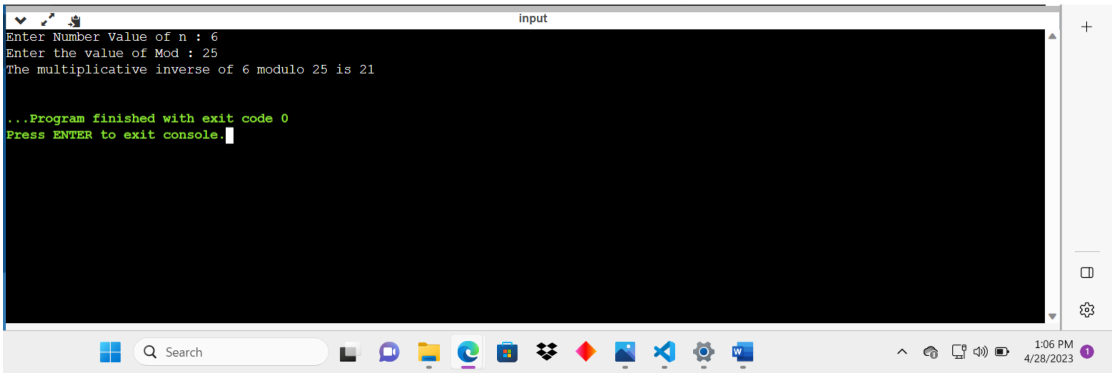


```
Enter first number: 3
Enter second number: 5
The GCD of 3 and 5 is: 1
Enter first number: 4
Enter second number: 8
The GCD of 4 and 8 is: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

8. Write a program to find out the Multiplicative inverse of a given number without using extended Euclidean algorithm

```
def multiplicative_inverse(n, mod):

    for i in range(mod):
        if (i * n) % mod == 1:
            return i
    return None

# Example usage
n = int(input("Enter Number Value of n : "))
mod = int(input("Enter the value of Mod : "))
inverse = multiplicative_inverse(n, mod)
if inverse is not None:
    print(f"The multiplicative inverse of {n} modulo {mod} is
{inverse}")
else:
    print(f"{n} has no inverse modulo {mod}")

import random
import math

def is_prime(num):
    """
    Checks whether the given number is prime or not.
    """
    if num == 2:
        return True
    if num % 2 == 0 or num == 1:
        return False
    for i in range(3, int(math.sqrt(num)) + 1, 2):
        if num % i == 0:
            return False
    return True

def find_primitive_root(p):
    """
    Finds a primitive root for a prime number p.
    """
    if not is_prime(p):
        return None
    factors = []
    phi = p - 1
    n = phi
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            factors.append(i)
            while n % i == 0:
                n /= i
    if n > 1:
        factors.append(n)
    for g in range(2, p):
        flag = True
        for factor in factors:
            if pow(g, phi // factor, p) == 1:
                flag = False
                break
        if flag:
            return g
    return None
```

Output -

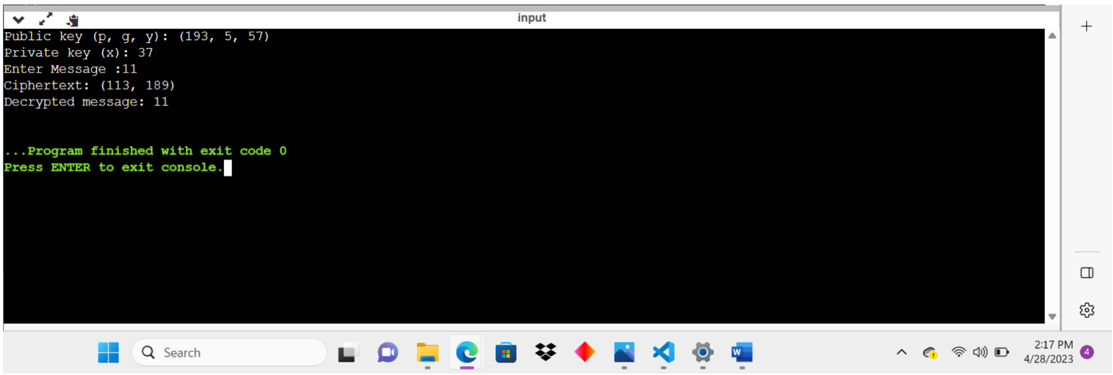


```
Enter Number Value of n : 6
Enter the value of Mod : 25
The multiplicative inverse of 6 modulo 25 is 21


...Program finished with exit code 0
Press ENTER to exit console.
```

9.Write a program to implement Elgamal Cryptosystem to generate the pair of keys and then show the encryption & decryption of a given message

```
import math
import random
def is_prime(num):
    """
    Checks whether the given number is prime or not.
    """
    if num == 2:
        return True
    if num % 2 == 0 or num == 1:
        return False
    for i in range(3, int(math.sqrt(num)) + 1, 2):
        if num % i == 0:
            return False
    return True

def find_primitive_root(p):
    """
    Finds a primitive root for a prime number p.
    """
    if not is_prime(p):
        return None
    factors = []
    phi = p - 1
    n = phi
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            factors.append(i)
            while n % i == 0:
                n /= i
    if n > 1:
        factors.append(n)
    for g in range(2, p):
        flag = True
        for factor in factors:
            if pow(g, phi // factor, p) == 1:
                flag = False
                break
        if flag:
            return g
    return None
def generate_keypair():
    """
    Generates a keypair for ElGamal cryptosystem.
    """
    p = random.randint(100, 1000)
    while not is_prime(p):
        p = random.randint(100, 1000)
    g = find_primitive_root(p)
    x = random.randint(2, p - 2)
    y = pow(g, x, p)
    return (p, g, y, x)

def encrypt_message(message, p, g, y):
    """
    Encrypts the message using ElGamal cryptosystem.
    """
    k = random.randint(2, p - 2)
    a = pow(g, k, p)
    b = (pow(y, k, p) * message) % p
    return (a, b)

def decrypt_message(ciphertext, p, x):
    """
    Decrypts the ciphertext using ElGamal cryptosystem.
    """
    a, b = ciphertext
    message = (b * pow(a, p - 1 - x, p)) % p
    return message

# Generate a keypair
p, g, y, x = generate_keypair()
print("Public key (p, g, y):", (p, g, y))
print("Private key (x):", x)

# Encrypt a message
message = int(input("Enter Message :"))
ciphertext = encrypt_message(message, p, g, y)
print("Ciphertext:", ciphertext)

# Decrypt the ciphertext
decrypted_message = decrypt_message(ciphertext, p, x)
print("Decrypted message:", decrypted_message)
```

Output -



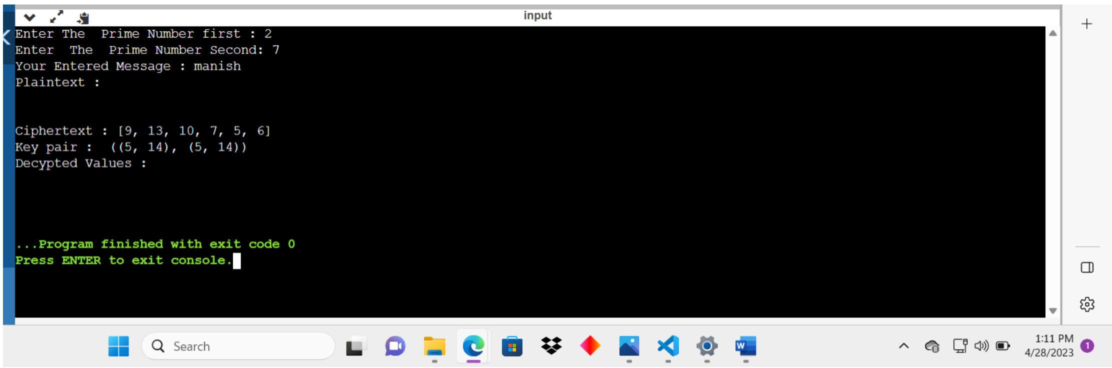
```
Public key (p, g, y): (193, 5, 57)
Private key (x): 37
Enter Message :11
Ciphertext: (113, 189)
Decrypted message: 11


...Program finished with exit code 0
Press ENTER to exit console.
```

10. W.A.P. to implement RSA Algorithm to generate a pair of keys and show the encryption and decryption by using a given key pair

```
import random
import math

while True:
   p = int(input("Enter The  Prime Number first : "))
   q = int(input("Enter  The  Prime Number Second: "))
   if(p!=q):
      break
   else:
      print("Values should be differentg")


message = input("Your Entered Message : ")


def generate_keypair(p, q):
   n = p * q
   phi = (p - 1) * (q - 1)
   # Choose e such that e and phi(n) are coprime
   e = random.randrange(1, phi)
   g = math.gcd(e, phi)
   while g != 1:
      e = random.randrange(1, phi)
      g = math.gcd(e, phi)
   # Calculate the private key
   d = pow(e, -1, phi)
  # print((e, n), (d, n))
   return ((e, n), (d, n))


public_key, private_key = generate_keypair(p, q)
```

```
def encrypt(public_key, plaintext):
    e, n = public_key
    ciphertext = [pow(ord(char), e, n) for char in plaintext]
    return ciphertext

def decrypt(private_key, ciphertext):
    d, n = private_key
    plaintext = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(plaintext)

ciphertext = encrypt(public_key, message)
plaintext = decrypt(private_key, ciphertext)

print("Plaintext :", plaintext)
print("Ciphertext :", ciphertext)
print("Key pair : ",generate_keypair(p,q))
print("Decypted Values : ",decrypt(private_key,ciphertext))
```

Output -

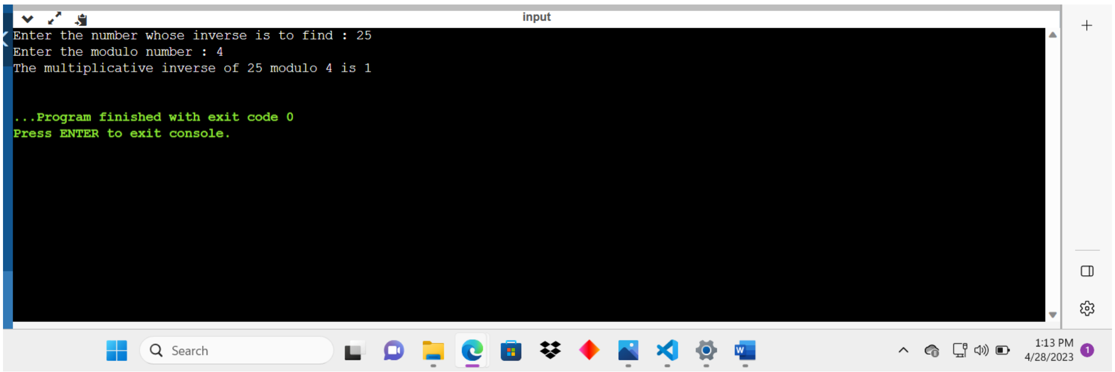


```
Enter The  Prime Number first : 2
Enter  The  Prime Number Second: 7
Your Entered Message : manish
Plaintext :


Ciphertext : [9, 13, 10, 7, 5, 6]
Key pair :  ((5, 14), (5, 14))
Decypted Values :



...Program finished with exit code 0
Press ENTER to exit console.
```

11. Write a program to find out the Multiplicative inverse of a given number by using Extended Euclidean algorithm

```
def extended_euclidean_algorithm(a, b):

    if b == 0:
        return (a, 1, 0)
    else:
        q = a // b
        r = a % b
        (gcd, s, t) = extended_euclidean_algorithm(b, r)
        return (gcd, t, s - q * t)
def multiplicative_inverse(a, m):
    (gcd, s, t) = extended_euclidean_algorithm(a, m)
    if gcd != 1:
        return None
    else:
        return s % m
a = int(input("Enter the number whose inverse is to find : "))
m = int(input("Enter the modulo number : "))
inverse = multiplicative_inverse(a, m)
if inverse is None:
    print(f"{a} has no multiplicative inverse modulo {m}")
else:
    print(f"The multiplicative inverse of {a} modulo {m} is
{inverse}")
```

Output -



```
Enter the number whose inverse is to find : 25
Enter the modulo number : 4
The multiplicative inverse of 25 modulo 4 is 1


...Program finished with exit code 0
Press ENTER to exit console.
```