1. def caesar\_cipher(message, shift):

cipher = ''

for char in message:

if char.isalpha():

char\_code = ord(char) - shift

if char.isupper():

if char\_code > ord('Z'):

char\_code -= 26

elif char\_code < ord('A'):

char\_code += 26

elif char.islower():

if char\_code > ord('z'):

char\_code -= 26

elif char\_code < ord('a'):

char\_code += 26

cipher += chr(char\_code)

else:

cipher += char

return cipher

message = input("Enter the string to be decrypted")

shift = 3

decrypted\_message = caesar\_cipher(message, shift)

print(decrypted\_message)

1. import string

# Define the mapping for the cipher

cipher\_map = {'a': 'q', 'b': 'w', 'c': 'e', 'd': 'r', 'e': 't',

'f': 'y', 'g': 'u', 'h': 'i', 'i': 'o', 'j': 'p',

'k': 'a', 'l': 's', 'm': 'd', 'n': 'f', 'o': 'g',

'p': 'h', 'q': 'j', 'r': 'k', 's': 'l', 't': 'z',

'u': 'x', 'v': 'c', 'w': 'v', 'x': 'b', 'y': 'n', 'z': 'm'}

# Define the reverse mapping for decryption

decipher\_map = {v: k for k, v in cipher\_map.items()}

def encrypt(message):

"""Encrypts the given message using the cipher map."""

# Convert message to lowercase

message = message.lower()

# Initialize the encrypted message

encrypted\_message = ''

# Encrypt each character in the message

for char in message:

if char in string.ascii\_lowercase:

encrypted\_char = cipher\_map[char]

else:

encrypted\_char = char

encrypted\_message += encrypted\_char

return encrypted\_message

message = input("Enter the text:")

encrypted\_message = encrypt(message)

print(encrypted\_message)

3.Play fair cipher

def toLowerCase(text):

return text.lower()

def removeSpaces(text):

newText = ""

for i in text:

if i == " ":

continue

else:

newText = newText + i

return newText

def Diagraph(text):

Diagraph = []

group = 0

for i in range(2, len(text), 2):

Diagraph.append(text[group:i])

group = i

Diagraph.append(text[group:])

return Diagraph

def FillerLetter(text):

k = len(text)

if k % 2 == 0:

for i in range(0, k, 2):

if text[i] == text[i+1]:

new\_word = text[0:i+1] + str('x') + text[i+1:]

new\_word = FillerLetter(new\_word)

break

else:

new\_word = text

else:

for i in range(0, k-1, 2):

if text[i] == text[i+1]:

new\_word = text[0:i+1] + str('x') + text[i+1:]

new\_word = FillerLetter(new\_word)

break

else:

new\_word = text

return new\_word

list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',

'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

def generateKeyTable(word, list1):

key\_letters = []

for i in word:

if i not in key\_letters:

key\_letters.append(i)

compElements = []

for i in key\_letters:

if i not in compElements:

compElements.append(i)

for i in list1:

if i not in compElements:

compElements.append(i)

matrix = []

while compElements != []:

matrix.append(compElements[:5])

compElements = compElements[5:]

return matrix

def search(mat, element):

for i in range(5):

for j in range(5):

if(mat[i][j] == element):

return i, j

def encrypt\_RowRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

if e1c == 4:

char1 = matr[e1r][0]

else:

char1 = matr[e1r][e1c+1]

char2 = ''

if e2c == 4:

char2 = matr[e2r][0]

else:

char2 = matr[e2r][e2c+1]

return char1, char2

def encrypt\_ColumnRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

if e1r == 4:

char1 = matr[0][e1c]

else:

char1 = matr[e1r+1][e1c]

char2 = ''

if e2r == 4:

char2 = matr[0][e2c]

else:

char2 = matr[e2r+1][e2c]

return char1, char2

def encrypt\_RectangleRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

char1 = matr[e1r][e2c]

char2 = ''

char2 = matr[e2r][e1c]

return char1, char2

def encryptByPlayfairCipher(Matrix, plainList):

CipherText = []

for i in range(0, len(plainList)):

c1 = 0

c2 = 0

ele1\_x, ele1\_y = search(Matrix, plainList[i][0])

ele2\_x, ele2\_y = search(Matrix, plainList[i][1])

if ele1\_x == ele2\_x:

c1, c2 = encrypt\_RowRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

# Get 2 letter cipherText

elif ele1\_y == ele2\_y:

c1, c2 = encrypt\_ColumnRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

else:

c1, c2 = encrypt\_RectangleRule(

Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

cipher = c1 + c2

CipherText.append(cipher)

return CipherText

text\_Plain = 'instruments'

text\_Plain = removeSpaces(toLowerCase(text\_Plain))

PlainTextList = Diagraph(FillerLetter(text\_Plain))

if len(PlainTextList[-1]) != 2:

PlainTextList[-1] = PlainTextList[-1]+'z'

key = "Monarchy"

print("Key text:", key)

key = toLowerCase(key)

Matrix = generateKeyTable(key, list1)

print("Plain Text:", text\_Plain)

CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)

CipherText = ""

for i in CipherList:

CipherText += i

print("CipherText:", CipherText)

4. def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return(key)

else:

for i in range(len(string) -

len(key)):

key.append(key[i % len(key)])

return("" . join(key))

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) +

ord(key[i])) % 26

x += ord('A')

cipher\_text.append(chr(x))

return("" . join(cipher\_text))

if \_name\_ == "\_main\_":

string = "GEEKSFORGEEKS"

keyword = "AYUSH"

key = generateKey(string, keyword)

cipher\_text = cipherText(string,key)

print("Ciphertext :", cipher\_text)

5. #include <stdio.h>

#include <string.h>

void affineCipher(char plain[], int key[])

{

int i, x;

char cipher[strlen(plain)];

for (i = 0; i < strlen(plain); i++) {

x = plain[i] - 'a';

x = (key[0] \* x + key[1]) % 26;

cipher[i] = x + 'a';

}

printf("Ciphertext: %s", cipher);

}

int main()

{

int key[] = { 17, 20 };

char plain[] = "twenty fifteen";

affineCipher(plain, key);

return 0;

}

6. import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class AffineCipherBreaker {

private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public static void main(String[] args) {

Scanner read = new Scanner(System.in);

String ciphertext = read.nextLine();

Map<Character, Integer> frequencyMap = new HashMap<>();

for (char c : ciphertext.toCharArray()) {

frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);

}

char mostFrequent = 'A';

char secondMostFrequent = 'A';

int highestFrequency = 0;

int secondHighestFrequency = 0;

for (char c : frequencyMap.keySet()) {

int frequency = frequencyMap.get(c);

if (frequency > highestFrequency) {

secondMostFrequent = mostFrequent;

secondHighestFrequency = highestFrequency;

mostFrequent = c;

highestFrequency = frequency;

} else if (frequency > secondHighestFrequency) {

secondMostFrequent = c;

secondHighestFrequency = frequency;

}

}

System.out.println("Most frequent letter: " + mostFrequent);

System.out.println("Second most frequent letter: " + secondMostFrequent);

}

}

7. #include<stdio.h>

char monocipher\_encr(char);

char alpha[27][3] = { { 'a', 'c' }, { 'b', 'i' }, { 'c', 'p' }, { 'd', 'h' }, {

'e', 'e' }, { 'f', 'r' }, { 'g', 'a' }, { 'h', 'b' }, { 'i', 'd' }, {

'j', 'f' }, { 'k', 'g' }, { 'l', 'j' }, { 'm', 'k' }, { 'n', 'l' }, {

'o', 'm' }, { 'p', 'n' }, { 'q', 'o' }, { 'r', 'q' }, { 's', 's' }, {

't', 't' }, { 'u', 'u' }, { 'v', 'v' }, { 'w', 'w' }, { 'x', 'x' }, {

'y', 'y' }, { 'z', 'z' } };

char str[20];

int main() {

char str[20], str2[20];

int i;

printf("\n enter a plaintext..");

gets(str);

for (i = 0; str[i]; i++) {

str2[i] = monocipher\_encr(str[i]);

}

str2[i] = '\0';

printf("\n Before Decryption..%s", str);

printf("\n After Decryption..%s\n", str2);

}

char monocipher\_encr(char a) {

int i;

for (i = 0; i < 27; i++) {

if (a == alpha[i][0])

break;

}

return alpha[i][1];

}

9. import itertools

def find\_keyword():

alphabet = 'ABCDEFGHIKLMNOPQRSTUVWXYZ'

combinations = itertools.combinations(alphabet, 25)

for keyword in combinations:

matrix = [[0]\*5 for \_ in range(5)]

for i, letter in enumerate(keyword):

row = i // 5

col = i % 5

matrix[row][col] = letter

valid = True

for row in range(5):

for col in range(5):

if matrix[row][col] == 0:

valid = False

break

if matrix[row][col] == 'I' or matrix[row][col] == 'J':

matrix[row][col] = 'IJ'

if matrix[row][col] in matrix[row][col+1:] + [matrix[i][col] for i in range(row+1, 5)]:

valid = False

break

if not valid:

break

if valid:

return keyword

return None

keyword = find\_keyword()

if keyword is not None:

print(f"The keyword is {keyword}.")

print(f"Its approximate power of 2 is {2\*\*(len(keyword)\*5):,.0f}.")

else:

print("No valid keyword was found.")

10. # Define the key matrix

key = [[9, 4], [5, 7]]

# Define the plaintext message

plaintext = "meet me at the usual place at ten rather than eight oclock"

# Convert the plaintext to uppercase and remove spaces

plaintext = plaintext.upper().replace(" ", "")

# Pad the plaintext with "X" if necessary to make the length a multiple of 2

if len(plaintext) % 2 != 0:

plaintext += "X"

# Split the plaintext into pairs of 2 letters and convert each pair to a vector

plaintext\_vectors = []

for i in range(0, len(plaintext), 2):

pair = plaintext[i:i+2]

vector = [ord(pair[0])-65, ord(pair[1])-65]

plaintext\_vectors.append(vector)

# Multiply each plaintext vector by the key matrix to get the corresponding ciphertext vector

ciphertext\_vectors = []

for vector in plaintext\_vectors:

ciphertext\_vector = [(key[0][0]\*vector[0] + key[0][1]\*vector[1]) % 26, (key[1][0]\*vector[0] + key[1][1]\*vector[1]) % 26]

ciphertext\_vectors.append(ciphertext\_vector)

# Convert each ciphertext vector back to a pair of letters

ciphertext = ""

for vector in ciphertext\_vectors:

pair = chr(vector[0]+65) + chr(vector[1]+65)

ciphertext += pair

# Print the ciphertext

print(ciphertext)

11. import java.util.Random;

public class VigenereCipher {

public static void main(String[] args) {

String plaintext = "HELLO WORLD";

int[] key = generateKey(plaintext.length());

System.out.println("Plaintext: " + plaintext);

String ciphertext = encrypt(plaintext, key);

System.out.println("Ciphertext: " + ciphertext);

String decryptedText = decrypt(ciphertext, key);

System.out.println("Decrypted text: " + decryptedText);

}

public static int[] generateKey(int length) {

int[] key = new int[length];

Random random = new Random();

for (int i = 0; i < length; i++) {

key[i] = random.nextInt(26) + 1;

}

return key;

}

public static String encrypt(String plaintext, int[] key) {

String ciphertext = "";

int keyIndex = 0;

for (int i = 0; i < plaintext.length(); i++) {

char c = plaintext.charAt(i);

int shift = key[keyIndex];

char encryptedChar = shiftChar(c, shift);

ciphertext += encryptedChar;

keyIndex = (keyIndex + 1) % key.length;

}

return ciphertext;

}

public static String decrypt(String ciphertext, int[] key) {

String decryptedText = "";

int keyIndex = 0;

for (int i = 0; i < ciphertext.length(); i++) {

char c = ciphertext.charAt(i);

int shift = key[keyIndex];

char decryptedChar = shiftChar(c, -shift);

decryptedText += decryptedChar;

keyIndex = (keyIndex + 1) % key.length;

}

return decryptedText;

}

public static char shiftChar(char c, int shift) {

if (!Character.isLetter(c)) {

return c;

}

int base = Character.isLowerCase(c) ? 'a' : 'A';

int offset = c - base;

int shiftedOffset = (offset + shift + 26) % 26;

return (char) (base + shiftedOffset);

}

}