Name :- Kaustubh Shrikant Kabra

Class:- TE Computer

ERP :-38

Subject :-LP2(AI) (BFS and DFS)

Code:-

```
import collections

# DFS algorithm
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)

    print(start)

    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

# BFS algorithm
def bfs(graph, root):

    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:

        # Dequeue a vertex from queue
        vertex = queue.popleft()
        print(str(vertex) + " ", end="")

        # If not visited, mark it as visited, and
        # enqueue it
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)

vertex = []
Connections = []

no_vertex = int(input("Enter total number of vertex : "))
start_vertex = int(input("Enter starting vertex : "))

for i in range(no_vertex):
    vertex_n = int(input("Enter vertex " + str(i + 1) + " : "))
    # creating an empty list
```

```
    vertex.append(vertex_n)
    temp = []


    # number of elements as input
    n = int(input("Enter number of connections : "))

    # iterating till the range
    for i in range(0, n):
        ele = int(input("Enter connected to " + str(vertex_n) + " : "))
        temp.append(ele)  # adding the element

    print(temp)
    Connections.append(temp)


print(vertex)
print(Connections)
graph={ vertex[i]:Connections[i] for i in range(no_vertex)}
graph_dfs = {vertex[i]:set(Connections[i]) for i in range(no_vertex)}
print(graph)


flag = 1
while flag == 1:
    print("/**************MENU***************/")
    print("1. DFS")
    print("2. BFS ")
    print("3. Exit ")
    choice = int(input("Enter your choice : "))

    if choice == 1:
        print("Following is DFS :")
        print(dfs(graph_dfs, start_vertex))
    elif choice == 2:
        print("Following is BFS : " )
        print(bfs(graph, start_vertex))
    elif choice == 3:
        print("Exit")
        flag = 0
    else:
        print("Wrong Choice,Please Choose Another Option.")
```

Output:-

Enter total number of vertex : 4

Enter starting vertex : 2

Enter vertex 1 : 0

Enter number of connections : 2

Enter connected to 0 : 1

Enter connected to 0 : 2

[1, 2]

Enter vertex 2 : 1

Enter number of connections : 1

Enter connected to 1 : 2

[2]

Enter vertex 3 : 2

Enter number of connections : 2

Enter connected to 2 : 0

Enter connected to 2 : 3

[0, 3]

Enter vertex 4 : 3

Enter number of connections : 1

Enter connected to 3 : 3

[3]

[0, 1, 2, 3]

[[1, 2], [2], [0, 3], [3]]

{0: [1, 2], 1: [2], 2: [0, 3], 3: [3]}

/**********MENU**********/

1. DFS

2. BFS

3. Exit

Enter your choice : 1

Following is DFS :

2

0

1

3

/**********MENU**********/

1. DFS

2. BFS

3. Exit

Enter your choice : 2

Following is BFS :

2 0 3 1

/**********MENU**********/

1. DFS

2. BFS

3. Exit

Enter your choice : 5

Wrong Choice,Please Choose Another Option.

/**********MENU**********/

1. DFS

2. BFS

3. Exit

Enter your choice : 3

Exit

Process finished with exit code 0

Name :- Kaustubh Shrikant Kabra

CLass:- TE Computer

ERP :-38

Subject :-LP2(AI) (A Star)

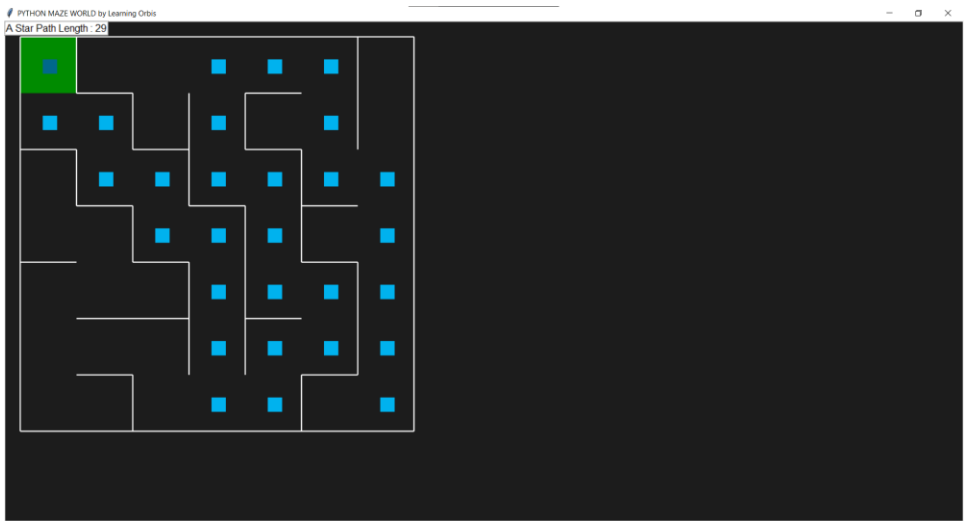
Code:-

```
from pyamaze import maze,agent,textLabel
from queue import PriorityQueue
def h(cell1,cell2):
    x1,y1=cell1
    x2,y2=cell2

    return abs(x1-x2) + abs(y1-y2)
def aStar(m):
    start=(m.rows,m.cols)
    g_score={cell:float('inf') for cell in m.grid}
    g_score[start]=0
    f_score={cell:float('inf') for cell in m.grid}
    f_score[start]=h(start,(1,1))

    open=PriorityQueue()
    open.put((h(start,(1,1)),h(start,(1,1)),start))
    aPath={}
    while not open.empty():
        currCell=open.get()[2]
        if currCell==(1,1):
            break
        for d in 'ESNW':
            if m.maze_map[currCell][d]==True:
                if d=='E':
                    childCell=(currCell[0],currCell[1]+1)
                if d=='W':
                    childCell=(currCell[0],currCell[1]-1)
                if d=='N':
                    childCell=(currCell[0]-1,currCell[1])
                if d=='S':
                    childCell=(currCell[0]+1,currCell[1])

                temp_g_score=g_score[currCell]+1
                temp_f_score=temp_g_score+h(childCell,(1,1))

                if temp_f_score < f_score[childCell]:
                    g_score[childCell]= temp_g_score
                    f_score[childCell]= temp_f_score
                    open.put((temp_f_score,h(childCell,(1,1)),childCell))
                    aPath[childCell]=currCell
    fwdPath={}
    cell=(1,1)
```

```
    while cell!=start:
        fwdPath[aPath[cell]]=cell
        cell=aPath[cell]
    return fwdPath

if __name__=='__main__':
    x = int(input("Enter X for X*X Maze :"))
    m=maze(x,x)
    m.CreateMaze()
    path=aStar(m)

    a=agent(m,footprints=True)
    m.tracePath({a:path})
    l=textLabel(m,'A Star Path Length',len(path)+1)

    m.run()
```

Output:-

Enter X for X*X Maze :7



Name :- Kaustubh Shrikant Kabra

CLass:- TE Computer

ERP :-38

Subject :-LP2(AI) (N Queens)

Code:-

```
# Function to check if two queens threaten each other or not
def isSafe(mat, r, c):
    # return false if two queens share the same column
    for i in range(r):
        if mat[i][c] == 'Q':
            return False

    # return false if two queens share the same `` diagonal
    (i, j) = (r, c)
    while i >= 0 and j >= 0:
        if mat[i][j] == 'Q':
            return False
        i = i - 1
        j = j - 1

    # return false if two queens share the same `/` diagonal4
    (i, j) = (r, c)
    while i >= 0 and j < len(mat):
        if mat[i][j] == 'Q':
            return False
        i = i - 1
        j = j + 1

    return True


def printSolution(mat):
    for r in mat:
        print(str(r).replace(',', '').replace('\'', ''))
    print()


def nQueen(mat, r):
    # if `N` queens are placed successfully, print the solution
    if r == len(mat):
        printSolution(mat)
        return

    # place queen at every square in the current row `r`
    # and recur for each valid movement
```

```
    for i in range(len(mat)):

        # if no two queens threaten each other
        if isSafe(mat, r, i):
            # place queen on the current square
            mat[r][i] = 'Q'

            # recur for the next row
            nQueen(mat, r + 1)

            # backtrack and remove the queen from the current square
            mat[r][i] = '–'


if __name__ == '__main__':
    # `N × N` chessboard
    N = int(input("Enter Number of Queen on N*N Chess Board :"))

    # `mat[][]` keeps track of the position of queens in
    # the current configuration
    mat = [['–' for x in range(N)] for y in range(N)]

    nQueen(mat, 0)
```

Output:-

Enter Number of Queen on N*N Chess Board :4

[– Q – –]

[– – – Q]

[Q – – –]

[– – Q –]

[– – Q –]

[Q – – –]

[– – – Q]

[– Q – –]

Process finished with exit code 0

```
C:\Users\asus\PycharmProjects\AStar\Scripts\python.exe "C:/Users/asus/PycharmProjects/AStar/N Queen Problem.py"
Enter Number of Queen on N*N Chess Board :4
[- Q - -]
[- - - Q]
[Q - - -]
[- - Q -]

[- - Q -]
[Q - - -]
[- - - Q]
[- Q - -]


Process finished with exit code 0
```

Name :- Kaustubh Shrikant Kabra

Class:- TE Computer

ERP :-38

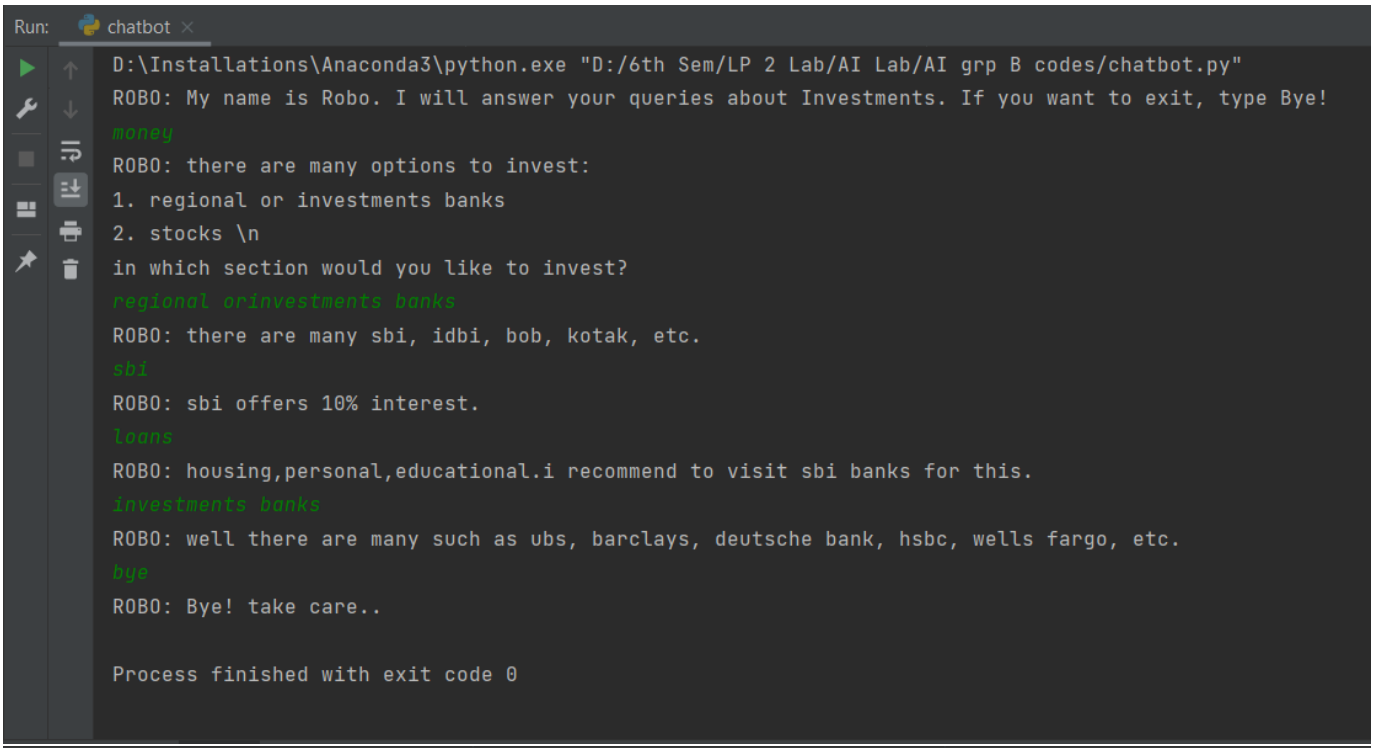
Subject :-LP2(AI) (Chatbot)

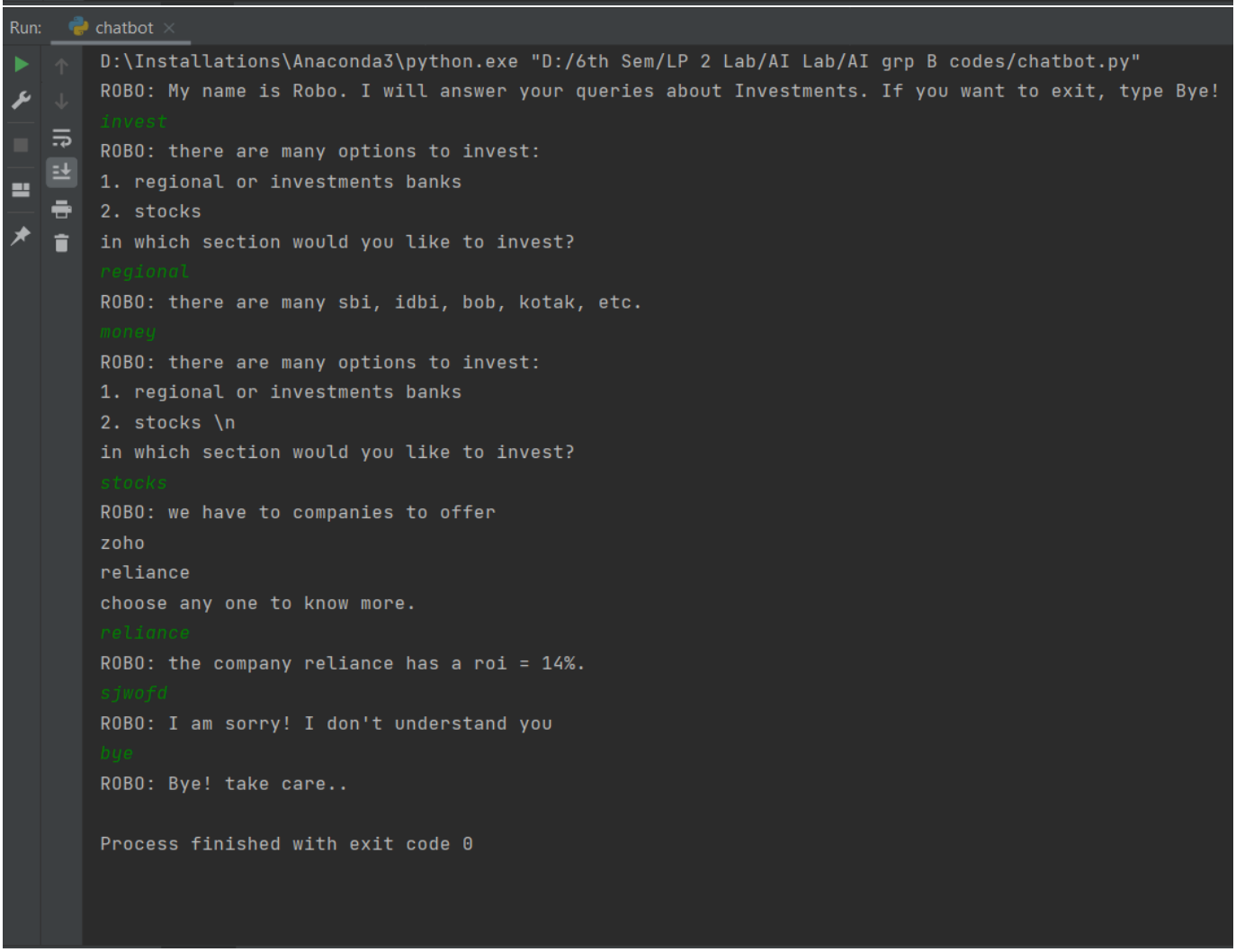
Code:-

```
import io
import random
import string
import warnings
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')
import nltk
from nltk.stem import WordNetLemmatizer
# nltk.download('popular', quiet=True)
# nltk.download('punkt')
# nltk.download('wordnet')

with open('chatbot.txt','r', encoding='utf8', errors ='ignore') as fin:
    raw = fin.read().lower()

#Tokenisation
sent_tokens = nltk.sent_tokenize(raw)
word_tokens = nltk.word_tokenize(raw)

# Preprocessing
lemmer = WordNetLemmatizer()
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))


# Keyword Matching
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey","Helo")
GREETING_RESPONSES = ["hi", "hey", "hi there", "hello", "I am glad! You are talking to me"]

def greeting(sentence):
    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)

def response(user_response):
    robo_response="
```

```
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx=vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if(req_tfidf==0):
        robo_response=robo_response+"I am sorry! I don't understand you"
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response

flag=True
print("ROBO: My name is Robo. I will answer your queries about Investments. If you want to exit, type Bye!")
while(flag==True):
    user_response = input()
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            print("ROBO: You are welcome..")
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response))
            else:
                print("ROBO: ",end="")
                res = response(user_response)
                nlines = res.count('\n')
                if nlines > 0:
                    res = res.split("\n",1)[1]
                print(res)
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("ROBO: Bye! take care..")
```

Output:-


Run:
chatbot
D:\Installations\Anaconda3\python.exe "D:/6th Sem/LP 2 Lab/AI Lab/AI grp B codes/chatbot.py"
ROBO: My name is Robo. I will answer your queries about Investments. If you want to exit, type Bye!
money
ROBO: there are many options to invest:
1. regional or investments banks
2. stocks \n
in which section would you like to invest?
regional orinvestments banks
ROBO: there are many sbi, idbi, bob, kotak, etc.
sbi
ROBO: sbi offers 10% interest.
loans
ROBO: housing,personal,educational.i recommend to visit sbi banks for this.
investments banks
ROBO: well there are many such as ubs, barclays, deutsche bank, hsbc, wells fargo, etc.
bye
ROBO: Bye! take care..
Process finished with exit code 0


Run:
chatbot
D:\Installations\Anaconda3\python.exe "D:/6th Sem/LP 2 Lab/AI Lab/AI grp B codes/chatbot.py"
ROBO: My name is Robo. I will answer your queries about Investments. If you want to exit, type Bye!
invest
ROBO: there are many options to invest:
1. regional or investments banks
2. stocks
in which section would you like to invest?
regional
ROBO: there are many sbi, idbi, bob, kotak, etc.
money
ROBO: there are many options to invest:
1. regional or investments banks
2. stocks \n
in which section would you like to invest?
stocks
ROBO: we have to companies to offer
zoho
reliance
choose any one to know more.
reliance
ROBO: the company reliance has a roi = 14%.
sjwofd
ROBO: I am sorry! I don't understand you
bye
ROBO: Bye! take care..
Process finished with exit code 0

Name :- Kaustubh Shrikant Kabra

Class:- TE Computer

ERP :-38

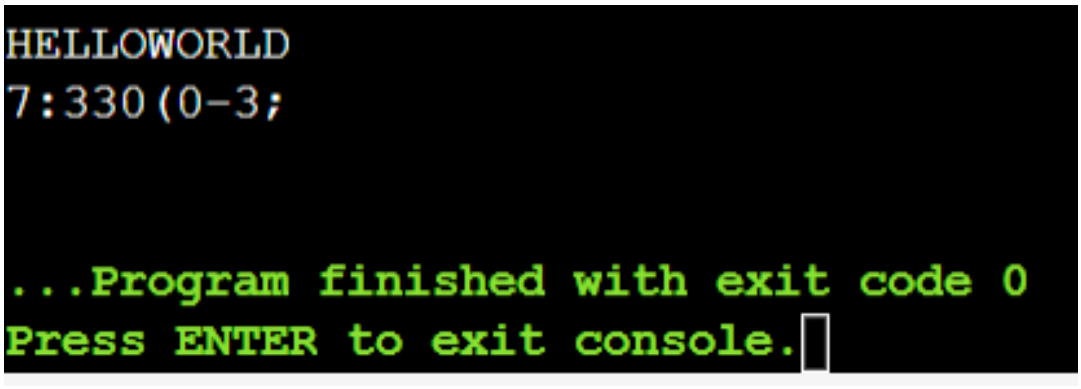
Subject :-LP2(IS) (Logical Operations)

Code:-

```
#include <iostream.h>
//using namespace std;
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
void main()
{
   //clrscr();
   char str[]="HELLOWORLD";
   char str1[11];
   char str2[11];
   int i,len;
   len = strlen(str);

   for(i=0;i<len;i++)
   {
      str1[i]=str[i] & 127;
      cout<<str1[i];
   }
   cout<<"\n";
   for(i=0;i<len;i++)
   {
      str2[i] = str[i] ^ 127;
      cout<<str2[i];
   }
   cout<<"\n";
```

```
    getch();
}
```

Output:-



Name :- Kaustubh Shrikant Kabra

CLass:- TE Computer

ERP :-38

Subject :-LP2(IS) (Transposition)

Code:-

```
import math

key = "HACK"

# Encryption
def encryptMessage(msg):
    cipher = ""

   # track key indices
    k_indx = 0

   msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))

   # calculate column of the matrix
    col = len(key)

   # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

   # add the padding character '_' in empty
    # the empty cell of the matix
   fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

   # create Matrix and insert message and
   # padding characters row-wise
   matrix = [msg_lst[i: i + col]
            for i in range(0, len(msg_lst), col)]

   # read matrix column-wise using key
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx]
                           for row in matrix])
        k_indx += 1

   return cipher
```

```
# Decryption
def decryptMessage(cipher):
    msg = ""

   # track key indices
    k_indx = 0

   # track msg indices
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)

   # calculate column of the matrix
    col = len(key)

   # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

   # convert key into list and sort
   # alphabetically so we can access
   # each character by its alphabetical position.
    key_lst = sorted(list(key))

   # create an empty matrix to
   # store deciphered message
    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]

   # Arrange the matrix column wise according
   # to permutation order by adding into new matrix
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])

       for j in range(row):
            dec_cipher[j][curr_idx] = msg_lst[msg_indx]
            msg_indx += 1
        k_indx += 1

   # convert decrypted msg matrix into a string
    try:
        msg = ''.join(sum(dec_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

# Driver Code

msg = (input("Enter Message: "))


cipher = encryptMessage(msg)
print("Encrypted Message: {}".
                format(cipher))

print("Decryped Message: {}".
      format(decryptMessage(cipher)))
```

Output:-

Enter Message: Its KK29 aka Kaustubh

Encrypted Message: tKaKt_s2kau_IK sh 9aub_

Decryped Message: Its KK29 aka Kaustubh

Process finished with exit code 0

```
C:\Users\asus\PycharmProjects\AStar\Scripts\python.exe "C:/Users/asus/PycharmProjects/AStar/2. Transposition.py"
Enter Message: Its KK29 aka Kaustubh
Encrypted Message: tKaKt_s2kau_IK  sh 9aub_
Decryped Message: Its KK29 aka Kaustubh

Process finished with exit code 0
```

Name :- Kaustubh Shrikant Kabra

CLass:- TE Computer

ERP :-38

Subject :-LP2(IS) (AES)

Code:-

```
import hashlib
from base64 import b64decode, b64encode

from Crypto import Random
from Crypto.Cipher import AES


class AESCipher(object):
    def __init__(self, key):
        self.block_size = AES.block_size
        self.key = hashlib.sha256(key.encode()).digest()

    def encrypt(self, plain_text):
        plain_text = self.__pad(plain_text)
        iv = Random.new().read(self.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        encrypted_text = cipher.encrypt(plain_text.encode())
        return b64encode(iv + encrypted_text).decode("utf-8")

    def decrypt(self, encrypted_text):
        encrypted_text = b64decode(encrypted_text)
        iv = encrypted_text[:self.block_size]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        plain_text = cipher.decrypt(encrypted_text[self.block_size:]).decode("utf-8")
        return self.__unpad(plain_text)

    def __pad(self, plain_text):
        number_of_bytes_to_pad = self.block_size - len(plain_text) % self.block_size
        ascii_string = chr(number_of_bytes_to_pad)
        padding_str = number_of_bytes_to_pad * ascii_string
        padded_plain_text = plain_text + padding_str
        return padded_plain_text

    @staticmethod
    def __unpad(plain_text):
        last_character = plain_text[len(plain_text) - 1:]
        return plain_text[:-ord(last_character)]


key = input("Enter Key: ")
aes = AESCipher(key)
```

```
flag = 1
while flag == 1:
   print("/**************MENU**************/")
   print("1. Encryption")
   print("2. Decryption")
   print("3. Exit ")
   choice = int(input("Enter your choice : "))

   if choice == 1:
      message = input("Enter message to encrypt: ")
      encryptedMessage = aes.encrypt(message)
      print("Encrypted Message:", encryptedMessage)

   elif choice == 2:
      message = input("Enter message to decrypt: ")
      decryptedMessage = aes.decrypt(message)
      print("Decrypted Message:", decryptedMessage)
   elif choice == 3:
      print("Exit")
      flag = 0
   else:
      print("Wrong Choice,Please Choose Another Option.")
```

Output:-

Enter Key: AISSMSIOIT

/**************MENU**************/

1. Encryption

2. Decryption

3. Exit

Enter your choice : 1

Enter Message to Encrypt: Its KK29 aka Kaustubh

Encrypted Message:
K4qVJgSw3vwuRZnUD5YEzVHk41HP796bfHGz7iKNAt1MyLxjzsAUyE7p+5Ape5xo

/**************MENU**************/

1. Encryption

2. Decryption

3. Exit

Enter your choice : 2

Enter Message to Decrypt:
K4qVJgSw3vwuRZnUD5YEzVHk41HP796bfHGz7iKNAt1MyLxjzsAUyE7p+5Ape5xo

Decrypted Message: Its KK29 aka Kaustubh

/**********MENU**********/

1. Encryption

2. Decryption

3. Exit

Enter your choice : 7

Wrong Choice,Please Choose Another Option.

/**********MENU**********/

1. Encryption

2. Decryption

3. Exit

Enter your choice : 3

Exit

Process finished with exit code 0

```
C:\Users\asus\PycharmProjects\AStar\Scripts\python.exe "C:/Users/asus/PycharmProjects/AStar/4. AES.py"
Enter Key: AISSMSIOIT
/*************MENU**************/
1. Encryption
2. Decryption
3. Exit
Enter your choice : 1
Enter Message to Encrypt: Its KK29 aka Kaustubh
Encrypted Message: K4qVJgSw3vwuRZnUD5YEzVHk41HP796bfHGz7iKNAt1MyLxjzsAUyE7p+5Ape5xo
/*************MENU**************/
1. Encryption
2. Decryption
3. Exit
Enter your choice : 2
Enter Message to Decrypt: K4qVJgSw3vwuRZnUD5YEzVHk41HP796bfHGz7iKNAt1MyLxjzsAUyE7p+5Ape5xo
Decrypted Message: Its KK29 aka Kaustubh
/*************MENU**************/
1. Encryption
2. Decryption
3. Exit
Enter your choice : 7
Wrong Choice,Please Choose Another Option.
/*************MENU**************/
1. Encryption
2. Decryption
3. Exit
Enter your choice : 3
Exit

Process finished with exit code 0
```

Name :- Kaustubh Shrikant Kabra

CLass:- TE Computer

ERP :-38

Subject :-LP2(IS) (DES)

Code:-

```
# Hexadecimal to binary conversion
def hex2bin(s):
   mp = {'0' : "0000",
      '1' : "0001",
      '2' : "0010",
      '3' : "0011",
      '4' : "0100",
      '5' : "0101",
      '6' : "0110",
      '7' : "0111",
      '8' : "1000",
      '9' : "1001",
      'A' : "1010",
      'B' : "1011",
      'C' : "1100",
      'D' : "1101",
      'E' : "1110",
      'F' : "1111" }
   bin = ""
   for i in range(len(s)):
      bin = bin + mp[s[i]]
   return bin

# Binary to hexadecimal conversion
def bin2hex(s):
   mp = {"0000" : '0',
      "0001" : '1',
      "0010" : '2',
      "0011" : '3',
      "0100" : '4',
      "0101" : '5',
      "0110" : '6',
      "0111" : '7',
      "1000" : '8',
      "1001" : '9',
      "1010" : 'A',
      "1011" : 'B',
      "1100" : 'C',
      "1101" : 'D',
      "1110" : 'E',
```

```
      "1111" : 'F' }
    hex = ""
    for i in range(0,len(s),4):
      ch = ""
      ch = ch + s[i]
      ch = ch + s[i + 1]
      ch = ch + s[i + 2]
      ch = ch + s[i + 3]
      hex = hex + mp[ch]

   return hex

# Binary to decimal conversion
def bin2dec(binary):

   binary1 = binary
    decimal, i, n = 0, 0, 0
    while(binary != 0):
      dec = binary % 10
      decimal = decimal + dec * pow(2, i)
      binary = binary//10
      i += 1
    return decimal

# Decimal to binary conversion
def dec2bin(num):
    res = bin(num).replace("0b", "")
    if(len(res)%4 != 0):
      div = len(res) / 4
      div = int(div)
      counter =(4 * (div + 1)) - len(res)
      for i in range(0, counter):
        res = '0' + res
    return res

# Permute function to rearrange the bits
def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
      permutation = permutation + k[arr[i] - 1]
    return permutation

# shifting the bits towards left by nth shifts
def shift_left(k, nth_shifts):
    s = ""
    for i in range(nth_shifts):
      for j in range(1,len(k)):
        s = s + k[j]
```

```
      s = s + k[0]
      k = s
      s = ""
    return k

# calculating xow of two strings of binary number a and b
def xor(a, b):
    ans = ""
    for i in range(len(a)):
      if a[i] == b[i]:
        ans = ans + "0"
      else:
        ans = ans + "1"
    return ans

# Table of Position of 64 bits at initial level: Initial Permutation Table
initial_perm = [58, 50, 42, 34, 26, 18, 10, 2,
            60, 52, 44, 36, 28, 20, 12, 4,
            62, 54, 46, 38, 30, 22, 14, 6,
            64, 56, 48, 40, 32, 24, 16, 8,
            57, 49, 41, 33, 25, 17, 9, 1,
            59, 51, 43, 35, 27, 19, 11, 3,
            61, 53, 45, 37, 29, 21, 13, 5,
            63, 55, 47, 39, 31, 23, 15, 7]

# Expansion D-box Table
exp_d = [32, 1 , 2 , 3 , 4 , 5 , 4 , 5,
      6 , 7 , 8 , 9 , 8 , 9 , 10, 11,
      12, 13, 12, 13, 14, 15, 16, 17,
      16, 17, 18, 19, 20, 21, 20, 21,
      22, 23, 24, 25, 24, 25, 26, 27,
      28, 29, 28, 29, 30, 31, 32, 1 ]

# Straight Permutation Table
per = [ 16, 7, 20, 21,
      29, 12, 28, 17,
      1, 15, 23, 26,
      5, 18, 31, 10,
      2, 8, 24, 14,
      32, 27, 3, 9,
      19, 13, 30, 6,
      22, 11, 4, 25 ]

# S-box Table
sbox = [[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
      [ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
      [ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
      [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 ]],
```

```
    [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
       [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
       [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
     [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 ]],

     [ [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
     [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
     [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
       [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 ]],

    [ [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
     [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
     [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
       [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14] ],

    [ [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
     [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
       [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
     [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 ]],

    [ [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
     [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
       [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
       [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13] ],

    [ [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
     [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
       [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
       [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12] ],

    [ [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
       [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
       [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
       [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11] ] ]

# Final Permutation Table
final_perm = [ 40, 8, 48, 16, 56, 24, 64, 32,
       39, 7, 47, 15, 55, 23, 63, 31,
       38, 6, 46, 14, 54, 22, 62, 30,
       37, 5, 45, 13, 53, 21, 61, 29,
       36, 4, 44, 12, 52, 20, 60, 28,
       35, 3, 43, 11, 51, 19, 59, 27,
       34, 2, 42, 10, 50, 18, 58, 26,
       33, 1, 41, 9, 49, 17, 57, 25 ]

def encrypt(pt, rkb, rk):
  pt = hex2bin(pt)
```

```
  # Initial Permutation
   pt = permute(pt, initial_perm, 64)
   print("After initial permutation", bin2hex(pt))

  # Splitting
   left = pt[0:32]
   right = pt[32:64]
   for i in range(0, 16):
      # Expansion D-box: Expanding the 32 bits data into 48 bits
      right_expanded = permute(right, exp_d, 48)

     # XOR RoundKey[i] and right_expanded
      xor_x = xor(right_expanded, rkb[i])

      # S-boxex: substituting the value from s-box table by calculating row and column
      sbox_str = ""
      for j in range(0, 8):
         row = bin2dec(int(xor_x[j * 6] + xor_x[j * 6 + 5]))
         col = bin2dec(int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2] + xor_x[j * 6 + 3] + xor_x[j * 6 + 4]))
         val = sbox[j][row][col]
         sbox_str = sbox_str + dec2bin(val)

     # Straight D-box: After substituting rearranging the bits
      sbox_str = permute(sbox_str, per, 32)

     # XOR left and sbox_str
      result = xor(left, sbox_str)
      left = result

     # Swapper
      if(i != 15):
         left, right = right, left
      print("Round ", i + 1, " ", bin2hex(left), " ", bin2hex(right), " ", rk[i])

  # Combination
   combine = left + right

  # Final permutation: final rearranging of bits to get cipher text
   cipher_text = permute(combine, final_perm, 64)
   return cipher_text

pt = "123456ABCD132536"
key = "AABB09182736CCDD"

# Key generation
# --hex to binary
key = hex2bin(key)
```

```
# --parity bit drop table
keyp = [57, 49, 41, 33, 25, 17, 9,
     1, 58, 50, 42, 34, 26, 18,
     10, 2, 59, 51, 43, 35, 27,
     19, 11, 3, 60, 52, 44, 36,
     63, 55, 47, 39, 31, 23, 15,
     7, 62, 54, 46, 38, 30, 22,
     14, 6, 61, 53, 45, 37, 29,
     21, 13, 5, 28, 20, 12, 4 ]

# getting 56 bit key from 64 bit using the parity bits
key = permute(key, keyp, 56)

# Number of bit shifts
shift_table = [1, 1, 2, 2,
          2, 2, 2, 2,
          1, 2, 2, 2,
          2, 2, 2, 1 ]

# Key- Compression Table : Compression of key from 56 bits to 48 bits
key_comp = [14, 17, 11, 24, 1, 5,
        3, 28, 15, 6, 21, 10,
        23, 19, 12, 4, 26, 8,
        16, 7, 27, 20, 13, 2,
        41, 52, 31, 37, 47, 55,
        30, 40, 51, 45, 33, 48,
        44, 49, 39, 56, 34, 53,
        46, 42, 50, 36, 29, 32 ]

# Splitting
left = key[0:28] # rkb for RoundKeys in binary
right = key[28:56] # rk for RoundKeys in hexadecimal

rkb = []
rk = []
for i in range(0, 16):
   # Shifting the bits by nth shifts by checking from shift table
   left = shift_left(left, shift_table[i])
   right = shift_left(right, shift_table[i])

  # Combination of left and right string
   combine_str = left + right

  # Compression of key from 56 to 48 bits
   round_key = permute(combine_str, key_comp, 48)

   rkb.append(round_key)
```

```
    rk.append(bin2hex(round_key))

print("Encryption")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ",cipher_text)

print("Decryption")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
print("Plain Text : ",text)
```

Output:-

Encryption

After initial permutation 14A7D67818CA18AD

Round 1 18CA18AD 5A78E394 194CD072DE8C

Round 2 5A78E394 4A1210F6 4568581ABCCE

Round 3 4A1210F6 B8089591 06EDA4ACF5B5

Round 4 B8089591 236779C2 DA2D032B6EE3

Round 5 236779C2 A15A4B87 69A629FEC913

Round 6 A15A4B87 2E8F9C65 C1948E87475E

Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0

Round 8 A9FC20A3 308BEE97 34F822F0C66D

Round 9 308BEE97 10AF9D37 84BB4473DCCC

Round 10 10AF9D37 6CA6CB20 02765708B5BF

Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5

Round 12 FF3C485F 22A5963B C2C1E96A4BF3

Round 13 22A5963B 387CCDAA 99C31397C91F

Round 14 387CCDAA BD2DD2AB 251B8BC717D0

Round 15 BD2DD2AB CF26B472 3330C5D9A36D

Round 16 19BA9212 CF26B472 181C5D75C66D

Cipher Text : C0B7A8D05F3A829C

Decryption

After initial permutation 19BA9212CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D

Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D

Round 3 387CCDAA 22A5963B 251B8BC717D0

Round 4 22A5963B FF3C485F 99C31397C91F

Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3

Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5

Round 7 10AF9D37 308BEE97 02765708B5BF

Round 8 308BEE97 A9FC20A3 84BB4473DCCC

Round 9 A9FC20A3 2E8F9C65 34F822F0C66D

Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0

Round 11 A15A4B87 236779C2 C1948E87475E

Round 12 236779C2 B8089591 69A629FEC913

Round 13 B8089591 4A1210F6 DA2D032B6EE3

Round 14 4A1210F6 5A78E394 06EDA4ACF5B5

Round 15 5A78E394 18CA18AD 4568581ABCCE

Round 16 14A7D678 18CA18AD 194CD072DE8C

Plain Text : 123456ABCD132536

Process finished with exit code 0

Name :- Kaustubh Shrikant Kabra

CLass:- TE Computer

ERP :-38

Subject :-LP2(IS) (RSA)

Code:-

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii

msg = (input("Enter Message to Encrypt and Decrypt : "))
msg = bytes(msg, 'utf-8')

keyPair = RSA.generate(3072)

pubKey = keyPair.publickey()
print(f"Public key:  (n={hex(pubKey.n)}, e={hex(pubKey.e)})")
pubKeyPEM = pubKey.exportKey()
print(pubKeyPEM.decode('ascii'))

print(f"Private key: (n={hex(pubKey.n)}, d={hex(keyPair.d)})")
privKeyPEM = keyPair.exportKey()
print(privKeyPEM.decode('ascii'))

# msg = input()
encryptor = PKCS1_OAEP.new(pubKey)
encrypted = encryptor.encrypt(msg)
print("Encrypted:", binascii.hexlify(encrypted))

decryptor = PKCS1_OAEP.new(keyPair)
decrypted = decryptor.decrypt(encrypted)
print('Decrypted:', decrypted)
```

Output:-

Enter Message to Encrypt and Decrypt : Its KK29 aka Kaustubh

Public key:

(n=0xc3ca908cbeadce58f8bf22a5711e1ebb14f68c72d38bea406618aad1371a34bc2ab378472a042b00a0ae2ce46
f9a395b69d164527719d8dbb5de6f78ef9a2b728702d84bd29f736106e6699df4a9e6dd44a696067920e71540ec3
684e37eb69d16f3d65a2431c05f56fbc7147e64b1a3682c2b22866b1426b18c9d8f449db0def0db75d0b26436313
6bbdcb829efc8fda7e51f8d6cd31aff2a630e6bfc16af9a7b2b50429b1443ae1b3617eda2b0cb27ede8501afabec62a
5f4f5ea2746f0bb59e8b42ab2c60c4362046ba8ac0aaed2c102f478b8643822090cf5919b63743c0220a128375945
1895220c8b526a67bc133424e06526824f82f83ac17efca35e948f0c301359e5f4ade7f8bdf0626a86ef2bb0eb6d77
dba747d7eb82a7b6ac53fba49d6c0fc2dc9d16f3d972fa7ffc5549b7a9c65b9ea54660739d2abcc0c201797d50b1ef

79a752d65d5042b9798d3323b2224a75be7a30c5af04c0deee77a09bfe7e7102c74135c253c445f699cfe42f4e4cde
642a437f4ef864e3d0197099d, e=0x10001)

-----BEGIN PUBLIC KEY-----

MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAw8qQjL6tzlj4vyKlcR4e

uxT2jHLTi+pAZhiq0TcaNLwqs3hHKgQrAKCuLORvmjlbadFkUncZ2Nu13m9475or

cocC2EvSn3NhBuZpnfSp5t1EppYGeSDnFUDsNoTjfradFvPWWiQxwF9W+8cUfmSx

o2gsKyKGaxQmsYydj0SdsN7w23XQsmQ2MTa73Lgp78j9p+UfjWzTGv8qYw5r/Bav

mnsrUEKbFEOuGzYX7aKwyyft6FAa+r7GKl9PXqJ0bwu1notCqyxgxDYgRrqKwKrt

LBAvR4uGQ4IgkM9ZGbY3Q8AiChKDdZRRiVIgyLUmpnvBM0JOBlJoJPgvg6wX78o1

6UjwwwE1nl9K3n+L3wYmqG7yuw621326dH1+uCp7asU/uknWwPwtydFvPZcvp//F

VJt6nGW56lRmBznSq8wMIBeX1Qse95p1LWXVBCuXmNMyOyIkp1vnowxa8EwN7ud6

Cb/n5xAsdBNcJTxEX2mc/kL05M3mQqQ39O+GTj0BlwmdAgMBAAE=

-----END PUBLIC KEY-----

Private key:
(n=0xc3ca908cbeadce58f8bf22a5711e1ebb14f68c72d38bea406618aad1371a34bc2ab378472a042b00a0ae2ce46
f9a395b69d164527719d8dbb5de6f78ef9a2b728702d84bd29f736106e6699df4a9e6dd44a696067920e71540ec3
684e37eb69d16f3d65a2431c05f56fbc7147e64b1a3682c2b22866b1426b18c9d8f449db0def0db75d0b26436313
6bbdcb829efc8fda7e51f8d6cd31aff2a630e6bfc16af9a7b2b50429b1443ae1b3617eda2b0cb27ede8501afabec62a
5f4f5ea2746f0bb59e8b42ab2c60c4362046ba8ac0aaed2c102f478b8643822090cf5919b63743c0220a128375945
1895220c8b526a67bc133424e06526824f82f83ac17efca35e948f0c301359e5f4ade7f8bdf0626a86ef2bb0eb6d77
dba747d7eb82a7b6ac53fba49d6c0fc2dc9d16f3d972fa7ffc5549b7a9c65b9ea54660739d2abcc0c201797d50b1ef
79a752d65d5042b9798d3323b2224a75be7a30c5af04c0deee77a09bfe7e7102c74135c253c445f699cfe42f4e4cde
642a437f4ef864e3d0197099d,
d=0x3826c2a112d88efaf64ffed43ae65c02e486b70e017cb99081976679fd171f73adbd6debdef17611c6835d6da0
52374befc3b5456f51f2df44400871432a507696a0eabe8827e1b3bc825d5d073ba8f1e18bf32fe5125a23becc5ff0
69bc400c3a76710dc61e9ca0db35f748f9dcd01360bf76197f3a7b7b83652414e0256781f0cac7f5b40bc87d01c90
c0aa7405540e6237092a358c1ffd73cb478a4c22ed79ba676ecbb442b0ae653f3b5dbf85f3352e852fd01d7afc69c3
20b9e84cd0a2aaa332cb57ae63658569b637daa2412c8dad3983e54d9ca7a5d433869c136093440105c316863752
e096cc8122d839adcdc0ca13a7e3007c94555703c9571bf8ada2c2634167a5666d2ded43fc9cfca128fafee93e39ad
bd54ed1320cb00d11ce5c269a3341954c9eba9120f8a15cc5ec72cdac1604d26b5fb3311659e089078f1c3d0def0a
af08124322c30e3941e6c7b5e8b519c44ed6225156fe33e40dd54999a714055a811012229f8190d1a51d7d583b78
e3fadac6e2e4d702f62f9fa333)

-----BEGIN RSA PRIVATE KEY-----

MIIG4wIBAAKCAYEAw8qQjL6tzlj4vyKlcR4euxT2jHLTi+pAZhiq0TcaNLwqs3hH

KgQrAKCuLORvmjlbadFkUncZ2Nu13m9475orcocC2EvSn3NhBuZpnfSp5t1EppYG

eSDnFUDsNoTjfradFvPWWiQxwF9W+8cUfmSxo2gsKyKGaxQmsYydj0SdsN7w23XQ

smQ2MTa73Lgp78j9p+UfjWzTGv8qYw5r/BavmnsrUEKbFEOuGzYX7aKwyyft6FAa

+r7GKl9PXqJ0bwu1notCqyxgxDYgRrqKwKrtLBAvR4uGQ4IgkM9ZGbY3Q8AiChKD

dZRRiVIgyLUmpnvBM0JOBlJoJPgvg6wX78o16UjwwwE1nl9K3n+L3wYmqG7yuw62

1326dH1+uCp7asU/uknWwPwtydFvPZcvp//FVJt6nGW56lRmBznSq8wMIBeX1Qse

95p1LWXVBCuXmNMyOyIkp1vnowxa8EwN7ud6Cb/n5xAsdBNcJTxEX2mc/kL05M3m

QqQ39O+GTj0BlwmdAgMBAAECggGAOCbCoRLYjvr2T/7UOuZcAuSGtw4BfLmQgZdm

ef0XH3OtvW3r3vF2EcaDXW2gUjdL78O1RW9R8t9EQAhxQypQdpag6r6IJ+GzvIJd

XQc7qPHhi/Mv5RJaI77MX/BpvEAMOnZxDcYenKDbNfdI+dzQE2C/dhl/Ont7g2Uk

FOAlZ4Hwysf1tAvIfQHJDAqnQFVA5iNwkqNYwf/XPLR4pMIu15umduy7RCsK5lPz

tdv4XzNS6FL9AdevxpwyC56EzQoqqjMstXrmNlhWm2N9qiQSyNrTmD5U2cp6XUM4

acE2CTRAEFwxaGN1LglsyBItg5rc3AyhOn4wB8lFVXA8lXG/itosJjQWelZm0t7U

P8nPyhKPr+6T45rb1U7RMgywDRHOXCaaM0GVTJ66kSD4oVzF7HLNrBYE0mtfszEW

WeCJB48cPQ3vCq8IEkMiww45QebHtei1GcRO1iJRVv4z5A3VSZmnFAVagRASIp+B

kNGlHX1YO3jj+trG4uTXAvYvn6MzAoHBANPyKyB+qTMnuDP2IjZFA4FBWiy2O7a/

ka5lj23/M15OE6uAxzbw9zxsDBZ3tkU87KLX68/KKs9brrs+MALlIrpDVe0z7q1N

2dLRauXRwK1KMmfgGG7sp5/1nquN7/EaTk67yFEYj+tsOXEn/RU3MX+HuAEzVoqu

MaZZ0+dhpYFFl4Ijyai9y6qktGZ/badA5iIAAunAQn/DnzoDbtstDfmqy7fD9s2x

0IXfsNBgmqiZmjZ6e6lsFngQeMMQnxwtvwKBwQDsfMpOSOdnj7fYHfDIUchyxchO

v3TkxQ/psu0hqjOOBtOqHFToEActBp9GnrGFlgVjINZPhQ1X/aQqYqs+mu0iUWwO

2OEOYXAoFwtQy77xh5jQBaGIJhSNB2+Wolv8Efz9vC76VCBSLlTcj20pzLiQLsY8

z/8b2EVo4TosLuwgBvdrteyidJfJaeHilaNxEEiIrYBOfrbU1EX820lCTLn9/tvj

SmeQBbb641N6SePl9JG3WPzdBjrI/fHq6FtmV6MCgcAbJJNjWPVASODtPqNJAfOd

9QmgWkIxeD0m8Xi55InmlOct+pMItTlkco3lvrUIDvJbNH3NoZ1z6tDox+EMLd4R

rpfthc4WQbcYqZsgDYm4Z50m8msOoZ4h/Smx3L6SyQSoTqIryJJ92uFMXYuq0OO0

6mOlO7bkkcRoAm8B3d59PLVXhE/KHWxc0TUNP1qCpewTBJ9a4jVh+WKF4nSq+w0k

ITxvr1gHJbOHwYr6VLTZzLoUKgF2RBJok+tzR8ioqi8CgcEAjH7K1d18FMuORkfr

X6cutfkIurgF40+r14RkWua6ADvQDjUMwF2dVcOkZpkrEBkDlFPS3qVGOytGF6RM

5kG2dff3gY6ZjiiXMEoYf+S7yNRtFdDymWc+OFbdlZlZmpS5P6lII5JNLWW3Jt3S

1c15LLeNMF3Fyq4e9mMwY0VxJMnevk/ziMRJ1PAhsbKCyk4JOaISIxAm4KRH/CPv

DwN0UBDUY+E1S5wJjF33nyQ8z8YPt+SXPVxRGk28Jnnqqw+PAoHAMaRAgssQ05Ba

g16ht7Zu2aLqPxhfBe8XgBG3I+kzSFUmTGfXkFxS6XA+yEwj9hDFbqFxLYjX+I5p

qdPQBVXp6X9CgY4KMokEbAaTYGslTWqATx9V5u208Xmg47Kmbfjr2RzbOqNsPc5I

yTZe2Nxd3MvE7aCJxQlN/KaUrb5NlSGgetexRsKCFALgGwF4F41PJV0R5NdfqoVd

Je9N445Lv9bGWv1hX0dHYbZryao6WuFpZde7Y7WLoxsAKbftDFCj

-----END RSA PRIVATE KEY-----

Encrypted:

b'6c810ba224b2eb60bfab6fc3c96192f640278a8e724490fab916450ef0a7bad006b90f70db810ce803352739753dc3018d0eb58a7707800808487f2004893bc3d0c5e4deb03eb587879377b6e33a2b6f44d12ac9836d1e8b2296af4c5fc97b06b090fedeeff6ea18c7ac21662cb67783f7061d914b7136d2232f03ebf6e1bc676096434ca6d883c7e8b017ea808b353933003cae78335d900b307eac919f475b903b33a1ab54a86f14ef1cf47b167cefb4391a17a91d1a9480a

3b030186872b8c2a575998231566173ff15190970d1329f99c7ce33f439580954725f7d4905c855b6a26452b47a287cc2ab1a88d29cdc5d8a80b11278ecff7e0bb1ef6aec0a9d63562be3a3132e0d89e5e1df07f3825d70681afd1951e00c66f69d29c95ff803ed298b2409066473a13362a807d35d8061c43b7574eb35960d329aad1514abaa29626b75501ff1b694ac4d1a8ceec75eafe7d68946a3f71757cc3544cb510c9236db7da53bd49240578e554fa4cd72ad92cefea895ab29ba0b3dd1da8fb44721'

Decrypted: b'Its KK29 aka Kaustubh'

Process finished with exit code 0

```
C:\Users\asus\PycharmProjects\LP2\Scripts\python.exe "C:/Users/asus/PycharmProjects/LP2(codes)/5. RSA.py"
Enter Message to Encrypt and Decrypt : Its KK29 aka Kaustubh
Public key:  (n=0xc3ca908cbeadce58f8bf22a5711e1ebb14f68c72d38bea406618aad1371a34bc2ab378472a042b00a0ae2ce46f9a395b69d164527719d8dbb5de6f78ef9a2b728702d84bd29f736106e6699df4
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAw8qQjL6tzlj4vyKlcR4e
uxT2jHLTi+pAZhiq0TcaNLwqs3hHKgQrAKCuLORvmjlbadFkUncZ2Nu13m9475or
cocC2EvSn3NhBuZpnfSp5t1EppYGeSDnFUDsNoTjfradFvPWWiQxwF9W+8cUfmSx
o2gsKyKGaxQmsYydj0SdsN7w23XQsmQ2MTa73Lgp78j9p+UfjWzTGv8qYw5r/Bav
mnsrUEKbFEOuGzYX7aKwyyft6FAa+r7GKl9PXqJ0bwu1notCqyxgxDYgRrqKwKrt
LBAvR4uGQ4IgkM9ZGbY3Q8AiChKDdZRRiVIgyLUmpnvBM0JOBlJoJPgvg6wX78o1
6UjwwwE1nl9K3n+L3wYmqG7yuw621326dH1+uCp7asU/uknWwPwtydFvPZcvp//F
VJt6nGW56lRmBznSq8wMIBeX1Qse95p1LWXVBCuXmNMyOyIkp1vnowxa8EwN7ud6
Cb/n5xAsdBNcJTxEX2mc/kL05M3mQqQ39O+GTj0BlwmdAgMBAAE=
-----END PUBLIC KEY-----
Private key: (n=0xc3ca908cbeadce58f8bf22a5711e1ebb14f68c72d38bea406618aad1371a34bc2ab378472a042b00a0ae2ce46f9a395b69d164527719d8dbb5de6f78ef9a2b728702d84bd29f736106e6699df4
-----BEGIN RSA PRIVATE KEY-----
MIIG4wIBAAKCAYEAw8qQjL6tzlj4vyKlcR4euxT2jHLTi+pAZhiq0TcaNLwqs3hH
KgQrAKCuLORvmjlbadFkUncZ2Nu13m9475orcocC2EvSn3NhBuZpnfSp5t1EppYG
eSDnFUDsNoTjfradFvPWWiQxwF9W+8cUfmSxo2gsKyKGaxQmsYydj0SdsN7w23XQ
smQ2MTa73Lgp78j9p+UfjWzTGv8qYw5r/BavmnsrUEKbFEOuGzYX7aKwyyft6FAa
+r7GKl9PXqJ0bwu1notCqyxgxDYgRrqKwKrtLBAvR4uGQ4IgkM9ZGbY3Q8AiChKD
dZRRiVIgyLUmpnvBM0JOBlJoJPgvg6wX78o16UjwwwE1nl9K3n+L3wYmqG7yuw62
1326dH1+uCp7asU/uknWwPwtydFvPZcvp//FVJt6nGW56lRmBznSq8wMIBeX1Qse
95p1LWXVBCuXmNMyOyIkp1vnowxa8EwN7ud6Cb/n5xAsdBNcJTxEX2mc/kL05M3m
QqQ39O+GTj0BlwmdAgMBAAECggGAOCbCoRLYjvr2T/7UOuZcAuSGtw4BfLmQgZdm
ef0XH3OtvW3r3vF2EcaDXW2gUjdL78O1RW9R8t9EQAhxQypQdpag6r6IJ+GzvIJd
XQc7qPHhi/Mv5RJaI77MX/BpvEAMOnZxDcYenKDbNfdI+dzQE2C/dhl/Ont7g2Uk
FOAlZ4Hwysf1tAvIfQHJDAqnQFVA5iNwkqNYwf/XPLR4pMIu15umduy7RCsK5lPz
tdv4XzNS6FL9AdevxpwyC56EzQoqqjMstXrmNlhWm2N9qiQSyNrTmD5U2cp6XUM4
```