Question 1:- Write a program to show **Transposition Cipher**

Code 1:-

**Client Side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

def encryptor(originalMessage,numberOfRails): #Function for encrypting the originalMessage

encryptedMessage = ""

#Preparing Rail Strings

railStrings = []

for i in range(numberOfRails):

railStrings.append("")

# logic for encryption

counter = 0

for character in originalMessage:

#if character == ' ' or character == '.':

# continue

railStrings[counter]+=character

counter = (counter+1)%numberOfRails

for row in railStrings:

encryptedMessage+=(row+"$") #Using $ as A Separator for ease in decryption

return encryptedMessage

numberOfRails = 3

userInput = input("\nPlease Enter Your Desired Message : ")

encryptedMessage = encryptor(userInput,numberOfRails)

s.connect((host, port))

s.send(str.encode(encryptedMessage))

print("Client Sent Message : ")

for char in encryptedMessage:

if char == "$":

continue

print(char,end='')

print('')

s.close

**Server Side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

s.bind((host,port))

s.listen(5)

def decryptor(encryptedMessage,numberOfRails): #Function for decrypting the encryptedMessage

originalMessage = ""

#preparing Rail Strings

railStrings = []

tempString = ""

for character in encryptedMessage:

if character == "$":

#create a new RailString

railStrings.append(tempString)

tempString = ""

else:

tempString += character

#logic for decrypting

counter = 0

while True:

flag = 0

for row in railStrings:

if(counter+1 <= len(row)):

originalMessage += row[counter]

else:

flag+=1

counter += 1

if flag == numberOfRails:

break

return originalMessage

while True:

c,addr = s.accept()

print("Got Connection From : ",addr)

encryptedMessage = (c.recv(2048)).decode('utf-8')

print("Server Received Message : ")

for char in encryptedMessage:

if char == "$":

continue

print(char, end='')

print('')

numberOfRails = 3

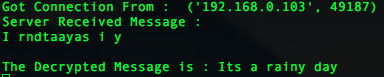
originalMessage = decryptor(encryptedMessage, numberOfRails)

print("\nThe Decrypted Message is : " + originalMessage)

c.close()

s.close

C:\Users\Aditya\Desktop\outputs\transpos\Screen Shot 2017-12-15 at 10.42.45 PM.png



Question 2:- write a program for **Play fair cipher**

Code 2:-

**Client Side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

def prepareKeyMatrix(keyString):

keyMatrix = [[0 for x in range(5)] for y in range(5)]

tempKeyMatrix = []

#logic to create the 2D key Matrix

for i in range(len(keyString)):

character = keyString[i]

if character == " ":

continue

if character in tempKeyMatrix:

continue

tempKeyMatrix.append(character)

if len(tempKeyMatrix) < 25: #we would not be able to make a 5 x 5 table

alphabets = "ABCDEFGHIKLMNOPQRSTUVWXYZ" # J and Q are omitted. I and J are Equivalent. Q is less Frequent

for alphabet in alphabets:

if alphabet not in tempKeyMatrix:

tempKeyMatrix.append(alphabet)

if len(tempKeyMatrix) == 25:

break

for i in range(5):

for j in range(5):

keyMatrix[i][j] = tempKeyMatrix[(5\*i)+j]

return keyMatrix

def breakIntoDigram(originalMessage):

digramList = []

#logic to create Digrams

i = 0

while i < len(originalMessage)-1:

if originalMessage[i] == " ":

i+=1

continue

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

tempString = originalMessage[i] + "X"

i+=1

elif originalMessage[i+1] == " ":

tempString = originalMessage[i]+originalMessage[i+2]

i+=3

else:

tempString = originalMessage[i] + originalMessage[i+1]

i+=2

digramList.append(tempString)

if digramList[-1][-1] != originalMessage[-1]:

tempString = originalMessage[-1] + "X"

digramList.append(tempString)

return digramList

def encryptor(originalDigramSet,keyMatrix): #Function for encrypting each digram

encryptedDigramList = []

#logic for encryption

for digram in originalDigramSet:

originalFirstCharacter = digram[0]

originalSecondCharacter = digram[1]

encodedFirstCharacter = ''

encodedSecondCharacter = ''

rowNumberForOriginalFirstCharacter = 0

columnNumberForOriginalFirstCharacter = 0

rowNumberForOriginalSecondCharacter = 0

columnNumberForOriginalSecondCharacter = 0

rowNumberForEncodedFirstCharacter = 0

columnNumberForEncodedFirstCharacter = 0

rowNumberForEncodedSecondCharacter = 0

columnNumberForEncodedSecondCharacter = 0

for i in range(5):

for j in range(5):

if keyMatrix[i][j] == originalFirstCharacter:

rowNumberForOriginalFirstCharacter = i

columnNumberForOriginalFirstCharacter = j

elif keyMatrix[i][j] == originalSecondCharacter:

rowNumberForOriginalSecondCharacter = i

columnNumberForOriginalSecondCharacter = j

if rowNumberForOriginalFirstCharacter == rowNumberForOriginalSecondCharacter:

# In Same Rows - Move to the Right, Wrapping Around the edge

rowNumberForEncodedFirstCharacter = rowNumberForOriginalFirstCharacter

rowNumberForEncodedSecondCharacter = rowNumberForOriginalSecondCharacter

columnNumberForEncodedFirstCharacter = (columnNumberForOriginalFirstCharacter+1)%5

columnNumberForEncodedSecondCharacter = (columnNumberForOriginalSecondCharacter+1)%5

elif columnNumberForOriginalFirstCharacter == columnNumberForOriginalSecondCharacter:

# In Same Columns - Move Downwards, Wrapping Around the edge

rowNumberForEncodedFirstCharacter = (rowNumberForOriginalFirstCharacter+1)%5

rowNumberForEncodedSecondCharacter = (rowNumberForOriginalSecondCharacter+1)%5

columnNumberForEncodedFirstCharacter = columnNumberForOriginalFirstCharacter

columnNumberForEncodedSecondCharacter = columnNumberForOriginalSecondCharacter

else:

#Different Rows And Columns

rowNumberForEncodedFirstCharacter = rowNumberForOriginalFirstCharacter

rowNumberForEncodedSecondCharacter = rowNumberForOriginalSecondCharacter

columnNumberForEncodedFirstCharacter = columnNumberForOriginalSecondCharacter

columnNumberForEncodedSecondCharacter = columnNumberForOriginalFirstCharacter

#Create A new digram

encodedFirstCharacter = keyMatrix[rowNumberForEncodedFirstCharacter][columnNumberForEncodedFirstCharacter]

encodedSecondCharacter = keyMatrix[rowNumberForEncodedSecondCharacter][columnNumberForEncodedSecondCharacter]

encodedDigram = encodedFirstCharacter + encodedSecondCharacter

#Add The New Digram to encryptedDigramList

encryptedDigramList.append(encodedDigram)

return encryptedDigramList

def joinDigrams(encryptedDigramList):

encryptedMessage = ""

#logic to change Digrams into message

for digram in encryptedDigramList:

encryptedMessage += digram

return encryptedMessage

key = "PLAYFAIR EXAMPLE"

keyMatrix = prepareKeyMatrix(key)

userInput = input("\nPlease Enter Your Desired Message [All Caps Only]: ")

userInput = userInput.upper()

digramList = breakIntoDigram(userInput)

encryptedDigramList = encryptor(digramList,keyMatrix)

encryptedMessage = joinDigrams(encryptedDigramList)

s.connect((host, port))

s.send(str.encode(encryptedMessage))

print("Client Sent Message : ",encryptedMessage)

s.close

**Server Side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

s.bind((host,port))

s.listen(5)

def prepareKeyMatrix(keyString):

keyMatrix = [[0 for x in range(5)] for y in range(5)]

tempKeyMatrix = []

#logic to create the 2D key Matrix

for i in range(len(keyString)):

character = keyString[i]

if character == " ":

continue

if character in tempKeyMatrix:

continue

tempKeyMatrix.append(character)

if len(tempKeyMatrix) < 25: #we would not be able to make a 5 x 5 table

alphabets = "ABCDEFGHIKLMNOPQRSTUVWXYZ" # J and Q are omitted. I and J are Equivalent. Q is less Frequent

for alphabet in alphabets:

if alphabet not in tempKeyMatrix:

tempKeyMatrix.append(alphabet)

if len(tempKeyMatrix) == 25:

break

for i in range(5):

for j in range(5):

keyMatrix[i][j] = tempKeyMatrix[(5\*i)+j]

return keyMatrix

def breakIntoDigram(originalMessage):

digramList = []

#logic to create Digrams

i = 0

while i < len(originalMessage)-1:

tempString = originalMessage[i] + originalMessage[i+1]

i+=2

digramList.append(tempString)

if digramList[-1][-1] != originalMessage[-1]:

tempString = originalMessage[-1] + "X"

digramList.append(tempString)

return digramList

def decryptor(encryptedDigramList,keyMatrix): #Function for decrypting the encryptedMessage

decryptedDigramList = []

#logic for decrypting

for digram in encryptedDigramList:

encodedFirstCharacter = digram[0]

encodedSecondCharacter = digram[1]

decodedFirstCharacter = ""

decodedSecondCharacter = ""

rowNumberForOriginalFirstCharacter = 0

columnNumberForOriginalFirstCharacter = 0

rowNumberForOriginalSecondCharacter = 0

columnNumberForOriginalSecondCharacter = 0

rowNumberForEncodedFirstCharacter = 0

columnNumberForEncodedFirstCharacter = 0

rowNumberForEncodedSecondCharacter = 0

columnNumberForEncodedSecondCharacter = 0

for i in range(5):

for j in range(5):

if keyMatrix[i][j] == encodedFirstCharacter:

rowNumberForEncodedFirstCharacter = i

columnNumberForEncodedFirstCharacter = j

elif keyMatrix[i][j] == encodedSecondCharacter:

rowNumberForEncodedSecondCharacter = i

columnNumberForEncodedSecondCharacter = j

if rowNumberForEncodedFirstCharacter == rowNumberForEncodedSecondCharacter:

columnNumberForOriginalFirstCharacter = (columnNumberForEncodedFirstCharacter - 1)%5

columnNumberForOriginalSecondCharacter = (columnNumberForEncodedSecondCharacter - 1)%5

rowNumberForOriginalFirstCharacter = rowNumberForEncodedFirstCharacter

rowNumberForOriginalSecondCharacter = rowNumberForEncodedSecondCharacter

elif columnNumberForEncodedFirstCharacter == columnNumberForEncodedSecondCharacter:

columnNumberForOriginalFirstCharacter = columnNumberForEncodedFirstCharacter

columnNumberForOriginalSecondCharacter = columnNumberForEncodedSecondCharacter

rowNumberForOriginalFirstCharacter = (rowNumberForEncodedFirstCharacter - 1)%5

rowNumberForOriginalSecondCharacter = (rowNumberForEncodedSecondCharacter - 1)%5

else:

columnNumberForOriginalFirstCharacter = columnNumberForEncodedSecondCharacter

columnNumberForOriginalSecondCharacter = columnNumberForEncodedFirstCharacter

rowNumberForOriginalFirstCharacter = rowNumberForEncodedFirstCharacter

rowNumberForOriginalSecondCharacter = rowNumberForEncodedSecondCharacter

decodedFirstCharacter = keyMatrix[rowNumberForOriginalFirstCharacter][columnNumberForOriginalFirstCharacter]

decodedSecondCharacter = keyMatrix[rowNumberForOriginalSecondCharacter][columnNumberForOriginalSecondCharacter]

decodedDigram = decodedFirstCharacter + decodedSecondCharacter

decryptedDigramList.append(decodedDigram)

return decryptedDigramList

def joinDigrams(decryptedDigramList):

originalMessage = ""

#logic to change Digrams into message

for digram in decryptedDigramList:

if digram[1] == "X":

originalMessage += digram[0]

else:

originalMessage += digram

return originalMessage

while True:

c,addr = s.accept()

print("Got Connection From : ",addr)

encryptedMessage = (c.recv(2048)).decode('utf-8')

print("Server Received Message : ",encryptedMessage)

key = "PLAYFAIR EXAMPLE"

keyMatrix = prepareKeyMatrix(key)

encryptedDigramList = breakIntoDigram(encryptedMessage)

decryptedList = decryptor(encryptedDigramList, keyMatrix)

originalMessage = joinDigrams(decryptedList)

print("\nThe Decrypted Message is : " + originalMessage)

c.close()

s.close

**C:\Users\Aditya\Desktop\outputs\playfair\Screen Shot 2017-12-15 at 10.41.09 PM.png**

**C:\Users\Aditya\Desktop\outputs\playfair\Screen Shot 2017-12-15 at 10.41.19 PM.png**

Question 3:- Write a program to show **Hill Cipher**

Code 3:-

**Client Side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

def valueOfAlphabet(alphabet):

value = ord(alphabet.upper())-ord('A')

return value

def alphabetOfValue(valueList):

text = ""

for i in valueList:

text += chr(ord('A') + i)

return text

def determinant(matrix,order):

determinantValue = 20

if(order == 2):

pass

elif(order == 3):

# detA = a11a22a33 + a21a32a13 + a31a12a23 - a11a32a23 - a31a22a13 - a21a12a33

determinantValue = (matrix[0][0]\*matrix[1][1]\*matrix[2][2]) + (matrix[1][0]\*matrix[2][1]\*matrix[0][2]) + (matrix[2][0]\*matrix[0][1]\*matrix[1][2]) - (matrix[0][0]\*matrix[2][1]\*matrix[1][2]) - (matrix[2][0]\*matrix[1][1]\*matrix[0][2]) - (matrix[1][0]\*matrix[0][1]\*matrix[2][2])

else:

#for order 4

pass

return determinantValue

def prepareAdjoint(matrix,order,modDeterminant):

if order == 3:

adjointMatrix = [[0 for i in range(3)] for j in range(3)]

adjointMatrix[0][0] = (((matrix[1][1] \* matrix[2][2]) - (matrix[1][2] \* matrix[2][1]))\*modDeterminant)%26

adjointMatrix[0][1] = (((matrix[0][2] \* matrix[2][1]) - (matrix[0][1] \* matrix[2][2]))\*modDeterminant)%26

adjointMatrix[0][2] = (((matrix[0][1] \* matrix[1][2]) - (matrix[0][2] \* matrix[1][1]))\*modDeterminant)%26

adjointMatrix[1][0] = (((matrix[1][2] \* matrix[2][0]) - (matrix[1][0] \* matrix[2][2]))\*modDeterminant)%26

adjointMatrix[1][1] = (((matrix[0][0] \* matrix[2][2]) - (matrix[0][2] \* matrix[2][0]))\*modDeterminant)%26

adjointMatrix[1][2] = (((matrix[0][2] \* matrix[1][0]) - (matrix[0][0] \* matrix[1][2]))\*modDeterminant)%26

adjointMatrix[2][0] = (((matrix[1][0] \* matrix[2][1]) - (matrix[1][1] \* matrix[2][0]))\*modDeterminant)%26

adjointMatrix[2][1] = (((matrix[0][1] \* matrix[2][0]) - (matrix[0][0] \* matrix[2][1]))\*modDeterminant)%26

adjointMatrix[2][2] = (((matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0]))\*modDeterminant)%26

return adjointMatrix

return [0,0]

def invertMatrix(matrixToBeInverted, n):

normalDeterminant = determinant(matrixToBeInverted, n)

modDeterminant = 0

while True:

if((modDeterminant\*normalDeterminant)%26 == 1):

break

modDeterminant += 1

invertedMatrix = prepareAdjoint(matrixToBeInverted, n, modDeterminant)

return invertedMatrix

def prepareTextValueList(text):

textValueList = []

for character in text:

textValueList.append(valueOfAlphabet(character))

return textValueList

def prepareInvertibleMatrix(keyValueList,n):

if len(keyValueList) != pow(n,2):

print("Invalid Key Length for given N")

return False

invertibleMatrix = [[0 for x in range(n)] for y in range(n)]

for i in range(n):

for j in range(n):

invertibleMatrix[i][j] = keyValueList[(n\*i)+j]

return invertibleMatrix

def prepareMessageVectorList(message,n):

if len(message)%n != 0:

#add Padding

paddingLength = n - (len(message) %n)

for i in range(paddingLength):

message+="X"

vectorList =[]

for i in range(0,len(message),n):

tempString = message[i:i+n]

tempVector = []

for character in tempString:

tempVector.append(valueOfAlphabet(character))

vectorList.append(tempVector)

return vectorList

def matrixMultiply(matA, matB, order):

productMatrix = [0 for i in range(order)]

for i in range(order):

product = 0

for j in range(order):

product += (matB[j] \* matA[i][j])

productMatrix[i] = product%26

return productMatrix

def prepareCodedVectorList(invertibleMatrix, keyValueList, n):

encodedVectorList = []

for eachKeyValueList in keyValueList:

encodedValueList = matrixMultiply(invertibleMatrix, eachKeyValueList, n)

encodedVectorList.append(encodedValueList)

return encodedVectorList

def vectorToMessage(vector):

newMessage = ""

for eachVector in vector:

newMessage += alphabetOfValue(eachVector)

return newMessage

def encryptor(originalMessage, key, n): #Function for encrypting the originalMessage

keyValueList = prepareTextValueList(key)

invertibleMatrix = prepareInvertibleMatrix(keyValueList, n)

messageVectorList = prepareMessageVectorList(originalMessage, n)

encodedVector = prepareCodedVectorList(invertibleMatrix, messageVectorList, n)

encryptedMessage = vectorToMessage(encodedVector)

return encryptedMessage

n = 3

while True:

key = input("Please Enter The Key \t\t: ")

key = key.upper()

if(len(key)!=9):

print("Invlaid Choice Of Key. Re-Enter..")

continue

if(determinant(prepareInvertibleMatrix(prepareTextValueList(key),3),3) != 0):

break

else:

print("Invlaid Choice Of Key. Re-Enter..")

message = input("Kindly Enter The Message \t: ")

message = message.upper()

if(len(message)%3 == 1):

message = message+"XX"

elif(len(message)%3 == 2):

message = message + "X"

else:

pass

encryptedMessage = encryptor(message, key, n)

s.connect((host, port))

s.send(str.encode(encryptedMessage))

print("Client Sent Message : ",encryptedMessage)

s.close

**Server Side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

s.bind((host,port))

s.listen(5)

def valueOfAlphabet(alphabet):

value = ord(alphabet.upper())-ord('A')

return value

def alphabetOfValue(valueList):

text = ""

for i in valueList:

text += chr(ord('A') + i)

return text

def determinant(matrix,order):

determinantValue = 20

if(order == 2):

pass

elif(order == 3):

# detA = a11a22a33 + a21a32a13 + a31a12a23 - a11a32a23 - a31a22a13 - a21a12a33

determinantValue = (matrix[0][0]\*matrix[1][1]\*matrix[2][2]) + (matrix[1][0]\*matrix[2][1]\*matrix[0][2]) + (matrix[2][0]\*matrix[0][1]\*matrix[1][2]) - (matrix[0][0]\*matrix[2][1]\*matrix[1][2]) - (matrix[2][0]\*matrix[1][1]\*matrix[0][2]) - (matrix[1][0]\*matrix[0][1]\*matrix[2][2])

else:

#for order 4

pass

return determinantValue

def prepareAdjoint(matrix,order,modDeterminant):

if order == 3:

adjointMatrix = [[0 for i in range(3)] for j in range(3)]

adjointMatrix[0][0] = (((matrix[1][1] \* matrix[2][2]) - (matrix[1][2] \* matrix[2][1]))\*modDeterminant)%26

adjointMatrix[0][1] = (((matrix[0][2] \* matrix[2][1]) - (matrix[0][1] \* matrix[2][2]))\*modDeterminant)%26

adjointMatrix[0][2] = (((matrix[0][1] \* matrix[1][2]) - (matrix[0][2] \* matrix[1][1]))\*modDeterminant)%26

adjointMatrix[1][0] = (((matrix[1][2] \* matrix[2][0]) - (matrix[1][0] \* matrix[2][2]))\*modDeterminant)%26

adjointMatrix[1][1] = (((matrix[0][0] \* matrix[2][2]) - (matrix[0][2] \* matrix[2][0]))\*modDeterminant)%26

adjointMatrix[1][2] = (((matrix[0][2] \* matrix[1][0]) - (matrix[0][0] \* matrix[1][2]))\*modDeterminant)%26

adjointMatrix[2][0] = (((matrix[1][0] \* matrix[2][1]) - (matrix[1][1] \* matrix[2][0]))\*modDeterminant)%26

adjointMatrix[2][1] = (((matrix[0][1] \* matrix[2][0]) - (matrix[0][0] \* matrix[2][1]))\*modDeterminant)%26

adjointMatrix[2][2] = (((matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0]))\*modDeterminant)%26

return adjointMatrix

return [0,0]

def invertMatrix(matrixToBeInverted, n):

normalDeterminant = determinant(matrixToBeInverted, n)

modDeterminant = 0

while True:

if((modDeterminant\*normalDeterminant)%26 == 1):

break

modDeterminant += 1

invertedMatrix = prepareAdjoint(matrixToBeInverted, n, modDeterminant)

return invertedMatrix

def prepareTextValueList(text):

textValueList = []

for character in text:

textValueList.append(valueOfAlphabet(character))

return textValueList

def prepareInvertibleMatrix(keyValueList,n):

if len(keyValueList) != pow(n,2):

print("Invalid Key Length for given N")

return False

invertibleMatrix = [[0 for x in range(n)] for y in range(n)]

for i in range(n):

for j in range(n):

invertibleMatrix[i][j] = keyValueList[(n\*i)+j]

return invertibleMatrix

def prepareMessageVectorList(message,n):

if len(message)%n != 0:

#add Padding

paddingLength = n - (len(message) %n)

for i in range(paddingLength):

message+="X"

vectorList =[]

for i in range(0,len(message),n):

tempString = message[i:i+n]

tempVector = []

for character in tempString:

tempVector.append(valueOfAlphabet(character))

vectorList.append(tempVector)

return vectorList

def matrixMultiply(matA, matB, order):

productMatrix = [0 for i in range(order)]

for i in range(order):

product = 0

for j in range(order):

product += (matB[j] \* matA[i][j])

productMatrix[i] = product%26

return productMatrix

def prepareCodedVectorList(invertibleMatrix, keyValueList, n):

encodedVectorList = []

for eachKeyValueList in keyValueList:

encodedValueList = matrixMultiply(invertibleMatrix, eachKeyValueList, n)

encodedVectorList.append(encodedValueList)

return encodedVectorList

def vectorToMessage(vector):

newMessage = ""

for eachVector in vector:

newMessage += alphabetOfValue(eachVector)

return newMessage

def decryptor(encryptedMessage, key, n): #Function for decrypting the encryptedMessage

#logic for decrypting

encryptedKeyValueList = prepareTextValueList(key)

invertibleMatrix = prepareInvertibleMatrix(encryptedKeyValueList, n)

invertedMatrix = invertMatrix(invertibleMatrix,n)

messageVectorList = prepareMessageVectorList(encryptedMessage, n)

decodedVector = prepareCodedVectorList(invertedMatrix, messageVectorList, n)

decryptedMessage = vectorToMessage(decodedVector)

return decryptedMessage

while True:

c,addr = s.accept()

print("Got Connection From : ",addr)

encryptedMessage = (c.recv(2048)).decode('utf-8')

print("Server Received Message : ",encryptedMessage)

n = 3

while True:

key = input("Please Enter The Key \t\t: ")

key = key.upper()

if (len(key) != 9):

print("Invlaid Choice Of Key. Re-Enter..")

continue

if (determinant(prepareInvertibleMatrix(prepareTextValueList(key), 3), 3) != 0):

break

else:

print("Invlaid Choice Of Key. Re-Enter..")

decryptedMessage = decryptor(encryptedMessage, key, n)

if(decryptedMessage[-2] == "X"):

decryptedMessage = decryptedMessage[:-2]

elif(decryptedMessage[-1] == "X"):

decryptedMessage = decryptedMessage[:-1]

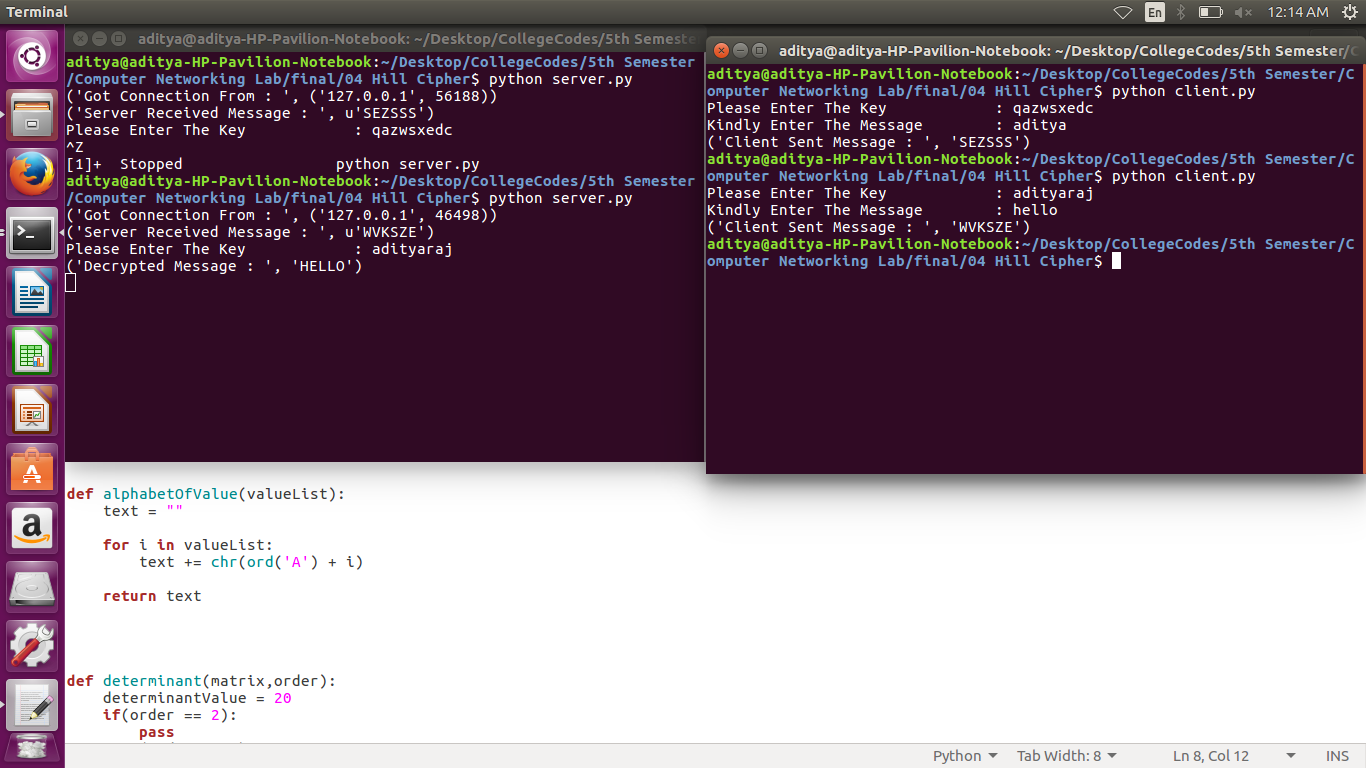
else:

pass

print("Decrypted Message : ", decryptedMessage)

c.close()

s.close



Question 4:- Write a program to show **RSA Encryption**

Code 4:-

**Client Side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

from math import sqrt

def encryptor(originalMessage,publicKey): #Function for encrypting the originalMessage

encryptedMessage = pow(originalMessage,publicKey[1]) % publicKey[0]

#logic for encryption

return encryptedMessage

def decryptor(encryptedMessage,privateKey): #Function for decrypting the encryptedMessage

originalMessage = pow(encryptedmessage,privateKey[1]) % privateKey[0]

#logic for decrypting

return originalMessage

def isPrime(n):

for i in range(2,int(sqrt(n))):

if n%i == 0:

return False

return True

def isCoprime(e,fi):

if e < fi:

for i in range(2,e+1):

if(fi%i == 0) and (e%i == 0):

return False

elif e>fi:

for i in range(2,fi+1):

if(e%i == 0) and (fi%i == 0):

return False

else:

return False

return True

def dCalculator(e,fi):

k=0

d=0

while True:

if (1+(k\*fi))%e == 0:

d = (1+(k\*fi))//e

return d

k+=1

return 0

while True:

primeNumber1 = int(input("Enter First Prime Number : "))

if isPrime(primeNumber1):

break

print("Not A Prime Number")

while True:

primeNumber2 = int(input("Enter The Second Prime Number : "))

if isPrime(primeNumber2):

break

print("Not A Prime Number")

modulus = primeNumber1 \* primeNumber2

totient = (primeNumber1 - 1) \* (primeNumber2 - 1)

while True:

publicKeyExponent = int(input("Enter The Public Key Exponent Of Your Message : "))

if isCoprime(publicKeyExponent,totient):

break

print("Invalid Key!!!")

privateKeyExponent = dCalculator(publicKeyExponent,totient)

publicKey = (modulus,publicKeyExponent)

privateKey = (modulus,privateKeyExponent)

print("Public Key : ",publicKey)

print("Private Key : ",privateKey)

userInput = int(input("\nPlease Enter Your Desired Numerical Message : "))

encryptedmessage = encryptor(userInput,publicKey)

print("\nThe Encypted Message is : "+str(encryptedmessage))

s.connect((host, port))

s.send(str.encode(encryptedmessage))

print("Client Sent Message : ",encryptedmessage)

s.close

**Server side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

s.bind((host,port))

s.listen(5)

def decryptor(encryptedMessage,privateKey): #Function for decrypting the encryptedMessage

originalMessage = pow(encryptedMessage,privateKey[1]) % privateKey[0]

#logic for decrypting

return originalMessage

while True:

c,addr = s.accept()

print("Got Connection From : ",addr)

encryptedMessage = (c.recv(2048)).decode('utf-8')

print("Server Received Message : ",encryptedMessage)

mod = input("Enter Modulus")

exp = input("Enter Private Exponent")

originalMessage = decryptor(encryptedMessage, (mod, exp))

print("\nThe Decrypted Message is : " + originalMessage)

c.close()

s.close

Question 5:- Write a program to show **Vernam Cipher**

Code 5:-

**Client side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

def valueOfAlphabet(alphabet):

return ord(alphabet)

def alphabetOfValue(value):

return chr(value)

def encryptor(originalMessage,key): #Function for encrypting the originalMessage

#logic for encryption

originalMessageList = [character for character in originalMessage]

keyList = [character for character in key]

encodedMessageValueList = []

encodedMessage = ""

for c in range(len(originalMessageList)):

encodedMessageValueList.append((valueOfAlphabet(originalMessageList[c])+valueOfAlphabet(keyList[c]))%128)

encodedMessage += alphabetOfValue(encodedMessageValueList[c])

return encodedMessage

userInput = input("\nPlease Enter Your Desired Message : ")

key = ""

while(len(key)!=len(userInput)):

key = input("\nPlease Enter A Valid Key : ")

encryptedMessage = encryptor(userInput,key)

print("\nThe Encypted Message is : "+encryptedMessage)

s.connect((host, port))

s.send(str.encode(encryptedMessage))

print("Client Sent Message : ",encryptedMessage)

s.close

**Server side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

s.bind((host,port))

s.listen(5)

def valueOfAlphabet(alphabet):

return ord(alphabet)

def alphabetOfValue(value):

return chr(value)

def decryptor(encodedMessage, key): #Function for decrypting the encryptedMessage

#logic for decrypting

encodedMessageList = [character for character in encodedMessage]

keyList = [character for character in key]

decodedMessageValueList = []

decodedMessage = ""

for c in range(len(encodedMessageList)):

decodedMessageValueList.append((valueOfAlphabet(encodedMessageList[c]) - valueOfAlphabet(keyList[c])) % 128)

decodedMessage += alphabetOfValue(decodedMessageValueList[c])

return decodedMessage

while True:

c,addr = s.accept()

print("Got Connection From : ",addr)

encryptedMessage = (c.recv(2048)).decode('utf-8')

print("Server Received Message : ",encryptedMessage)

key = ""

while (len(key) != len(encryptedMessage)):

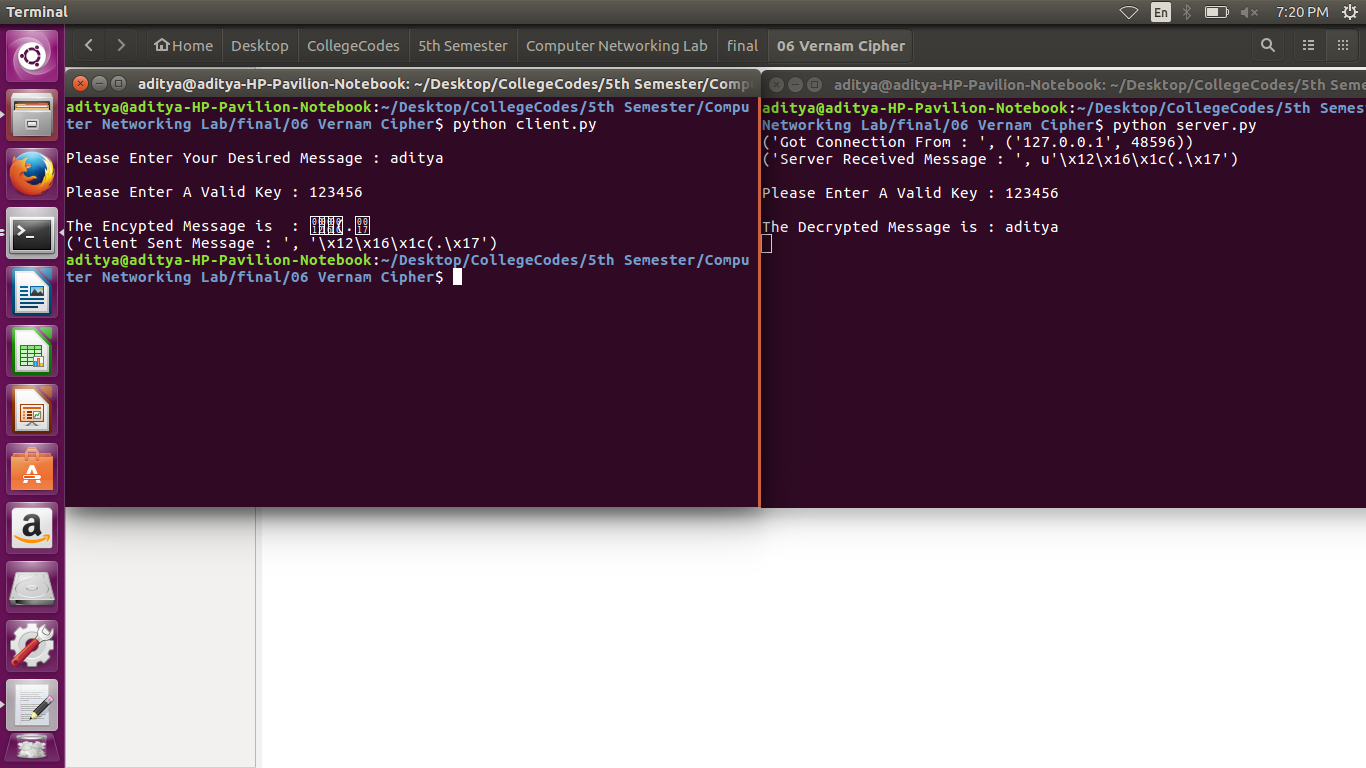
key = input("\nPlease Enter A Valid Key : ")

originalMessage = decryptor(encryptedMessage, key)

print("\nThe Decrypted Message is : " + originalMessage)

c.close()

s.close

****

Question 6:- Write a program to show **Diffie Hellman Cipher**

Code 6:-

**Client side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

p\_g\_dictionary = {

1 : [0],

2 : [1],

3 : [2],

4 : [3],

5 : [2,3],

6 : [5],

7 : [3,5],

9 : [2,5],

10: [3, 7],

11: [2, 6, 7, 8],

13: [2, 6, 7, 11],

14: [3, 5],

17: [3, 5, 6, 7, 10, 11, 12, 14],

18: [5, 11],

19: [2, 3, 10, 13, 14, 15],

22: [7, 13, 17, 19],

23: [5, 7, 10, 11, 14, 15, 17, 19, 20, 21],

25: [2, 3, 8, 12, 13, 17, 22, 23],

26: [7, 11, 15, 19],

27: [2, 5, 11, 14, 20, 23],

29: [2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27],

31: [3, 11, 12, 13, 17, 21, 22, 24],

34: [3, 5, 7, 11, 23, 27, 29, 31],

37: [2, 5, 13, 15, 17, 18, 19, 20, 22, 24, 32, 35],

38: [3, 13, 15, 21, 29, 33],

41: [6, 7, 11, 12, 13, 15, 17, 19, 22, 24, 26, 28, 29, 30, 34, 35],

43: [3, 5, 12, 18, 19, 20, 26, 28, 29, 30, 33, 34],

46: [5, 7, 11, 15, 17, 19, 21, 33, 37, 43],

47: [5, 10, 11, 13, 15, 19, 20, 22, 23, 26, 29, 30, 31, 33, 35, 38, 39, 40, 41, 43, 44, 45],

49: [3, 5, 10, 12, 17, 24, 26, 33, 38, 40, 45, 47],

50: [3, 13, 17, 23, 27, 33, 37, 47],

53: [2, 3, 5, 8, 12, 14, 18, 19, 20, 21, 22, 26, 27, 31, 32, 33, 34, 35, 39, 41, 45, 48, 50, 51],

54: [5, 11, 23, 29, 41, 47],

58: [3, 11, 15, 19, 21, 27, 31, 37, 39, 43, 47, 55],

59: [2, 6, 8, 10, 11, 13, 14, 18, 23, 24, 30, 31, 32, 33, 34, 37, 38, 39, 40, 42, 43, 44, 47, 50, 52, 54, 55, 56],

61: [2, 6, 7, 10, 17, 18, 26, 30, 31, 35, 43, 44, 51, 54, 55, 59],

62: [3, 11, 13, 17, 21, 43, 53, 55],

67: [2, 7, 11, 12, 13, 18, 20, 28, 31, 32, 34, 41, 44, 46, 48, 50, 51, 57, 61, 63],

71: [7, 11, 13, 21, 22, 28, 31, 33, 35, 42, 44, 47, 52, 53, 55, 56, 59, 61, 62, 63, 65, 67, 68, 69]

}

p = 0

g = 0

while True:

p = int(input("Enter A Valid Value of p in range [1 - 71] : "))

if p in p\_g\_dictionary.keys():

break

else:

print("Please Choose A Valid Value !!")

while True:

print("Choose a value for g from the following list : ",p\_g\_dictionary[p])

g = int(input("Enter the Value : "))

if g in p\_g\_dictionary[p]:

break

else:

print("Please Choose A Valid Value !!")

print("p = ",p)

print("g = ",g)

x = int(input("Please Choose an integer that is secret to you : "))

R1 = (g\*\*x)%p

s.connect((host, port))

s.send(str.encode(str(R1)))

R2 = int((s.recv(2048)).decode('utf-8'))

#print("Client Sent Message : ",encryptedMessage)

shared\_key = (R2\*\*x)%p

print("Shared Key : ",shared\_key)

s.send(str.encode(str(shared\_key)))

s.close

**Server side**

import socket

s = socket.socket()

host = socket.gethostname()

port = 8080

s.bind((host,port))

s.listen(5)

p\_g\_dictionary = {

1 : [0],

2 : [1],

3 : [2],

4 : [3],

5 : [2,3],

6 : [5],

7 : [3,5],

9 : [2,5],

10: [3, 7],

11: [2, 6, 7, 8],

13: [2, 6, 7, 11],

14: [3, 5],

17: [3, 5, 6, 7, 10, 11, 12, 14],

18: [5, 11],

19: [2, 3, 10, 13, 14, 15],

22: [7, 13, 17, 19],

23: [5, 7, 10, 11, 14, 15, 17, 19, 20, 21],

25: [2, 3, 8, 12, 13, 17, 22, 23],

26: [7, 11, 15, 19],

27: [2, 5, 11, 14, 20, 23],

29: [2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27],

31: [3, 11, 12, 13, 17, 21, 22, 24],

34: [3, 5, 7, 11, 23, 27, 29, 31],

37: [2, 5, 13, 15, 17, 18, 19, 20, 22, 24, 32, 35],

38: [3, 13, 15, 21, 29, 33],

41: [6, 7, 11, 12, 13, 15, 17, 19, 22, 24, 26, 28, 29, 30, 34, 35],

43: [3, 5, 12, 18, 19, 20, 26, 28, 29, 30, 33, 34],

46: [5, 7, 11, 15, 17, 19, 21, 33, 37, 43],

47: [5, 10, 11, 13, 15, 19, 20, 22, 23, 26, 29, 30, 31, 33, 35, 38, 39, 40, 41, 43, 44, 45],

49: [3, 5, 10, 12, 17, 24, 26, 33, 38, 40, 45, 47],

50: [3, 13, 17, 23, 27, 33, 37, 47],

53: [2, 3, 5, 8, 12, 14, 18, 19, 20, 21, 22, 26, 27, 31, 32, 33, 34, 35, 39, 41, 45, 48, 50, 51],

54: [5, 11, 23, 29, 41, 47],

58: [3, 11, 15, 19, 21, 27, 31, 37, 39, 43, 47, 55],

59: [2, 6, 8, 10, 11, 13, 14, 18, 23, 24, 30, 31, 32, 33, 34, 37, 38, 39, 40, 42, 43, 44, 47, 50, 52, 54, 55, 56],

61: [2, 6, 7, 10, 17, 18, 26, 30, 31, 35, 43, 44, 51, 54, 55, 59],

62: [3, 11, 13, 17, 21, 43, 53, 55],

67: [2, 7, 11, 12, 13, 18, 20, 28, 31, 32, 34, 41, 44, 46, 48, 50, 51, 57, 61, 63],

71: [7, 11, 13, 21, 22, 28, 31, 33, 35, 42, 44, 47, 52, 53, 55, 56, 59, 61, 62, 63, 65, 67, 68, 69]

}

p = 0

g = 0

while True:

p = int(input("Enter A Valid Value of p in range [1 - 71] : "))

if p in p\_g\_dictionary.keys():

break

else:

print("Please Choose A Valid Value !!")

while True:

print("Choose a value for g from the following list : ",p\_g\_dictionary[p])

g = int(input("Enter the Value : "))

if g in p\_g\_dictionary[p]:

break

else:

print("Please Choose A Valid Value !!")

print("p = ",p)

print("g = ",g)

y = int(input("Please Choose an integer that is secret to you : "))

R2 = (g\*\*y)%p

c,addr = s.accept()

c.send(str.encode(str(R2)))

R1 = int((c.recv(2048)).decode('utf-8'))

shared\_key = (R1\*\*y)%p

print("Shared Key : ",shared\_key)

othersSharedKey = int((c.recv(2048)).decode('utf-8'))

if(othersSharedKey == shared\_key):

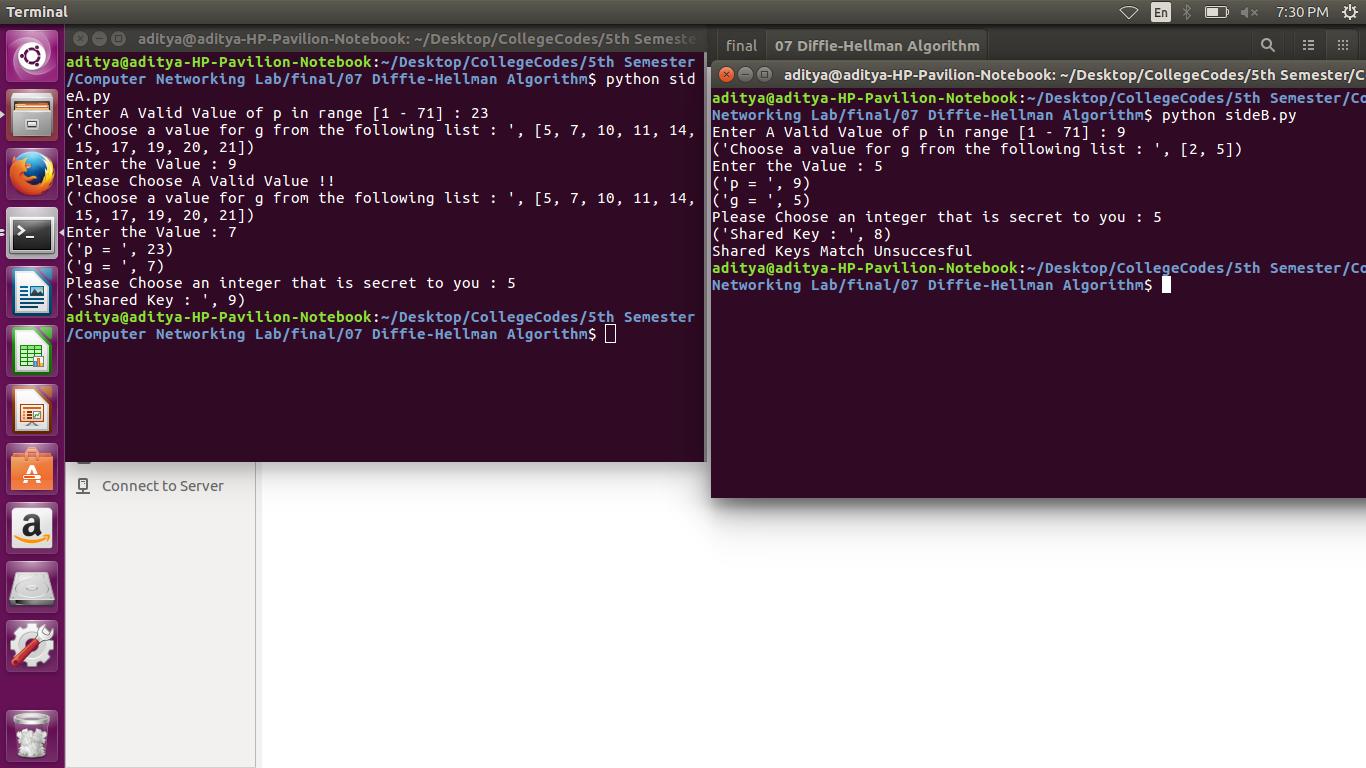
print("Shared Keys Match Succesfully")

else:

print("Shared Keys Match Unsuccesful")

c.close()

s.close



Question 7:- Write a program to show **64 Bit Des**

Code 7:-

initial\_permu\_index=[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]

inverse\_permu\_index=[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,]

expansion\_table=[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,]

permu\_table=[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]

def final\_permu(value):

inverted = ''

for i in inverse\_permu\_index:

inverted += value[i-1]

return inverted

def iniPermu(original\_message):

mess = ''

for i in initial\_permu\_index:

mess += message[i-1]

return mess

def expansion(right\_half):

expanded\_right = ''

for i in expansion\_table:

expanded\_right += right\_half[i-1]

return expanded\_right

def binary\_to\_int\_4bit(binary):

binary\_to\_int\_4b = {'0000':0,'0001':1,'0010':2,'0011':3,'0100':4,'0101':5,'0110':6,'0111':7,'1000':8,'1001':9,'1010':10,'1011':11,'1100':12,'1101':13,'1110':14,'1111':15}

x = binary\_to\_int\_4b[binary]

return x

def binary\_to\_int\_2bit(binary):

binary\_to\_int\_2b = {'00':0,'01':1,'10':2,'11':3}

return binary\_to\_int\_2b[binary]

def int\_to\_binary\_4bit(intVal):

int\_to\_binary\_4b = {0:'0000',1:'0001',2:'0010',3:'0011',4:'0100',5:'0101',6:'0110',7:'0111',8:'1000',9:'1001',10:'1010',11:'1011',12:'1100',13:'1101',14:'1110',15:'1111'}

return int\_to\_binary\_4b[intVal]

def sBox(col,row):

sBox\_array = [[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]]

return int\_to\_binary\_4bit(sBox\_array[row][col])

def compress(value\_to\_compress):

col = binary\_to\_int\_4bit(value\_to\_compress[1:5])

row = binary\_to\_int\_2bit(value\_to\_compress[0]+value\_to\_compress[-1])

#print("Col : ",col," Row : ",row)

compressed\_value = sBox(col, row)

#print(len(compressed\_value))

return compressed\_value

def func(R, K):

primary\_XOR = ''

#print("R : ",len(R))

for i in range(len(R)):

primary\_XOR += str(int(R[i])^int(K[i]))

XOR\_to\_send =''

for i in range(0,48,6):

XOR\_to\_send += compress(primary\_XOR[i:i+6])

return XOR\_to\_send

def fiestal\_round\_right\_processing(right\_half, key):

expanded\_right = expansion(right\_half)

value\_to\_be\_XORed = func(expanded\_right, key[:48])

return value\_to\_be\_XORed

def XOR\_left\_fiestal(l\_half, f\_value):

XOR\_value = ''

for i in range(len(l\_half)):

XOR\_value += str(int(l\_half[i])^int(f\_value[i]))

return XOR\_value

message = '1111110010001001110110111000111001111000011010101010101010101010'

keys = ['11100000010111010100001100110101000100110111111100110101', '10001110111111110111101110010100010111010010010011110111' , '10001001101111101110110110101100110011100111100100101101', '11101000110110011001100011000001001001000001010111101001' ,'10011000000001011011100111000001000111100100011000111000','00011111011010000110001101111100011001011100111110011101','01100001101000100100110101001111010100000001110110111011','00001100001101011000011100100010101011000000001100111101','11011010100110100011001011010001110111111111010101111010','01111001111101010011001101010011001100110100010001100110','01101101110001111000101000000000010011101100101100110000','10111101101110001000011011001010100110000011000000110111','01010100101111101001001101000101001001011010100101011000','01010100111111010101001010111101111011001010001110110010','11010101101010000011011100010010110100010100110101011101','00011100110101000001001000001010110011011001110101100101']

new\_message = message

print(message)

for i in range(16):

left\_half = new\_message[:32]

right\_half = new\_message[32:]

fiestaled\_value = fiestal\_round\_right\_processing(right\_half, keys[i])

new\_right = XOR\_left\_fiestal(left\_half, fiestaled\_value)

new\_message = right\_half+new\_right

left\_half = new\_message[:32]

right\_half = new\_message[32:]

new\_message = right\_half+left\_half

message = new\_message

print(new\_message)

# Decryption

new\_message = message

for i in range(16):

left\_half = new\_message[:32]

right\_half = new\_message[32:]

fiestaled\_value = fiestal\_round\_right\_processing(right\_half, keys[15-i])

new\_right = XOR\_left\_fiestal(left\_half, fiestaled\_value)

new\_message = right\_half + new\_right

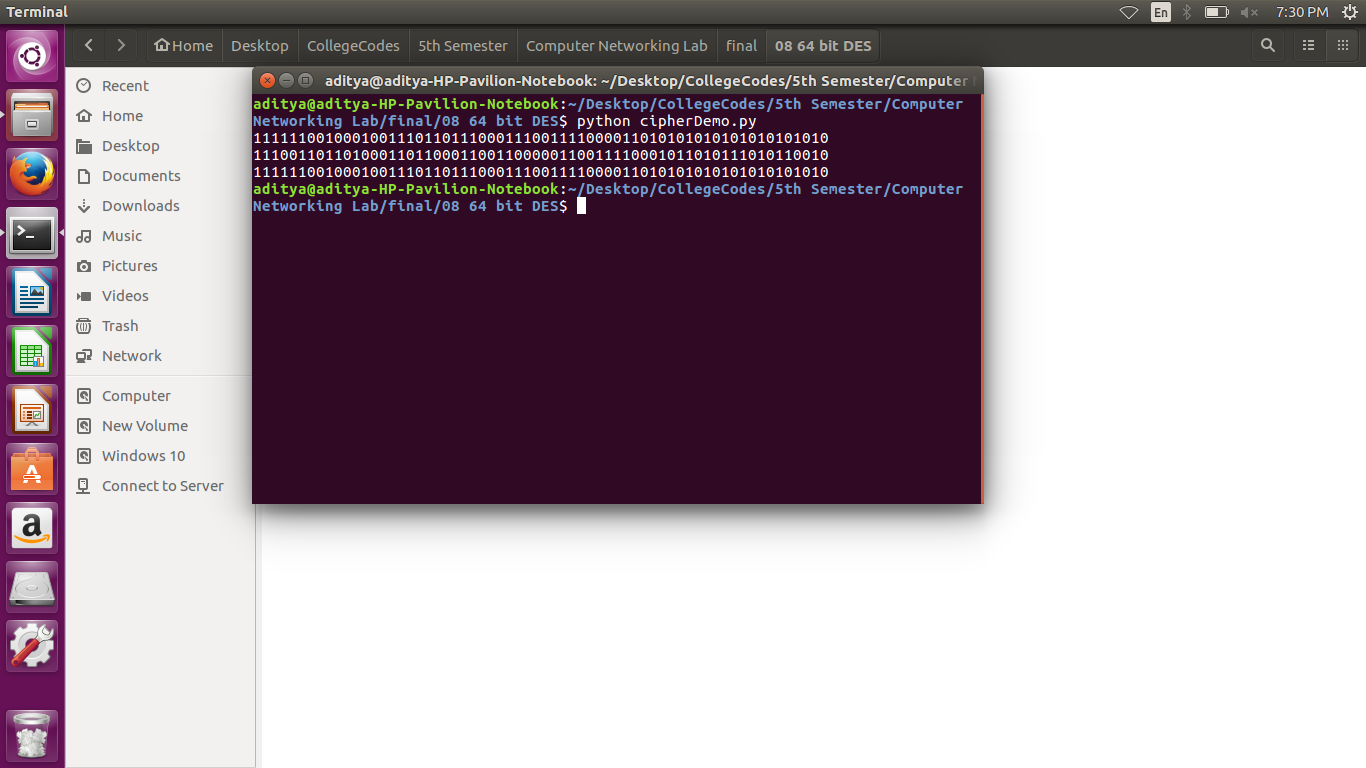
left\_half = new\_message[:32]

right\_half = new\_message[32:]

new\_message = right\_half+left\_half

#new\_message = final\_permu(new\_message)

print(new\_message)

****

Question 8:- Write a program to show **One to One / Many to Many Chat System**

Code 8:-

**Client Side**

import threading

from queue import Queue

import socket

import time

message\_q = Queue()

s = socket.socket()

host = socket.gethostname()

port = 8080

s.connect((host, port))

class Sender(threading.Thread):

def run(self):

name = input("Client name Please : ")

while True:

message = input("> ")

if message == "exit":

s.close()

return

message = name + " Says :" + message

s.send(str.encode(message))

print("Message Sent")

class Receive(threading.Thread):

def run(self):

while True:

data = s.recv(1024)

if not data:

continue

message = (data).decode('utf-8')

message = message.split('$')

for i in message:

print("> ", i)

sending\_thread = Sender()

receiving\_thread = Receive()

sending\_thread.start()

receiving\_thread.start()

**Server Side**

import threading

import socket

import time

message\_q = []

client\_list = []

s = socket.socket()

host = socket.gethostname()

port = 8080

s.bind((host, port))

s.listen(5)

class Receive(threading.Thread):

def run(self):

while True:

time.sleep(0.5)

global message\_q

sock = client\_list[-1]

data = sock.recv(1024)

message = (data).decode('utf-8')

message\_q.append(message)

print(message)

class Client\_manager(threading.Thread):

def run(self):

while True:

c, addr = s.accept()

client\_list.append(c)

rcv\_thread = Receive()

rcv\_thread.start()

#print(c)

class Sender(threading.Thread):

def run(self):

global message\_q

while True:

if(len(message\_q)!=0):

message = ''

for i in message\_q:

message += (i + "$")

for c in client\_list:

#print(c)

c.send(str.encode(message))

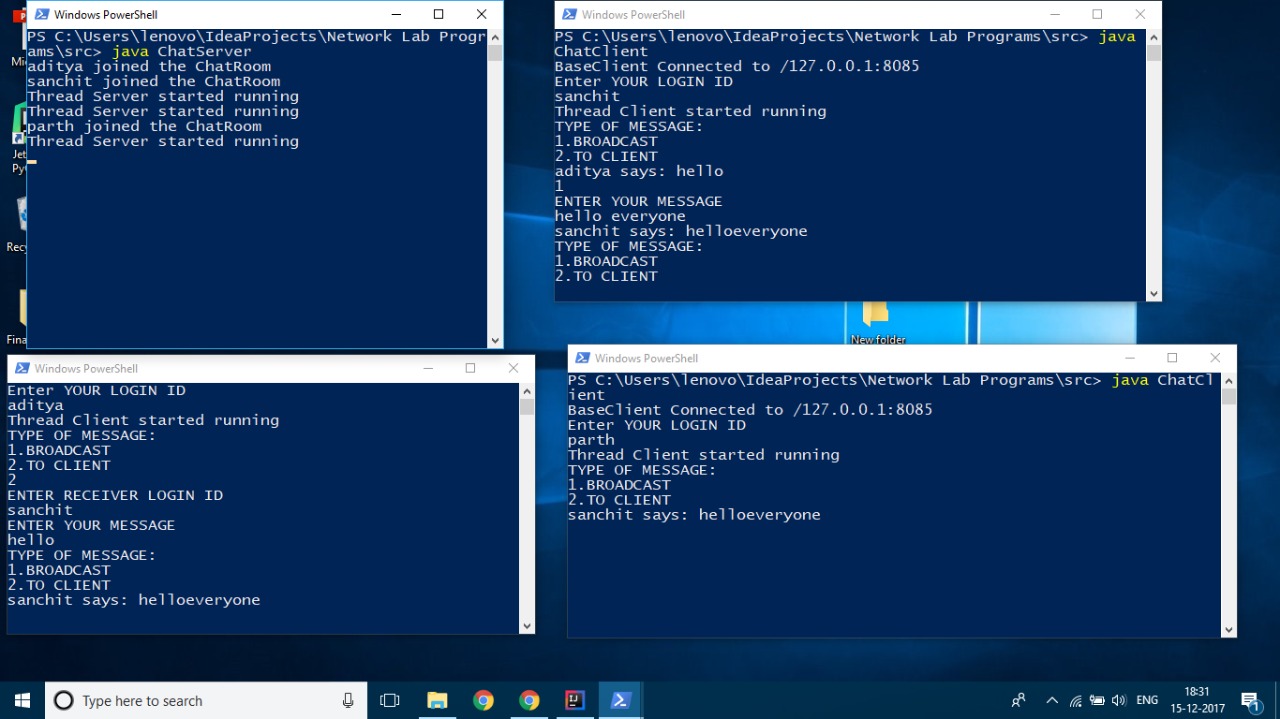
message\_q = []

sending\_thread = Sender()

port\_scanner = Client\_manager()

port\_scanner.start()

sending\_thread.start()

****

Question 9:- Write a program to **Retrieve Data From Remote Database**

At remote database the server listens for client connection.

Code 09:-

**Client Side**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.Scanner;

public class Client {

public static void main(String args[]){

int port=8085;

try {

Scanner scn=new Scanner(System.in);

Socket socket=new Socket("127.0.0.1",port);

System.out.println("BaseClient Connected to "+socket.getRemoteSocketAddress());

String key="",clientText="";

InputStream ios= socket.getInputStream();

DataInputStream dataInputStream=new DataInputStream(ios);

DataOutputStream dataOutputStream=new DataOutputStream(socket.getOutputStream());

MainThread mainThread=new MainThread(socket);

ReceiveDisplayMessages receiveDisplayMessages=new ReceiveDisplayMessages(socket);

mainThread.start();

receiveDisplayMessages.start();

} catch (UnknownHostException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

public static class MainThread extends Thread{

Scanner scanner;

Socket socket;

DataOutputStream dataOutputStream;

MainThread(Socket socket){

scanner=new Scanner(System.in);

this.socket=socket;

try {

dataOutputStream=new DataOutputStream(socket.getOutputStream());

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

super.run();int x;

String id,wherecolumnName,wherecolumnValue,columnName,columnValue,query;

while (true){

System.out.println("Choose Among the following options:");

System.out.println("1. SELECT BY ID");

System.out.println("2. SELECT BY COLUMN NAME AND VALUE");

System.out.println("3. UPDATE");

System.out.println("4. DELETE BY ID");

System.out.println("5. DELETE");

System.out.println("6. INSERT");

System.out.println("7. SELECT ALL");

x=scanner.nextInt();

try {

switch (x) {

case 1:

System.out.println("ENTER ID");

id = scanner.next();

query = "SELECTID" + " " + id;

dataOutputStream.writeUTF(query);

break;

case 2:

System.out.println("ENTER WHERE COLUMN");

wherecolumnName = scanner.next();

System.out.println("ENTER WHERE COLUMN VALUE");

wherecolumnValue = scanner.next();

query = "SELECT" + " " + wherecolumnName + " " + wherecolumnValue;

dataOutputStream.writeUTF(query);

break;

case 3:

System.out.println("ENTER COLUMN NAME");

columnName = scanner.next();

System.out.println("ENTER COLUMN VALUE");

columnValue = scanner.next();

System.out.println("ENTER WHERE COLUMN NAME");

wherecolumnName = scanner.next();

System.out.println("ENTER WHERE COLUMN VALUE");

wherecolumnValue = scanner.next();

query = "UPDATE" + " " + columnName + " " + columnValue + " " + wherecolumnName + " " + wherecolumnValue;

dataOutputStream.writeUTF(query);

break;

case 4:

System.out.println("ENTER ID");

id = scanner.next();

query = "DELETEID" + " " + id;

dataOutputStream.writeUTF(query);

break;

case 5:

System.out.println("ENTER WHERE COLUMN");

wherecolumnName = scanner.next();

System.out.println("ENTER WHERE COLUMN VALUE");

wherecolumnValue = scanner.next();

query = "DELETE" + " " + wherecolumnName + " " + wherecolumnValue;

dataOutputStream.writeUTF(query);

break;

case 6:

System.out.println("ENTER ID");

id = scanner.next();

System.out.println("ENTER Name");

String name = scanner.next();

System.out.println("ENTER EMAIL");

String email = scanner.next();

System.out.println("ENTER SEX");

String sex = scanner.next();

System.out.println("ENTER AGE");

String age = scanner.next();

query="INSERT"+" "+id+" "+name+" "+email+" "+sex+" "+age;

dataOutputStream.writeUTF(query);

break;

case 7:

query="SELECTALL";

dataOutputStream.writeUTF(query);

break;

}

}

catch (IOException e){

System.out.println(e.getMessage());

}

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public static class ReceiveDisplayMessages extends Thread{

Socket socket;

DataInputStream dataInputStream;

ReceiveDisplayMessages(Socket socket){

this.socket=socket;

try {

dataInputStream=new DataInputStream(socket.getInputStream());

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

super.run();

while (true){

try {

if(dataInputStream!=null&&dataInputStream.available()>0){

String displayMessage=dataInputStream.readUTF();

System.out.println(displayMessage);

}

// System.out.println("Sleep Starts");

Thread.sleep(2000);

// System.out.println("Sleep Ends");

} catch (IOException e) {

e.printStackTrace();

} catch (InterruptedException e) {

// e.printStackTrace();

}

}

}

}

}

**Server Side**

import javax.print.attribute.standard.MediaSize;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.sql.\*;

import java.util.ArrayList;

import java.util.Scanner;

import java.util.StringTokenizer;

public class Server {

public static final String DATABASE\_URL="jdbc:mysql://localhost:3306/sanchitdatabase";

public static final String JDBC\_DRIVER="com.mysql.jdbc.Driver";

public static final String USERNAME="root";

public static final String PASS="root";

public static void main(String args[]){

Connection connection;

Statement statement;

try {

//Registering jdbc driver

Class.forName(JDBC\_DRIVER);

// open a connection

connection = DriverManager.getConnection(DATABASE\_URL, USERNAME, PASS);

// create database;

statement = connection.createStatement();

int port=8085;

Scanner scn=new Scanner(System.in);

ServerSocket serverSocket=new ServerSocket(port);

// serverSocket.setSoTimeout(10000);

// This stops

while (true){

System.out.println("Server Says: Client Connected to Server at port "+serverSocket.getLocalPort());

Socket socket=serverSocket.accept();

System.out.println("Server Says: Client Connected "+socket.getLocalAddress());

ClientHandlingThread clientThread=new ClientHandlingThread(socket,statement);

clientThread.start();

}

} catch (IOException e) {

e.printStackTrace();

}

catch (SQLException e){

System.out.println(e.getMessage());

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

public static class ClientHandlingThread extends Thread{

Socket socket;

Statement statement;

DataInputStream dataInputStream;

DataOutputStream dataOutputStream;

String clientText="",key="";

ClientHandlingThread(Socket socket,Statement statement){

this.socket=socket;

this.statement=statement;

try {

dataInputStream=new DataInputStream(socket.getInputStream());

dataOutputStream=new DataOutputStream(socket.getOutputStream());

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

super.run();

String sendData=""; boolean done;

ArrayList<RowData> arrayList=new ArrayList<>();

while(true){

try {

if(dataInputStream.available()>0) {

clientText = dataInputStream.readUTF();

StringTokenizer stringTokenizer= new StringTokenizer(clientText);

String input=stringTokenizer.nextToken();

while (stringTokenizer.hasMoreTokens()){

if(input.equals("SELECTID")){

String id=stringTokenizer.nextToken();

arrayList=getDatabyID(statement,id);

sendData=getRowDataToString(arrayList);

if(sendData.isEmpty())

dataOutputStream.writeUTF("EMPTY OUTPUT");

else

dataOutputStream.writeUTF(sendData);

}

else if(input.equals("SELECT")){

String wherecolumnName=stringTokenizer.nextToken();

String wherecolumnValue=stringTokenizer.nextToken();

arrayList= getDatabyColumn(statement,wherecolumnName,wherecolumnValue);

sendData=getRowDataToString(arrayList);

if(sendData.isEmpty())

dataOutputStream.writeUTF("EMPTY OUTPUT");

else

dataOutputStream.writeUTF(sendData);

}

else if(input.equals("SELECTALL")){

arrayList=getData(statement);

sendData=getRowDataToString(arrayList);

if(sendData.isEmpty())

dataOutputStream.writeUTF("EMPTY OUTPUT");

else

dataOutputStream.writeUTF(sendData);

}

else if(input.equals("UPDATE")){

String columnName=stringTokenizer.nextToken();

String columnValue=stringTokenizer.nextToken();

String wherecolumnName=stringTokenizer.nextToken();

String wherecolumnValue=stringTokenizer.nextToken();

done= updateDatabyColumn(statement,columnName,columnValue,wherecolumnName,wherecolumnValue);

if(done)

dataOutputStream.writeUTF("UPDATE SUCCESSFUL");

else

dataOutputStream.writeUTF("OOPS SOMETHING WAS WRONG");

}

else if(input.equals("DELETEID")){

String id=stringTokenizer.nextToken();

done=deleteDatabyID(statement,id);

if(done)

dataOutputStream.writeUTF("DELETE SUCCESSFUL");

else

dataOutputStream.writeUTF("OOPS SOMETHING WAS WRONG");

}

else if(input.equals("DELETE")){

String wherecolumnName=stringTokenizer.nextToken();

String wherecolumnValue=stringTokenizer.nextToken();

done=deleteData(statement,wherecolumnName,wherecolumnValue);

if(done)

dataOutputStream.writeUTF("DELETE SUCCESSFUL");

else

dataOutputStream.writeUTF("OOPS SOMETHING WAS WRONG");

}

else if(input.equals("INSERT")){

String id= (stringTokenizer.nextToken());

String Name=stringTokenizer.nextToken();

String email=stringTokenizer.nextToken();

String sex=stringTokenizer.nextToken();

int age= Integer.parseInt(stringTokenizer.nextToken());

done=insertData(statement,id, Name,email,sex,age);

if(done)

dataOutputStream.writeUTF("INSERT SUCCESSFUL");

else

dataOutputStream.writeUTF("OOPS SOMETHING WAS WRONG");

}

}

}

else{

// System.out.println("Server:Client Didn't Write anything");

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

public static boolean insertData(Statement statement, String id,String Name,String email,String sex,int age){

try {

ResultSet rs= statement.executeQuery("SELECT COUNT(\*) AS rowcount from employee where id="+id+";");

id=modifyValue(id);

if(rs!=null&&rs.next()&&rs.getInt("rowcount")==0){

String query="Insert into employee values("+id+","+"'"+Name+"',"

+"'"+email+"',"

+"'"+sex+"',"

+age+");";

System.out.println(query);

statement.executeUpdate(query);

}

else{

return false;

}

} catch (SQLException e) {

e.printStackTrace();

return false;

}

return true;

}

public static boolean updateData(Statement statement, int id,String Name,String email,String sex,int age){

try {

ResultSet rs= statement.executeQuery("SELECT COUNT(\*) AS rowcount from employee where id="+id+";");

if(rs!=null&&rs.next()&&rs.getInt("rowcount")>0){

String query="Update employee set name='"+Name+"'"

+",email='"+email+"'"

+",sex='"+sex+"'"

+",age="+age+" where id="+id+";";

System.out.println(query);

statement.executeUpdate(query);

}

else{

return false;

}

} catch (SQLException e) {

e.printStackTrace();

return false;

}

return true;

}

public static boolean updateDatabyColumn(Statement statement,String columnToChange,String columnValue,String whereColumn,String whereColumnValue){

try {

whereColumnValue=modifyValue(whereColumnValue);

columnValue=modifyValue(columnValue);

String query="Update employee set "+columnToChange+"="+columnValue+" where "+whereColumn+"="+whereColumnValue+";";

System.out.println(query);

statement.executeUpdate(query);

} catch (SQLException e) {

e.printStackTrace();

return false;

}

return true;

}

public static boolean deleteDatabyID(Statement statement,String id){

try {

ResultSet rs= statement.executeQuery("SELECT COUNT(\*) AS rowcount from employee where id="+id+";");

if(rs!=null&&rs.next()&&rs.getInt("rowcount")>0){

String query="delete from employee where id="+id+";";

System.out.println(query);

statement.executeUpdate(query);

}

else{

return false;

}

} catch (SQLException e) {

e.printStackTrace();

return false;

}

return true;

}

public static boolean deleteData(Statement statement,String whereColumn,String whereValue){

try {

whereValue=modifyValue(whereValue);

String query="delete from employee where "+whereColumn+"="+whereValue+";";

System.out.println(query);

statement.executeUpdate(query);

} catch (SQLException e) {

e.printStackTrace();

return false;

}

return true;

}

public static String getRowDataToString(ArrayList<RowData> arrayList){

StringBuilder message=new StringBuilder();

for (int i = 0; i <arrayList.size() ; i++) {

message.append("ID: "+arrayList.get(i).ID+"\n");

message.append("NAME: "+arrayList.get(i).NAME+"\n");

message.append("EMAIL: "+arrayList.get(i).EMAIL+"\n");

message.append("SEX: "+arrayList.get(i).SEX+"\n");

message.append("AGE: "+arrayList.get(i).AGE+"\n");

}

return message.toString();

}

public static ArrayList<RowData> getDatabyID(Statement statement, String id){

ArrayList<RowData> data=new ArrayList<>();

try {

ResultSet resultSet= statement.executeQuery("SELECT \* from employee where id="+id);

while (resultSet!=null&&resultSet.next()){

int age=resultSet.getInt("age");

int idid=resultSet.getInt("id");

String sex=resultSet.getString("sex");

String name=resultSet.getString("name");

String email=resultSet.getString("email");

data.add(new RowData(idid,name,email,sex,age));

}

} catch (SQLException e) {

e.printStackTrace();

}

return data;

}

public static ArrayList<RowData> getDatabyColumn(Statement statement, String columnName, String value){

ArrayList<RowData> data=new ArrayList<>();

try {

ResultSet resultSet=null;

if(isInteger(value)){

resultSet= statement.executeQuery("SELECT \* from employee where "+columnName+"="+value);

}

else{

resultSet= statement.executeQuery("SELECT \* from employee where "+columnName+"="+"'"+value+"'");

}

while (resultSet!=null&&resultSet.next()){

int age=resultSet.getInt("age");

int id=resultSet.getInt("id");

String sex=resultSet.getString("sex");

String name=resultSet.getString("name");

String email=resultSet.getString("email");

data.add(new RowData(id,name,email,sex,age));

}

} catch (SQLException e) {

e.printStackTrace();

}

return data;

}

public static ArrayList<RowData> getData(Statement statement){

ArrayList<RowData> data=new ArrayList<>();

try {

ResultSet resultSet= statement.executeQuery("SELECT \* from employee;");

while (resultSet!=null&&resultSet.next()){

int age=resultSet.getInt("age");

int id=resultSet.getInt("id");

String sex=resultSet.getString("sex");

String name=resultSet.getString("name");

String email=resultSet.getString("email");

data.add(new RowData(id,name,email,sex,age));

}

} catch (SQLException e) {

e.printStackTrace();

}

return data;

}

static class RowData{

int ID;

String NAME;

String EMAIL;

String SEX;

int AGE;

public RowData(int ID, String NAME, String EMAIL, String SEX, int AGE) {

this.ID = ID;

this.NAME = NAME;

this.EMAIL = EMAIL;

this.SEX = SEX;

this.AGE = AGE;

}

}

public static boolean isInteger( String input ) {

try {

Integer.parseInt( input );

return true;

}

catch( Exception e ) {

return false;

}

}

public static String modifyValue(String value){

if(isInteger(value)){

return value;

}

else{

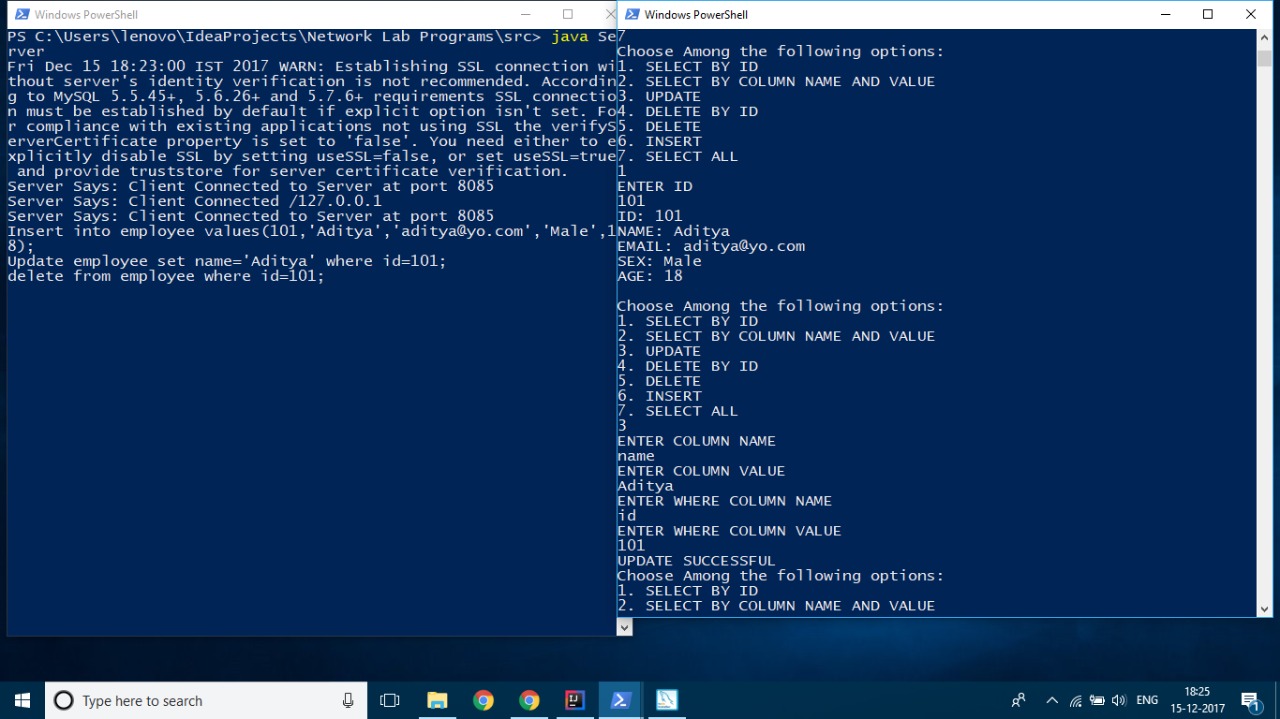
value="'"+value+"'";

}

return value;

}

}

****

Question 10:- Write a program to implement **SMTP**

Code 10:-

**Client Side**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.Scanner;

import java.util.StringTokenizer;

public class MailClient\_Client {

public static final int PORT=8085;

public static void main(String args[]){

String loginName;

DataInputStream dataInputStream;

DataOutputStream dataOutputStream;

try {

Scanner scanner = new Scanner(System.in);

Socket socket = new Socket("127.0.0.1", PORT);

System.out.println("BaseClient Connected to " + socket.getRemoteSocketAddress());

System.out.println("Enter Your Login: ");

dataInputStream=new DataInputStream(socket.getInputStream());

dataOutputStream=new DataOutputStream(socket.getOutputStream());

loginName=scanner.next();

StringTokenizer stringTokenizer;

String reciever,data,sender,receivedData;

dataOutputStream.writeUTF(loginName);

while (true){

System.out.println("\nCHOOSE ANY OF THE OPTIONS:\n" +

"1.SEND\n" +

"2.GET MY EMAILS\n" +

"3. VIEW MY MAIL\n"+

"4. GET MY SENT MAILS\n" +

"5. VIEW MY SENT MAIL\n"

);

int option=scanner.nextInt();

switch (option){

case 1:

System.out.println("Enter the receiver");

reciever=scanner.next();

System.out.println("Enter Data");

scanner.nextLine();

data=scanner.nextLine();

dataOutputStream.writeUTF("SEND "+reciever+" "+data);

break;

case 2:

dataOutputStream.writeUTF("GET");

break;

case 3:

System.out.println("Enter the sender");

sender=scanner.next();

dataOutputStream.writeUTF("VIEW "+sender);

break;

case 4:

dataOutputStream.writeUTF("GETSENT");

break;

case 5:

System.out.println("Enter the reciever");

reciever=scanner.next();

dataOutputStream.writeUTF("VIEWSENT "+reciever);

break;

}

if(option>=1&&option<=5){

while (dataInputStream.available()<=0);

receivedData=dataInputStream.readUTF();

stringTokenizer=new StringTokenizer(receivedData);

String messageType = stringTokenizer.nextToken();

if (messageType.equals("SEND")){

String mess=stringTokenizer.nextToken();

if(mess.equals("DONE"))

System.out.println("MAIL SENT SUCCESSFULLY");

else

System.out.println("RECIEVER DOESNT EXIST");

}

else if(messageType.equals("GET")){

System.out.println("========List of Emails========");

while (stringTokenizer.hasMoreTokens()){

System.out.println(stringTokenizer.nextToken());

}

}

else if(messageType.equals("GETSENT")){

System.out.println("========List of Sent Emails========");

while (stringTokenizer.hasMoreTokens()){

System.out.println(stringTokenizer.nextToken());

}

}

else if(messageType.equals("VIEW")){

while (stringTokenizer.hasMoreTokens()){

String d="",printdata="";

while (stringTokenizer.hasMoreTokens()){

d=stringTokenizer.nextToken();

if(d.equals("END"))

break;

printdata=printdata+d+" ";

}

System.out.println(printdata+"\n=======================");

}

}

else if(messageType.equals("VIEWSENT")){

while (stringTokenizer.hasMoreTokens()){

String d="",printdata="";

while (stringTokenizer.hasMoreTokens()){

d=stringTokenizer.nextToken();

if(d.equals("END"))

break;

printdata=printdata+d+" ";

}

System.out.println(printdata+"\n=======================");

}

}

}

}

} catch (UnknownHostException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Server Side**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.StringTokenizer;

public class MailClientServer {

public static final int PORT=8085;

public static HashMap<String,ArrayList<Email>>emailMap;

public static void main(String args[]){

try {

emailMap=new HashMap<String,ArrayList<Email>>();

ServerSocket serverSocket=new ServerSocket(PORT);

while (true){

Socket socket=serverSocket.accept();

ClientHandlingThread clientThread=new ClientHandlingThread(socket);

clientThread.start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

static class Email{

String sender;

String receiver;

String data;

public Email(String sender, String receiver, String data) {

this.sender = sender;

this.receiver = receiver;

this.data = data;

}

}

public static class ClientHandlingThread extends Thread{

Socket socket;

String loginName;

DataInputStream dataInputStream;

DataOutputStream dataOutputStream;

ClientHandlingThread(Socket socket){

this.socket=socket;

try {

dataInputStream = new DataInputStream(socket.getInputStream());

dataOutputStream = new DataOutputStream(socket.getOutputStream());

loginName=dataInputStream.readUTF().trim();

if(!emailMap.containsKey(loginName)){

emailMap.put(loginName,new ArrayList<Email>());

}

System.out.println(loginName+" Logged In");

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

super.run();

ArrayList<Email> arrayList;

Email email;

while (true) {

String data=new String();

try {

if (dataInputStream.available() > 0) {

String messagefromClient = dataInputStream.readUTF();

StringTokenizer stringTokenizer = new StringTokenizer(messagefromClient);

String messageType = stringTokenizer.nextToken();

if (messageType.equals("SEND")){

String to=stringTokenizer.nextToken();

if(emailMap.containsKey(to)) {

while (stringTokenizer.hasMoreTokens()) {

data += stringTokenizer.nextToken();

}

arrayList = emailMap.get(loginName);

arrayList.add(new Email(loginName, to, data));

arrayList = emailMap.get(to);

arrayList.add(new Email(loginName, to, data));

dataOutputStream.writeUTF("SEND DONE");

}

else{

dataOutputStream.writeUTF("SEND ERROR");

}

}

else if(messageType.equals("GET")){

System.out.println("GET CALLED ");

String sendEmailNames=new String();

arrayList=emailMap.get(loginName);

for (int i = 0; i <arrayList.size() ; i++) {

email=arrayList.get(i);

if(email.receiver.equals(loginName)){

sendEmailNames+=email.sender+" ";

}

}

dataOutputStream.writeUTF("GET "+sendEmailNames);

}

else if(messageType.equals("GETSENT")){

String receiverEmailNames=new String();

arrayList=emailMap.get(loginName);

for (int i = 0; i <arrayList.size() ; i++) {

email=arrayList.get(i);

if(email.sender.equals(loginName)){

receiverEmailNames+=email.receiver+" ";

}

}

dataOutputStream.writeUTF("GETSENT "+receiverEmailNames);

}

else if(messageType.equals("VIEW")){

String sender=stringTokenizer.nextToken();

String sendData=new String();

arrayList=emailMap.get(loginName);

if(emailMap.containsKey(sender)) {

for (int i = 0; i < arrayList.size(); i++) {

email = arrayList.get(i);

if (email.receiver.equals(loginName) && email.sender.equals(sender)) {

sendData += email.data + " END ";

}

}

dataOutputStream.writeUTF("VIEW " + sendData);

}

else{

dataOutputStream.writeUTF("VIEW " + "NOTHING");

}

}

else if(messageType.equals("VIEWSENT")){

String receiver=stringTokenizer.nextToken();

String sendData=new String();

arrayList=emailMap.get(loginName);

if(emailMap.containsKey(receiver)) {

for (int i = 0; i < arrayList.size(); i++) {

email = arrayList.get(i);

if (email.sender.equals(loginName) && email.receiver.equals(receiver)) {

sendData += email.data + " END ";

}

}

dataOutputStream.writeUTF("VIEWSENT " + sendData);

}

else{

dataOutputStream.writeUTF("VIEWSENT " + "NOTHING");

}

}

}

}

catch(IOException e){

e.printStackTrace();

}

}

}

}

}