**Practical 1**

Write a program to perform encryption and decryption using Caesar cipher algorithm.

Encryption procedure: if the alphabet index is even then increase the value by one else decrease the key value by one.

Decryption Procedure: if the alphabet index is even then decrypting the value by one else increase the key value by one.

* **CODE :-**

key=1

s\_msg=input("Enter your message : ")

sm\_l=list(s\_msg)

def encrypt(sm\_l,key):

l1=[]

for k in sm\_l:

if (ord(k)==32):

l1.append(" ")

else:

x=0

m=int(ord(k))

if k.isupper():

x=65

else:

x=97

if m%2==0:

j=chr((m+key-x)%26+x)

else:

j=chr((m-key-x)%26+x)

l1.append(j)

return l1

def decrypt(l1,key):

l2=[]

for k in l1:

if(ord(k)==32):

l2.append(" ")

else:

x=0

m=int(ord(k))

if k.isupper():

x=65

else:

x=97

if m%2==0:

j=chr((m+key-x)%26+x)

else:

j=chr((m-key-x)%26+x)

l2.append(j)

return l2

en\_sml=encrypt(sm\_l,key)

print("Encrypted data is : ")

for x in en\_sml:

print(x,end="")

print()

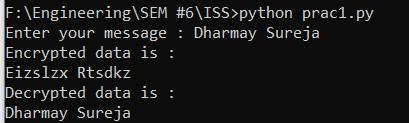
de\_sml=decrypt(en\_sml,key)

print("Decrypted data is : ")

for x in de\_sml:

print(x,end="")

* **OUTPUT :-**

****

**Practical 2**

Write a program to find plain text messages and key information corresponds to following cipher text messages using brute-force technique on Caesar cipher.

1.      PmttwEmtkwumBwCDXKM

2.      Qefpfpzxbpbozfmeboxidlofqej

3.      TrvjviTzgyvizjNvrbRcxfizkyd

4.      LbhNerFzneggbNggnpxPnrfrePvcure

* **CODE :-**

def en(s1):

#s1=input("Enter the cipher text: ")

l1=list(s1)

str1=[]

key=1

while key<27:

y=" "

for k in l1:

x=0

if k.isupper():

x=65

else:

x=97

y+=chr(((ord(k)-key-x)%26+x))

str1.append(y)

key+=1

print("Decrypted from ",s1)

print()

i=0

while i<26:

print(str1[i])

i+=1

s1="PmttwEmtkwumBwCDXKM"

en(s1)

print()

s1="Qefpfpzxbpbozfmeboxidlofqej"

en(s1)

print()

s1="TrvjviTzgyvizjNvrbRcxfizkyd"

en(s1)

print()

s1="LbhNerFzneggbNggnpxPnrfrePvcure"

en(s1)

* **OUTPUT :-**

"""1. key 8

HelloWelcomeToUVPCE

2. key 23

Thisiscaesercipheralgorithm

3. key 17

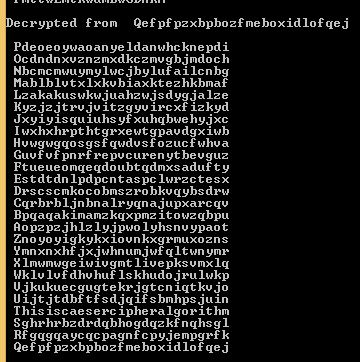
CaeserCipherisWeakAlgorithm

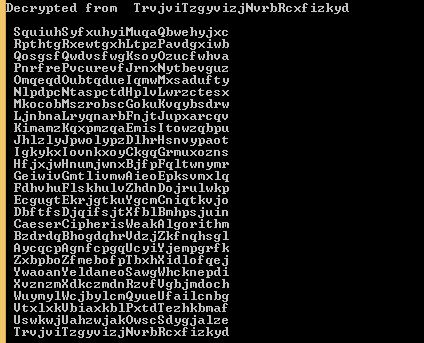
4. key 13

YouAreSmarttoAttackCaeserCipher

"""









**Practical 3**

**Write a program to perform encryption and decryption using Monoalphabetic Cipher Technique.**

* **CODE :-**

dicts = {"a":"z","b":"y","c":"x","d":"w","e":"v","f":"u","g":"t","h":"s","i":"r","j":"q","k":"p","l":"o","m":"n","n":"m","o":"l","p":"k","q":"j","r":"i","s":"h","t":"g","u":"f","v":"e","w":"d","x":"c","y":"b","z":"a",}

dictc = {"A":"Z","B":"Y","C":"X","D":"W","E":"V","F":"U","G":"T","H":"S","I":"R","J":"Q","K":"P","L":"O","M":"N","N":"M","O":"L","P":"K","Q":"J","R":"I","S":"H","T":"G","U":"F","V":"E","W":"D","X":"C","Y":"B","Z":"A",}

a=input("enter message ")

l1=list(a)

l2=[]

def get\_key(val):

if(val.isupper()):

for key, value in dictc.items():

if val == value:

return key

else:

for key, value in dicts.items():

if val == value:

return key

print(" \nencrypted text is")

for i in l1:

if(i.isupper()):

print(dictc[i],end="")

z=dictc[i]

l2.append(z)

elif(ord(i)==32):

l2.append(" ")

print(i,end="")

else:

print(dicts[i],end="")

z=dicts[i]

l2.append(z)

print("\nDecrypted text is ")

for i in l2:

no=ord(i)

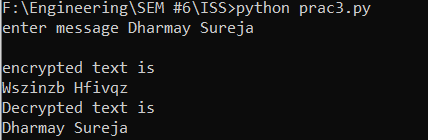
if(no==32):

print(" ",end="")

else:

print(get\_key(i),end="")

* **OUTPUT :-**

****

**Practical 4**

Write a program to perform encryption and decryption using Polyalphabetic Cipher (Vigenere Cipher) Technique.

* **CODE :-**

s\_msg=input("Enter your message : ")

sm\_l=list(s\_msg)

key=list(input("enter key : "))

ke1=[]

p=0

for i in sm\_l:

ke1.append(key[p])

p=(p+1)%len(key)

def encrypt(sm\_l,key):

l1=[]

o=0

for k in sm\_l:

x=0

ke=int(ord(key[o]))

m=int(ord(k))

o=o+1

if k.isupper():

x=65

else:

x=97

j=chr(((m-x)+(ke-x))%26+x)

l1.append(j)

return l1

def decrypt(l1,key):

l2=[]

o=0

for k in l1:

x=0

ke=int(ord(key[o]))

m=int(ord(k))

o=o+1

if k.isupper():

x=65

else:

x=97

j=chr(((m-x)-(ke-x) +26)%26+x)

l2.append(j)

return l2

en\_sml=encrypt(sm\_l,ke1)

print("Encrypted data is : ")

for x in en\_sml:

print(x,end="")

print()

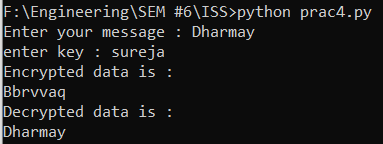
de\_sml=decrypt(en\_sml,ke1)

print("Decrypted data is : ")

for x in de\_sml:

print(x,end="")

* **OUTPUT :-**

****

**Practical 5**

Write a program to perform encryption and decryption using Hill Cipher Technique.

* **CODE :-**

import math

def generate\_matrix(key):

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

matrix = []

for i in key.upper():

if i not in matrix:

matrix.append(i)

for j in alphabet:

if j not in matrix:

matrix.append(j)

matrix\_group = []

for e in range(5):

matrix\_group.append('')

matrix\_group[0] = matrix[0:5]

matrix\_group[1] = matrix[5:10]

matrix\_group[2] = matrix[10:15]

matrix\_group[3] = matrix[15:20]

matrix\_group[4] = matrix[20:25]

return matrix\_group

def msg\_to\_digraphs(msg):

message = []

for k in msg:

message.append(k)

for j in range(len(message)):

if " " in message:

message.remove(" ")

i = 0

for e in range(len(message) // 2):

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

message.insert(i + 1, 'X')

i = i + 2

if len(message) % 2 == 1:

message.append("X")

i = 0

new = []

for x in range(1, len(message) // 2 + 1):

new.append(message[i:i + 2])

i = i + 2

return new

def find\_position(key, letter):

x = y = 0

for i in range(5):

for j in range(5):

if key[i][j] == letter:

x = i

y = j

return x, y

def encryption(message):

message = msg\_to\_digraphs(message)

key\_matrix = generate\_matrix(key)

cipher = []

for e in message:

p1, q1 = find\_position(key\_matrix, e[0])

p2, q2 = find\_position(key\_matrix, e[1])

if p1 == p2:

if q1 == 4:

q1 = -1

if q2 == 4:

q2 = -1

cipher.append(key\_matrix[p1][q1 + 1])

cipher.append(key\_matrix[p1][q2 + 1])

elif q1 == q2:

if p1 == 4:

p1 = -1;

if p2 == 4:

p2 = -1;

cipher.append(key\_matrix[p1 + 1][q1])

cipher.append(key\_matrix[p2 + 1][q2])

else:

cipher.append(key\_matrix[p1][q2])

cipher.append(key\_matrix[p2][q1])

return ''.join(cipher)

def cipher\_to\_digraphs(cipher):

i = 0

new = []

for x in range(len(cipher) // 2):

new.append(cipher[i:i + 2])

i = i + 2

return new

def decryption(cipher):

cipher = cipher\_to\_digraphs(cipher)

key\_matrix = generate\_matrix(key)

plaintext = []

for e in cipher:

p1, q1 = find\_position(key\_matrix, e[0])

p2, q2 = find\_position(key\_matrix, e[1])

if p1 == p2:

if q1 == 4:

q1 = -1

if q2 == 4:

q2 = -1

plaintext.append(key\_matrix[p1][q1 - 1])

plaintext.append(key\_matrix[p1][q2 - 1])

elif q1 == q2:

if p1 == 4:

p1 = -1;

if p2 == 4:

p2 = -1;

plaintext.append(key\_matrix[p1 - 1][q1])

plaintext.append(key\_matrix[p2 - 1][q2])

else:

plaintext.append(key\_matrix[p1][q2])

plaintext.append(key\_matrix[p2][q1])

for i in range(len(plaintext)):

if "X" in plaintext:

plaintext.remove("X")

output = ""

for j in plaintext:

output += j

return output.lower()

key = input("Enter key : ")

message = input("Enter original message : ")

print(msg\_to\_digraphs(message))

print(generate\_matrix(key))

print("Encryption")

msg\_final=message.upper()

print(encryption(msg\_final))

choice=int(input("Continue decryption process, press 1:"))

if(choice==1):

key = input("Enter key : ")

cipher = input("Enter cipher text: ")

print("Decryption")

cipher\_final=cipher.upper()

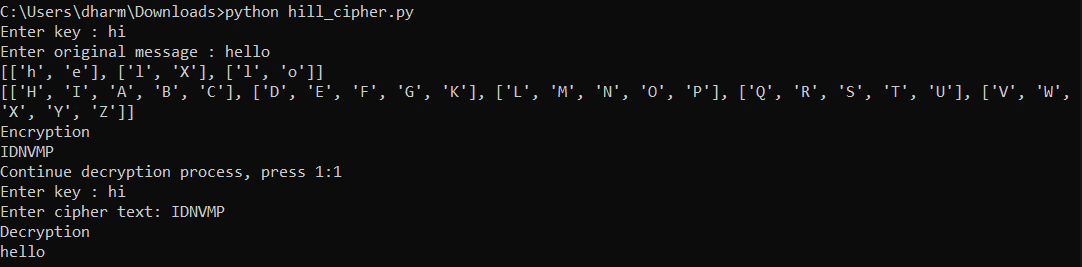
print(decryption(cipher\_final))

else:

print("Invalid Choice.")

exit();

* **OUTPUT :-**



**Practical 6**

Write a program to perform encryption and decryption using Playfair Cipher Technique

* **CODE :-**

def encryption(pt,key):

k = 0

cf=""

while k < len(pt):

i1 = k

i2 = k + 1

k = k + 2

for i in range(5):

for j in range(5):

if pt[i1] == key[i][j] or (pt[0] =='J' and key[i][j] == 'I'):

r1=i

c1=j

elif pt[i2] == key[i][j] or (pt[1] =='J' and key[i][j] == 'I'):

r2=i

c2=j

if r1 == r2 :

cf = cf + key[r1][(c1+1)%5] + key[r1][(c2+1)%5]

elif c1 == c2 :

cf = cf + key[(r1+1)%5][c1] + key[(r2+1)%5][c1]

else:

cf = cf + key[r1][c2] + key[r2][c1]

return cf

def decryption(cf,key):

pt=" "

k = 0

while k < len(cf):

i1 = k

i2 = k + 1

k = k + 2

for i in range(5):

for j in range(5):

if cf[i1] == key[i][j] or (cf[0] =='J' and key[i][j] == 'I'):

r1=i

c1=j

elif cf[i2] == key[i][j] or (cf[1] =='J' and key[i][j] == 'I'):

r2=i

c2=j

if r1 == r2 :

pt = pt + key[r1][(c1-1)%5] + key[r1][(c2-1)%5]

elif c1 == c2 :

pt = pt + key[(r1-1)%5][c1] + key[(r2-1)%5][c1]

else:

pt = pt + key[r1][c2] + key[r2][c1]

return pt

keymatrix = [['M','O','N','A','R'],

['C','H','Y','B','D'],

['E','F','G','I','K'],

['L','P','Q','S','T'],

['U','V','W','X','Z']]

for i in range(5):

for j in range(5):

print(keymatrix[i][j], end = " ")

print()

pt = input("Enter Plain Text = ")

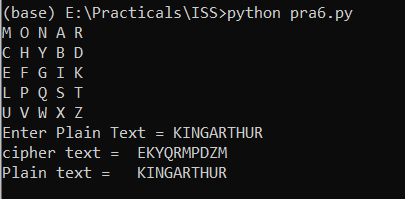
cf = encryption(pt,keymatrix)

print("cipher text = ",cf)

var = decryption(cf,keymatrix)

print("Plain text = ",var)

* **OUTPUT :-**



**Practical 7**

Write a program to perform encryption and decryption using Railfence Cipher.

* **CODE :-**

def encryptRF(text, key):

rail = [['\n' for i in range(len(text))]

for j in range(key)]

dir\_down = False

row, col = 0, 0

for i in range(len(text)):

if (row == 0) or (row == key - 1):

dir\_down = not dir\_down

rail[row][col] = text[i]

col += 1

if dir\_down:

row += 1

else:

row -= 1

result = []

for i in range(key):

for j in range(len(text)):

if rail[i][j] != '\n':

result.append(rail[i][j])

return("" . join(result))

def decryptRF(ciphertext, key):

rail = [['\n' for i in range(len(ciphertext))]

for j in range(key)]

dir\_down = None

row, col = 0, 0

for i in range(len(ciphertext)):

if row == 0:

dir\_down = True

if row == key - 1:

dir\_down = False

rail[row][col] = '\*'

col += 1

if dir\_down:

row += 1

else:

row -= 1

index = 0

for i in range(key):

for j in range(len(ciphertext)):

if ((rail[i][j] == '\*') and

(index < len(ciphertext))):

rail[i][j] = ciphertext[index]

index += 1

result = []

row, col = 0, 0

for i in range(len(ciphertext)):

if row == 0:

dir\_down = True

if row == key-1:

dir\_down = False

if (rail[row][col] != '\*'):

result.append(rail[row][col])

col += 1a

if dir\_down:

row += 1

else:

row -= 1

return("".join(result))

pt=input("Enter Plain Text:")

p=int(input("Enter key:"))

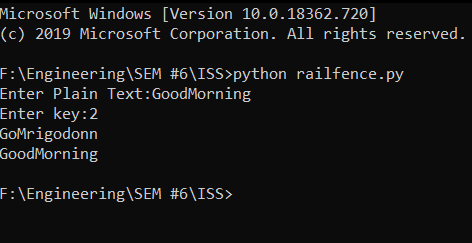
en\_str=encryptRF(pt,p)

de\_str=decryptRF(en\_str,p)

print(en\_str)

print(de\_str)

* **OUTPUT :-**



**Practical 8**

**Write a program to perform encryption and decryption using columnar transposition algorithm**

* **CODE :-**

import math

key = "epic"

def encryptMessage(msg):

cipher = ""

k\_indx = 0

msg\_len = float(len(msg))

msg\_lst = list(msg)

key\_lst = sorted(list(key))

col = len(key)

row = int(math.ceil(msg\_len / col))

fill\_null = int((row \* col) - msg\_len)

msg\_lst.extend('\_' \* fill\_null)

matrix = [msg\_lst[i: i + col]

for i in range(0, len(msg\_lst), col)]

for \_ in range(col):

c\_idx = key.index(key\_lst[k\_indx])

cipher += ''.join([row[c\_idx]

for row in matrix])

k\_indx += 1

return cipher

def decryptMessage(cipher):

msg = ""

k\_indx = 0

msg\_indx = 0

msg\_len = float(len(cipher))

msg\_lst = list(cipher)

col = len(key)

row = int(math.ceil(msg\_len / col))

key\_lst = sorted(list(key))

d\_cipher = []

for \_ in range(row):

d\_cipher += [[None] \* col]

for \_ in range(col):

c\_idx = key.index(key\_lst[k\_indx])

for j in range(row):

d\_cipher[j][c\_idx] = msg\_lst[msg\_indx]

msg\_indx += 1

k\_indx += 1

try:

msg = ''.join(sum(d\_cipher, []))

except TypeError:

raise TypeError("This program cannot",

"handle repeating words.")

null\_count = msg.count('\_')

if null\_count > 0:

return msg[: -null\_count]

return msg

msg = "Dharmay Sureja"

cipher = encryptMessage(msg)

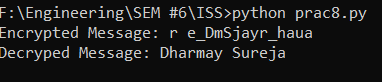
print("Encrypted Message: {}".

format(cipher))

print("Decryped Message: {}".

format(decryptMessage(cipher)))

* **OUTPUT :-**



**Practical : 9**

**Write a function to find out multiplicative inverse using Extended Euclidean algorithm and perform encryption and decryption using affine cipher algorithm.**

* **Extended Euclidean Algorithm**
* **CODE :-**

import math

def extendedeuclidian(r1,r2):

q=0

r=0

t1=0

t2=1

s1=1

s2=0

s=0

t=0

i=0

while r2!=0:

if i>0:

r1=r2

r2=r

if r2==0:

q=0

break

q=math.floor(r1/r2)

r=r1-(q\*r2)

if i>0:

t1=t2

t2=t

s1=s2

s2=s

t=t1-(q\*t2)

s=s1=(q\*s2)

i+=1

t1=t2

t2=t

s1=s2

s2=s

return r1,t1

a=int(input("Enter first value : "))

b=int(input("Enter second value : "))

r1=max(a,b)

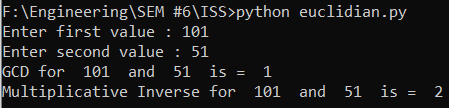
r2=min(a,b)

gcd,inverse=extendedeuclidian(r1,r2)

print("GCD for ",r1," and ",r2," is = ",gcd)

print("Multiplicative Inverse for ",r1," and ",r2," is = ",inverse)

* **OUTPUT :-**

****

* **Affine Cipher**
* **CODE :-**

import math

pt=list(input("Enter plaintext : "))

a=int(input("Enter value of a = "))

b=int(input("Enter value of b = "))

def encryption(pt,a,b):

en=[]

i=0

while i<len(pt):

k=0

if pt[i].isupper():

k=65

else:

k=97

x=ord(pt[i])-k

fox=chr((((a\*x)+b)%26)+k)

en.append(fox)

i+=1

return en

def euclidian\_inverse(r1,r2):

q=0

r=0

t1=0

t2=1

t=0

i=0

while r2!=0:

if i>0:

r1=r2

r2=r

if r2==0:

q=0

break

q=math.floor(r1/r2)

r=r1-(q\*r2)

if i>0:

t1=t2

t2=t

t=t1-(q\*t2)

i+=1

t1=t2

t2=t

inverse=t1%26

return inverse

def decryption(en,c,b):

de=[]

i=0

while i<len(en):

k=0

if en[i].isupper():

k=65

else:

k=97

x=ord(en[i])-k

fox=chr(((c\*(x-b))%26)+k)

de.append(fox)

i+=1

return de

en=encryption(pt,a,b)

#print(en)

print("Encrypted text is : ")

i=0

while i<len(en):

print(en[i],end="")

i+=1

print()

c=euclidian\_inverse(26,a)

de=decryption(en,c,b)

print("Decrypted text is : ")

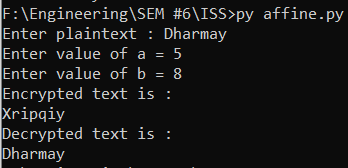
i=0

while i<len(de):

print(de[i],end="")

i+=1

* **OUTPUT :-**



**Practical 10**

Write a program to perform Diffie Hellman key Exchange and perform Caesar cipher algorithm

* **CODE :-**

import random

import hashlib

import sys

g=7

p=11

a=3

b=6

A = (g\*\*a) % p

B = (g\*\*b) % p

print('g: ',g,' , n: ',p, ' ')

print('\nA calculates:')

print('a : ',a)

print('A value (A): ',A,' (g^a) mod p')

print('\nB calculates:')

print('b : ',b)

print('B value (B): ',B,' (g^b) mod p')

print('\nA calculates:')

keyA=(B\*\*a) % p

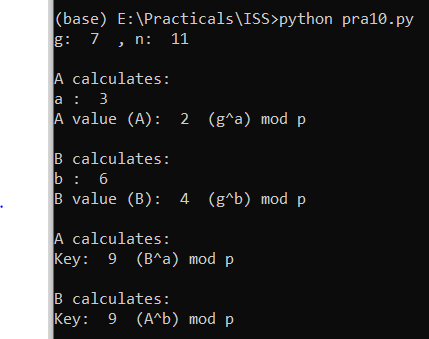
print('Key: ',keyA,' (B^a) mod p')

print('\nB calculates:')

keyB=(A\*\*b) % p

print('Key: ',keyB,' (A^b) mod p')

* **OUTPUT :-**



**Practical 11**

Write a program to test prime number using Miller Rabin Theorem.

* **CODE :-**

import random

def power(x, y, p):

res = 1;

x = x % p;

while (y > 0):

if (y & 1):

res = (res \* x) % p;

y = y>>1; # y = y/2

x = (x \* x) % p;

return res;

def miillerTest(d, n):

a = 2 + random.randint(1, n - 4);

x = power(a, d, n);

if (x == 1 or x == n - 1):

return True;

while (d != n - 1):

x = (x \* x) % n;

d \*= 2;

if (x == 1):

return False;

if (x == n - 1):

return True;

return False;

def isPrime( n, k):

if (n <= 1 or n == 4):

return False;

if (n <= 3):

return True;

d = n - 1;

while (d % 2 == 0):

d //= 2;

for i in range(k):

if (miillerTest(d, n) == False):

return False;

return True;

k = 4;

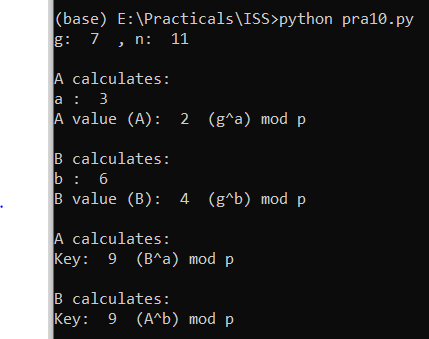
print("All primes smaller than 156: ");

for n in range(1,156):

if (isPrime(n, k)):

print(n , end=" ");

* **OUTPUT :-**



**Practical 12**

Write a program to perform encryption and decryption using RSA algorithm.

* **CODE :-**

from decimal import Decimal

def gcd(a,b):

if b==0:

return a

else:

return gcd(b,a%b)

p = int(input('Enter the value of p = '))

q = int(input('Enter the value of q = '))

no = int(input('Enter the value of text = '))

n = p\*q

t = (p-1)\*(q-1)

for e in range(2,t):

if gcd(e,t)== 1:

break

for i in range(1,10):

x = 1 + i\*t

if x % e == 0:

d = int(x/e)

break

ctt = Decimal(0)

ctt =pow(no,e)

ct = ctt % n

dtt = Decimal(0)

dtt = pow(ct,d)

dt = dtt % n

print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(ct)+' decrypted text = '+str(dt))

* **OUTPUT :-**

