Name :- Akash Mete

Class:- TE Computer

ERP :-52

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 :- Akash Mete

CLass:- TE Computer

ERP :-52

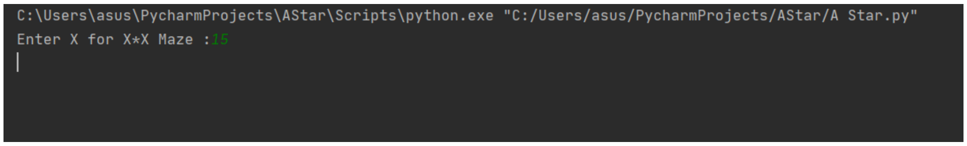
Subject :-LP2(AI) (A Star)

Code:-

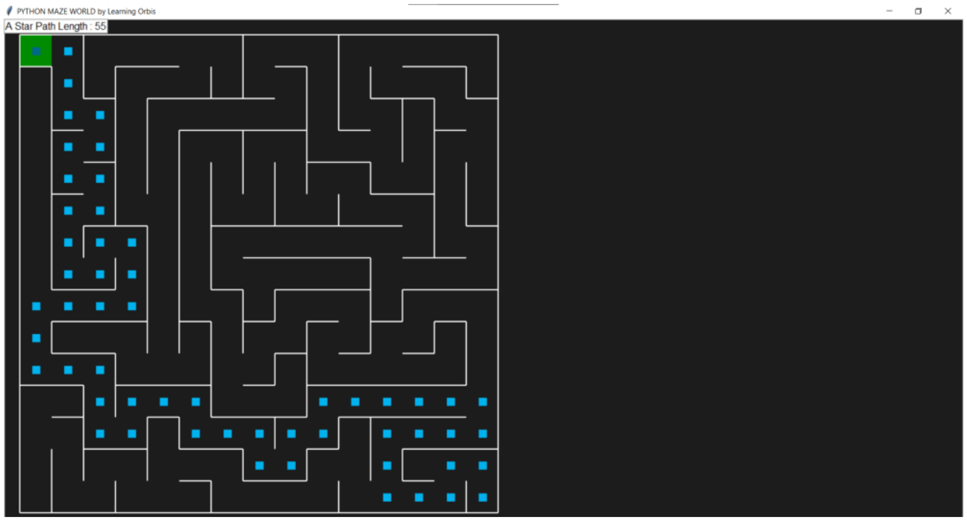
```
from pyamaze import maze,agent,textLabel
from queue import PriorityQueue
def h(cell1,cell2):
    x1,y1=cell1
    x2,y2=cell2

    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 :15



```
C:\Users\asus\PycharmProjects\AStar\Scripts\python.exe "C:/Users/asus/PycharmProjects/AStar/A Star.py"
Enter X for X*X Maze :15
```



A Star Path Length : 55

Name :- Akash Mete

CLass:- TE Computer

ERP :-52

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 :7

[Q – – – – – –]

[– – Q – – – –]

[– – – – Q – –]

[– – – – – – Q]

[– Q – – – – –]

[– – – Q – – –]

[– – – – – Q –]


[Q – – – – – –]

[– – – Q – – –]

[– – – – – – Q]

[– – Q – – – –]

[– – – – – Q –]

[– Q – – – – –]

[– – – – Q – –]
```

-
-
-
-

[– – – – – – Q]

[– – – – Q – –]

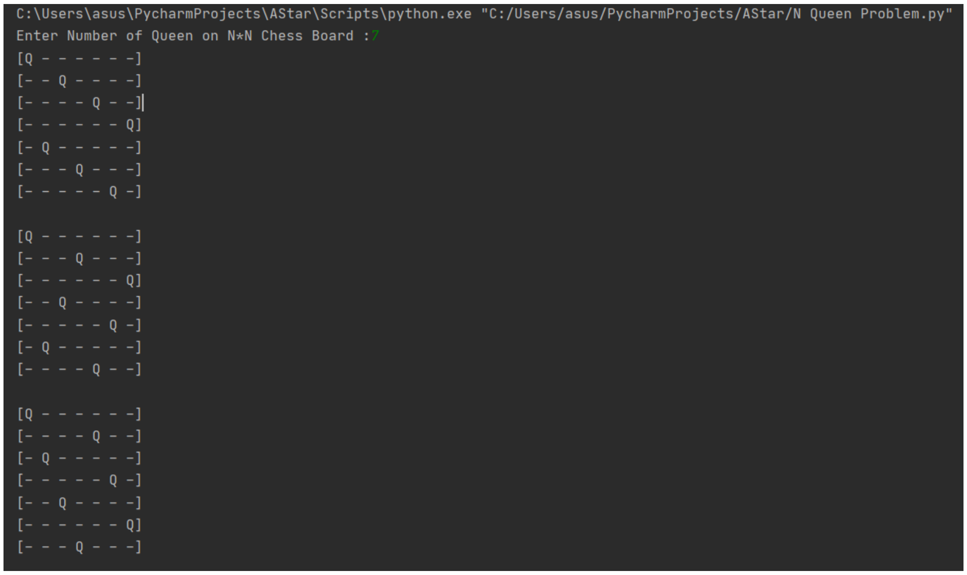
[– – Q – – – –]

[Q – – – – – –]

[– – – – – Q –]

[– – – Q – – –]

[– Q – – – – –]



```
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 :7
[Q - - - - - -]
[- - Q - - - -]
[- - - - Q - -]
[- - - - - - Q]
[- Q - - - - -]
[- - - Q - - -]
[- - - - - Q -]

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

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

Name :- Akash Mete

Class:- TE Computer

ERP :-52

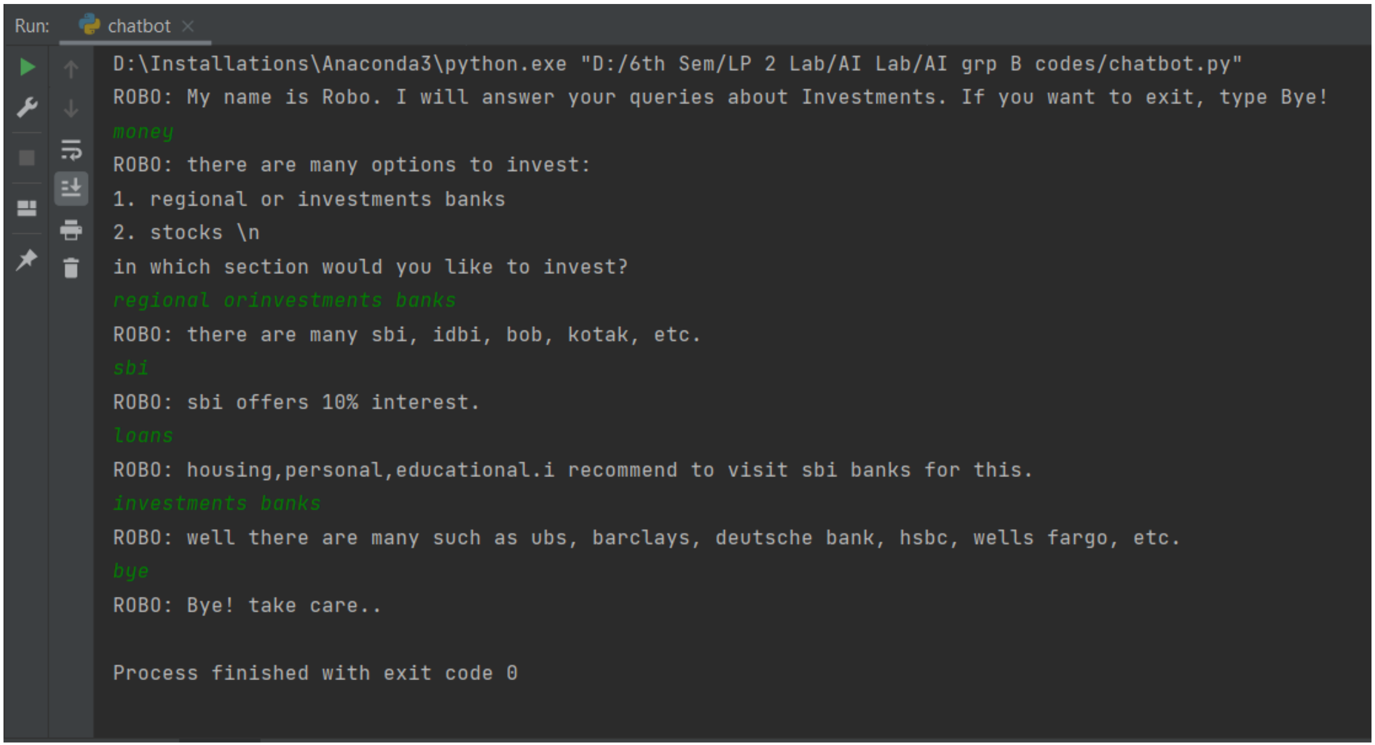
Subject :-LP2(AI) (Chatbot)

Code:-

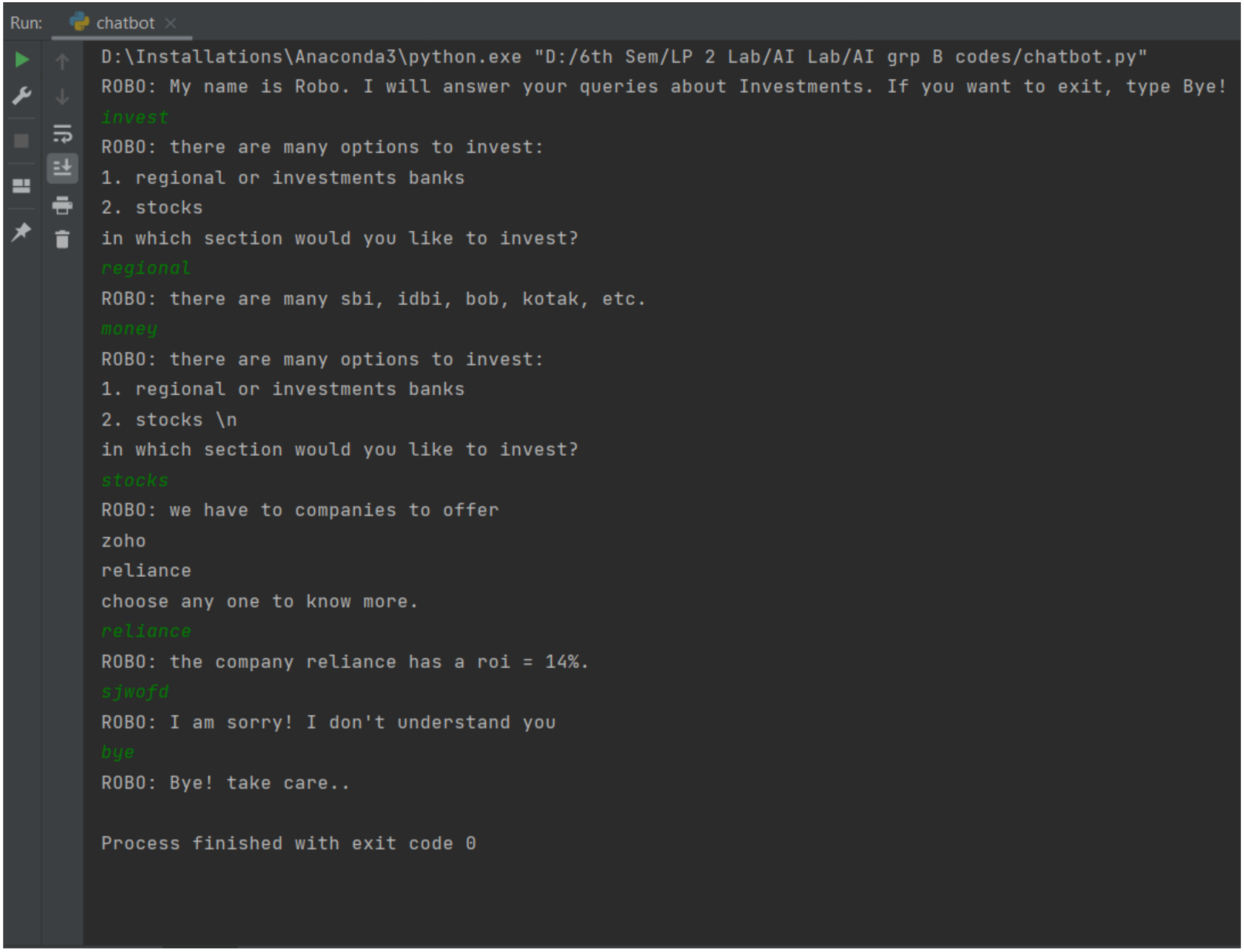
```
import io
import random
import string
import warnings
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')
import nltk
from nltk.stem import WordNetLemmatizer
# nltk.download('popular',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



```
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
```



```
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 :- Akash Mete

Class:- TE Computer

ERP :-52

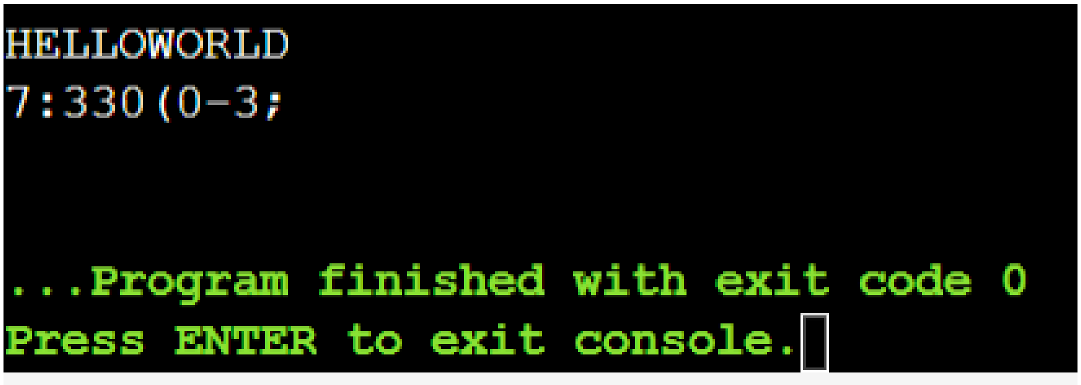
Subject :-LP2(IS) (Logical Operations)

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:-



```
HELLOWORLD
7:330(0-3;


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

Name :- Akash Mete

CLass:- TE Computer

ERP :-52

Subject :-LP2(IS) (Transposition)

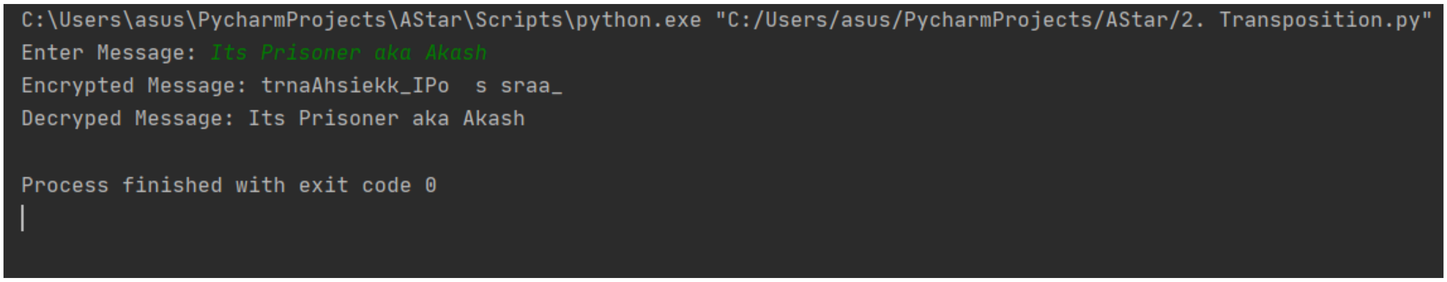
Code:-

```
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 Prisoner aka Akash

Encrypted Message: trnaAhsiekk_IPo  s sraa_

Decryped Message: Its Prisoner aka Akash


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 Prisoner aka Akash
Encrypted Message: trnaAhsiekk_IPo  s sraa_
Decryped Message: Its Prisoner aka Akash

Process finished with exit code 0
```

Name :- Akash Mete

CLass:- TE Computer

ERP :-52

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 Prisoner aka Akash

Encrypted Message:
GY/PDYq+1bKWs3D/JZ5c1PV92ChidWrk1Z218y+K1epmMNCp39hlLcpYvhq3PpDI

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

1. Encryption
```

```
2. Decryption

3. Exit

Enter your choice : 2

Enter message to decrypt:
GY/PDYq+1bKWs3D/JZ5c1PV92ChidWrk1Z218y+K1epmMNCp39hlLcpYvhq3PpDI

Decrypted Message: Its Prisoner aka Akash

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

1. Encryption

2. Decryption

3. Exit

Enter your choice : 5

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 :- Akash Mete

CLass:- TE Computer

ERP :-52

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 :- Akash Mete

CLass:- TE Computer

ERP :-52

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 Prisoner aka Akash

Public key:
(n=0x976c7495a43432362de688b2d916e5f77ce6fd8e5d6fd5b02432e150368edcd02c4c9dee5502c88bfd67ae7c
24a14f18c770ed2475eb04afb1a591ee1f4dc7412b950d580ab47f2873638936a8d2c3d8c02fbb6f8366b9b69974f
eb76d57f64d1a3ab009117cf772d6f520b4ee4e8db889087b06e1f53ef1001a9c58fc2b0d8e6871cf04b126aed009a
f1e4675cab1e6206c9e37c0e60c86f5c313bc012ac7525f9c5e38ed33cdba8f8f656a3727cb650f0c0c22d929c62f2
7423c1acd669ef7483792c2b8ea7c9bcb09822fd54eab79e924534ed33b5a6eaa84eadd79610ec60d26666ef31443
115901c6b8c331cda79be18e9a44cc5e4dfde9b81a44c21f6c686ef7ee1d228b1397a1fe2f4f5256c4978bb9e3c416
dd243e4567b2bdead2bd26ab8b098d3b71f06b1263a768f0fcadbfb1724ebcf90b2c2a95015b8d1df035262cff80e5
37252ec23f3efe260f565e3255a1605a114ddc9414463c844280075f9b57088ea4740dae624978a446870f2ed18ce
464e31041bf8f5f3f92d9c7fb, e=0x10001)
```

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

MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAl2x0laQ0MjYt5oiy2Rbl
93zm/Y5db9WwJDLhUDaO3NAsTJ3uVQLIi/1nrnwkoU8Yx3DtJHXrBK+xpZHuH03H
QSuVDVgKtH8oc2OJNqjSw9jAL7tvg2a5tpl0/rdtV/ZNGjqwCRF893LW9SC07k6N
uIkIewbh9T7xABqcWPwrDY5occ8EsSau0AmvHkZ1yrHmIGyeN8DmDIb1wxO8ASrH
Ul+cXjjtM826j49lajcny2UPDAwi2SnGLydCPBrNZp73SDeSwrjqfJvLCYIv1U6r
eekkU07TO1puqoTq3XlhDsYNJmZu8xRDEVkBxrjDMc2nm+GOmkTMXk396bgaRMIf
bGhu9+4dIosTl6H+L09SVsSXi7njxBbdJD5FZ7K96tK9JquLCY07cfBrEmOnaPD8
rb+xck68+QssKpUBW40d8DUmLP+A5TclLsI/Pv4mD1ZeMlWhYFoRTdyUFEY8hEKA
B1+bVwiOpHQNrmJJeKRGhw8u0YzkZOMQQb+PXz+S2cf7AgMBAAE=

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

Private key:

```
(n=0x976c7495a43432362de688b2d916e5f77ce6fd8e5d6fd5b02432e150368edcd02c4c9dee5502c88bfd67ae7c
24a14f18c770ed2475eb04afb1a591ee1f4dc7412b950d580ab47f2873638936a8d2c3d8c02fbb6f8366b9b69974f
eb76d57f64d1a3ab009117cf772d6f520b4ee4e8db889087b06e1f53ef1001a9c58fc2b0d8e6871cf04b126aed009a
f1e4675cab1e6206c9e37c0e60c86f5c313bc012ac7525f9c5e38ed33cdba8f8f656a3727cb650f0c0c22d929c62f2
7423c1acd669ef7483792c2b8ea7c9bcb09822fd54eab79e924534ed33b5a6eaa84eadd79610ec60d26666ef31443
115901c6b8c331cda79be18e9a44cc5e4dfde9b81a44c21f6c686ef7ee1d228b1397a1fe2f4f5256c4978bb9e3c416
dd243e4567b2bdead2bd26ab8b098d3b71f06b1263a768f0fcadbfb1724ebcf90b2c2a95015b8d1df035262cff80e5
37252ec23f3efe260f565e3255a1605a114ddc9414463c844280075f9b57088ea4740dae624978a446870f2ed18ce
464e31041bf8f5f3f92d9c7fb,
d=0x999bceb39edd1f0811e2ddf97b25877f05edd87a26148a7f226445bd2170d0ffe7c5dc25d231fdfa451926203
99334b2110b0b0659b8b80af8a3858e3cf8a1e6b2acd9d9de6ce1871f71c72d9e701b7788e98db314bc38d9212e0
0d758224b719be767d1f5de57b2354325e8102265678b5bac5cd1b6aacb15d7e8e891a6fb1c3947cfed153e05f31f
b5237946dfab3da5818a5f4a08153288f80424f6ea2143fc49f180be358c6b1d0727fcbff1abf0db7ad534e4d2992c
171f51e9be99c3d7ccac475dcbfa48a4a8328686d3329e48f40204678daf52ad4f6510b53687bea45f41b20ae2afcf
2e655d49162de29b06ec87abd9c61fe83a6d0e1e798c82fdd6216706069b111c1b2828b771aa80e5e933665f843b
cf4e8d25529fd2e99064c16140594d9f9fce03e193fdd15f3a8f5498ff14d893837a30720a81a684e2cd10ac1964fa9
30fdbd4c6afad3acb88b427b3b700deceac27e281e9fe88c74f602a54ae86757024d9ac41afc13c0d76b9b652c9852
bd134ebcb424514ccfcd5cd9)
```

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

MIIG5AIBAAKCAYEAl2x0laQ0MjYt5oiy2Rbl93zm/Y5db9WwJDLhUDaO3NAsTJ3u
VQLIi/1nrnwkoU8Yx3DtJHXrBK+xpZHuH03HQSuVDVgKtH8oc2OJNqjSw9jAL7tv
g2a5tpl0/rdtV/ZNGjqwCRF893LW9SC07k6NuIkIewbh9T7xABqcWPwrDY5occ8E
sSau0AmvHkZ1yrHmIGyeN8DmDIb1wxO8ASrHUl+cXjjtM826j49lajcny2UPDAwi
2SnGLydCPBrNZp73SDeSwrjqfJvLCYIv1U6reekkU07TO1puqoTq3XlhDsYNJmZu
8xRDEVkBxrjDMc2nm+GOmkTMXk396bgaRMIfbGhu9+4dIosTl6H+L09SVsSXi7nj
xBbdJD5FZ7K96tK9JquLCY07cfBrEmOnaPD8rb+xck68+QssKpUBW40d8DUmLP+A
5TclLsI/Pv4mD1ZeMlWhYFoRTdyUFEY8hEKAB1+bVwiOpHQNrmJJeKRGhw8u0Yzk

ZOMQQb+PXz+S2cf7AgMBAAECggGACZm86znt0fCBHi3fl7JYd/Be3YeiYUin8iZE
W9IXDQ/+fF3CXSMf36RRkmIDmTNLIRCwsGWbi4CvijhY48+KHmsqzZ2d5s4Ycfcc
ctnnAbd4jpjbMUvDjZIS4A11giS3Gb52fR9d5XsjVDJegQImVni1usXNG2qssV1+
jokab7HDlHz+0VPgXzH7UjeUbfqz2lgYpfSggVMoj4BCT26iFD/EnxgL41jGsdBy
f8v/Gr8Nt61TTk0pksFx9R6b6Zw9fMrEddy/pIpKgyhobTMp5I9AIEZ42vUq1PZR
C1Noe+pF9Bsgrir88uZV1JFi3imwbsh6vZxh/oOm0OHnmMgv3WIWcGBpsRHBsoKL
dxqoDl6TNmX4Q7z06NJVKf0umQZMFhQFlNn5/OA+GT/dFfOo9UmP8U2JODejByCo
GmhOLNEKwZZPqTD9vUxq+tOsuItCeztwDezqwn4oHp/ojHT2AqVK6GdXAk2axBr8
E8DXa5tlLJhSvRNOvLQkUUzPzVzZAoHBAMRns+b73+bkYBEAh/1VJ1nUgsc0+8E/
KE06MNAXY0WL3vPLDw/THZwZtLylkDdTOtVVLI4zay/2ChE6PWha8HRertxDjqGz
RsUweUL4s6pRMwevhsm5qmK4K6eqdUf/UuFx9xkanHFLgzduzhhAhrshcq/Nd8D7
o2Oisn3W6IgjuqK5dt3WZ40liPrDtcsSCkLE7kaOF/pe563+QR/CC+1Kl61EKz7l
3NTLuqfIwdLDVyphuCWauTBAfi2Kqa1OswKBwQDFXrKnyc35OyWsP5gdvhPGom+H
UfM8k8bK+Bt4Mm4KSCnsWI+1rvgj9LKRgubwN3WL5Ag20VG5Jvulw0HpE8ZkTNWM
8gfGA4GZrOcPDvJ7c9Gg+cexvjRXNgdpXeVUFZu4W27bmXSiWJfLJUD0ZIBH6n5z
zKi+otltzlLoaB+BxD+/euACt4sVO/eRf4/j0+3Wk7tNp1XZdPLdNRZ9H+mgcfhl
nxa5dhjXBA74ek68r8RO3G9ojFadqAlaTLtyxZkCgcEAt7ahCvCTMTBxw7WRfp/G
XTpw0dF3o/1lv0ctHZii3QzGkZhhEFZTng5Vhxf+3CFYKPCw6pqiKoykQhUOF6zo
upFOUu5GXm6JRi3fX4uu0yN87jV7iPnIrOrEuuKxLZVge0zU64B+0WLm7FUTJpBE
9omE83joCXXYEXzAJQF/JMj27Ps6eqrw1ZBEnvut8rN/MZFvqEOFnkZjw9bOJ9yk
t2NMmV/oa78rX0jp4cPhuTnLMPOTAmnFy6Kn5AWOTXQNAoHBALLJE4DWV1Sa9YdQ
nBTlJ7jZT7n+zB1lp8AYe5mn5PI/aGqF1rg3ZOP9NvyE3XlgY4Ry7dXqSuMzouUH
ON9PYHle+FsSq2P9rRpt+2gynAikY5I0cWZa68LMWG5j9ebzI/oeKQ+XtIWTRv1o
I6y+lU2P5zgyffEiR18mdQe9ujysbyqevej4Jm73wUz1hnxUb6/eZt7y49t2CsHC
4zo4/EKwutgjAkzB48JyFLWU5VoaxfLBz9Gevp9VphM8StiukQKBwD8YvVqJzntE
5d3OADTb0V0hwWKpbpkQyXMcMWIGWf5j8tmdmR30pMLHKdLvAHTSzrgpSn/2MZ7e
6PH+qvymwYLya5qGWIag7CJ092e/YA5gqv0ZAFGGjh69hzA1vxnr5i2Gl6ymZu1+
2PixJGRJDrbqYEqM6Fvy9YUT7doNfq6027cs6L/ao3KjvRS923ueKqjLSpguENty
vHQz8BirzWHjPdIeelQyqSdeTvtnWoUrbkmvD4XDghtRzUH2DacWJw==

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

Encrypted:
b'48a6bf527656ab6f0871b4ccfe437b024896cccab9d8a5201b358c9a06e04037a296f3459ed88bc857548574cef3
7952fbe148734fc0e171d177f54e35e4945020e489afe1c387614e202cc0d3a16066b28709cf1f75eeb4a1c3e5cd46

```
0895df08caf3a2a5e92d6c3cf59c46803a9059c55c9fffde8b537674107306e7a0da75ff49325a5fa5e851a024b8ebc
1e81ed921fcc5ed743ba62e81e31c4d1afcfa0c42385ec519c6b4a7e7bddb6ecd0d72ede793a09414d4cfeff368f6e1
2800ea4a180080d74af19cfb159efd4c9dc7b7fbe9a117d148efcccde7012292e02a0603b070b2b29f77fb45bc3d79
313b26c8ac3719b7c65c542e46b7108654ec86feb4256e23f3e89f7f84c48d5f7ecd5b5a11692c5e73f3fef95c956c
01457490bdf84b0828ee9b0374b599b8c56783e1a037e8f511df2297b7c8821c313bbff9d7d4189da92a4639f86b5
ff5d485b13252b686e2a1184a059df0b6fdfc59aa4357258c8c1989f1a7bd19ab1b2605239609576f2b23a22d89f1
857b2b8268bd3a9547'
```

Decrypted: b'Its Prisoner aka Akash'

```
Process finished with exit code 0
```