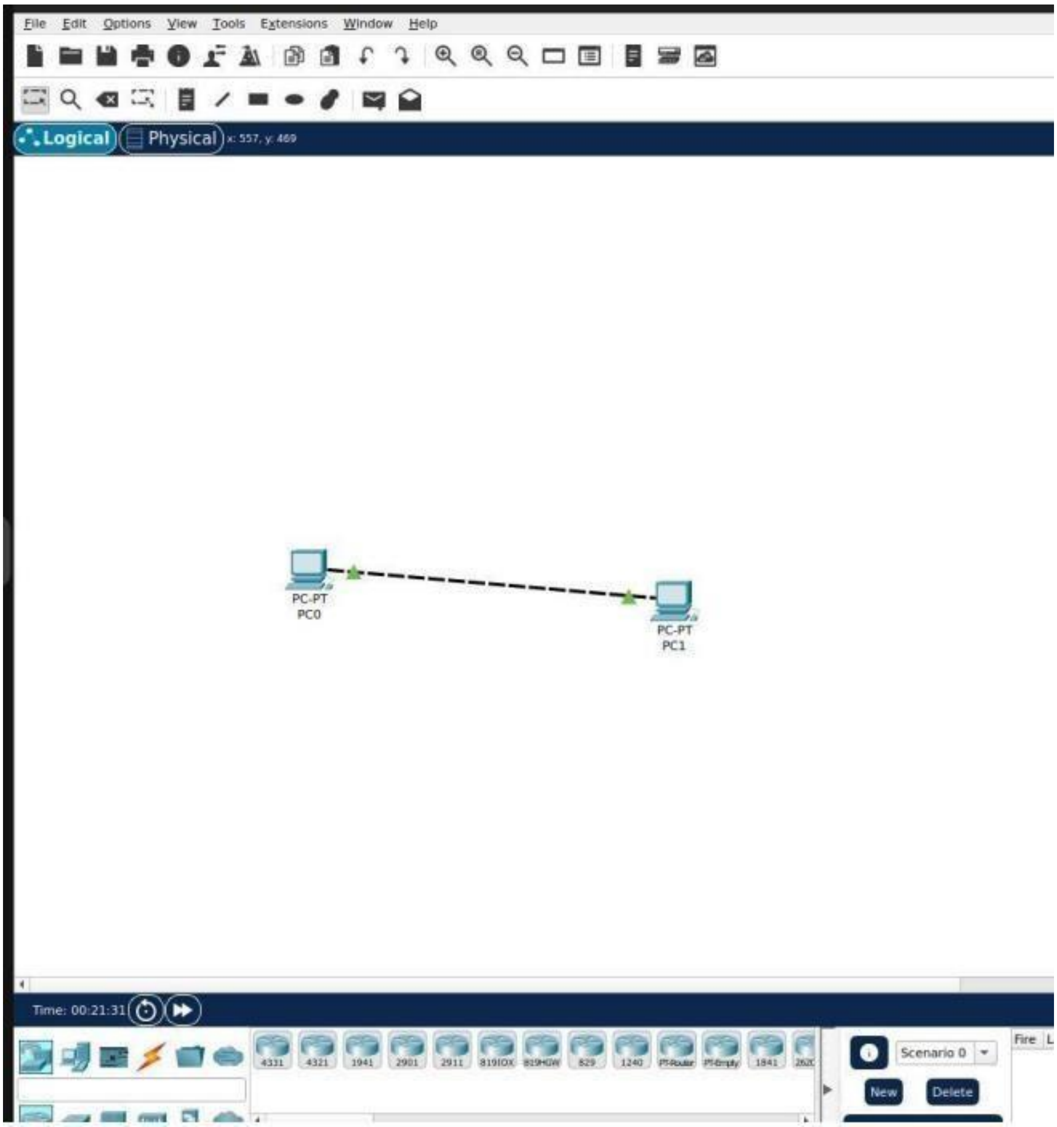
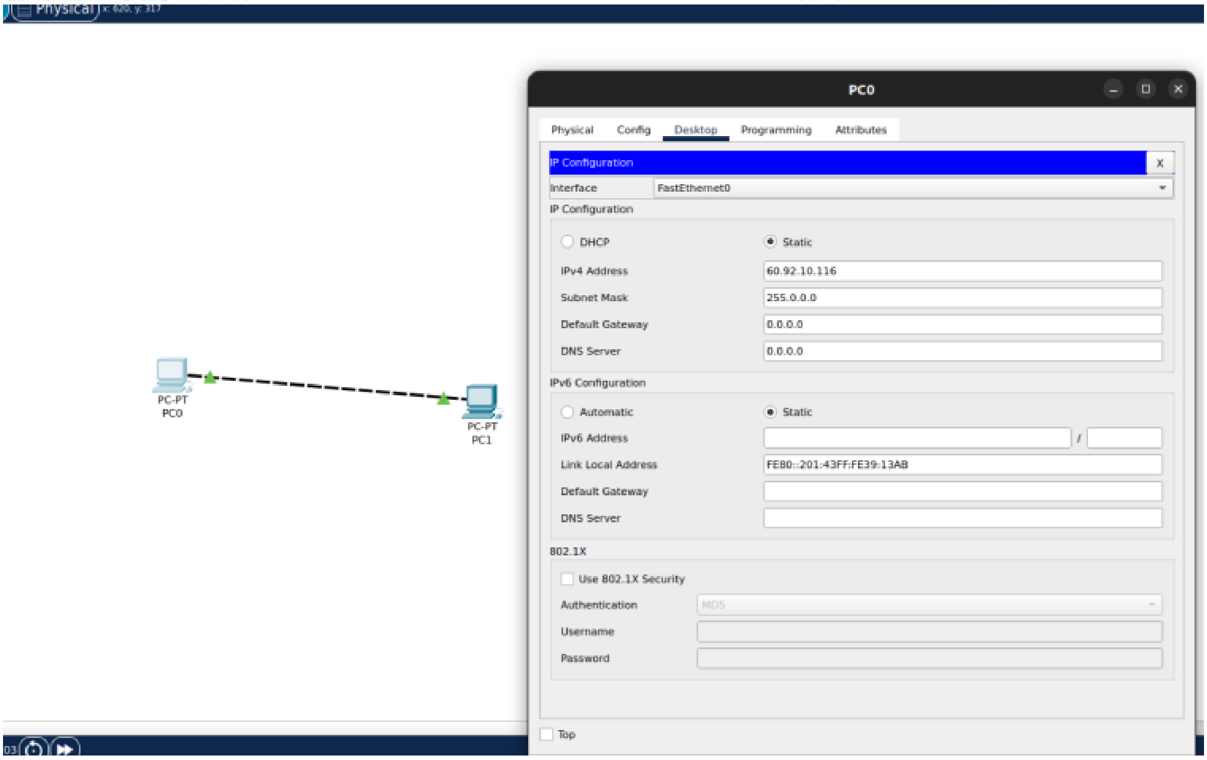
LAN network using packet tracer –


File Edit Options View Tools Extensions Window Help
Logical
Physical
x: 557, y: 469
PC-PT
PC0
PC-PT
PC1
Time: 00:21:31
4331 4321 1941 2901 2911 819IOX 819HGW 829 1240 PT-Router PT-Empty 1841 2620
Scenario 0
New
Delete
Fire



IP Configuration

Interface: FastEthernet0

IP Configuration

○ DHCP ● Static

IPv4 Address: 60.92.10.116

Subnet Mask: 255.0.0.0

Default Gateway: 0.0.0.0

DNS Server: 0.0.0.0

IPv6 Configuration

○ Automatic ● Static

IPv6 Address: /

Link Local Address: FE80::201:43FF:FE39:13AB

Default Gateway:

DNS Server:

802.1X

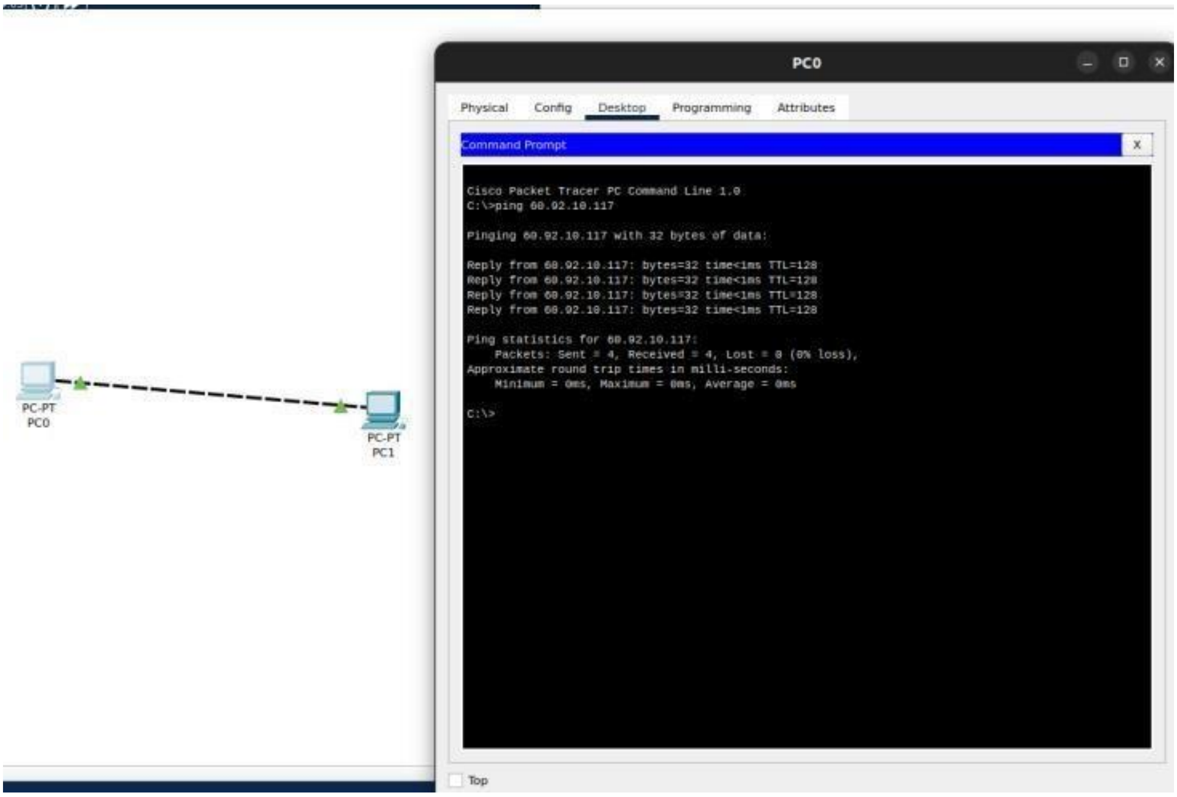
□ Use 802.1X Security

Authentication: MD5

Username:

Password:

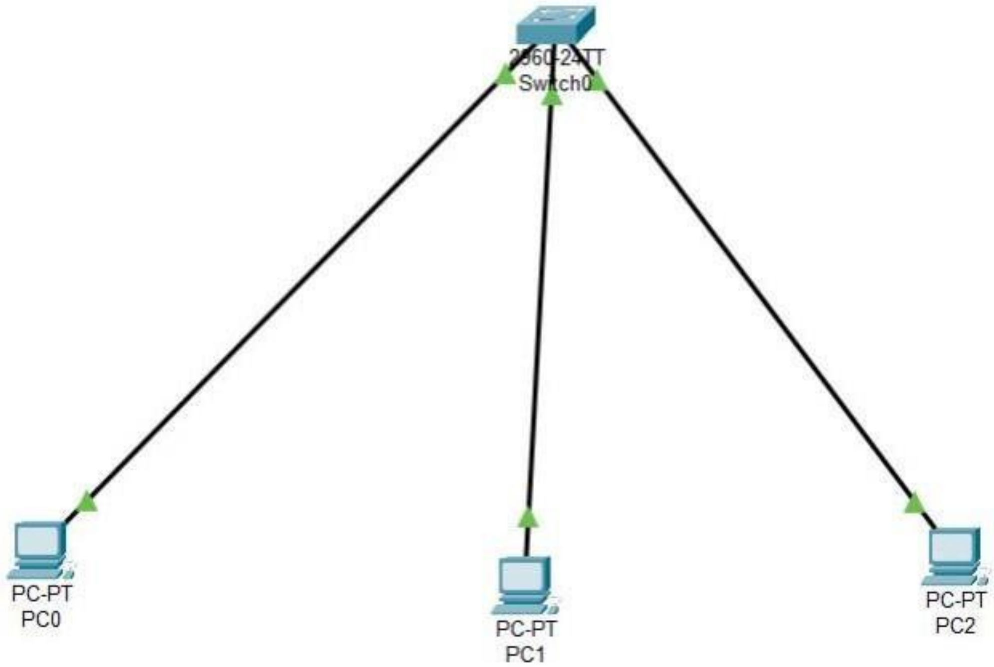
□ Top

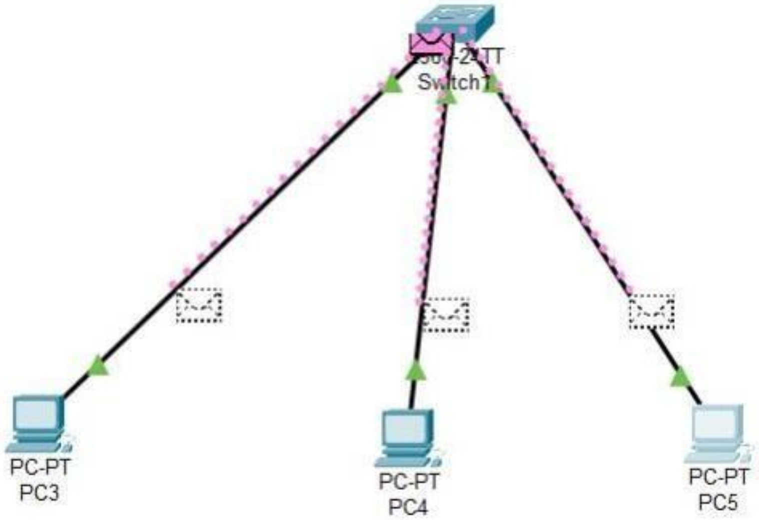


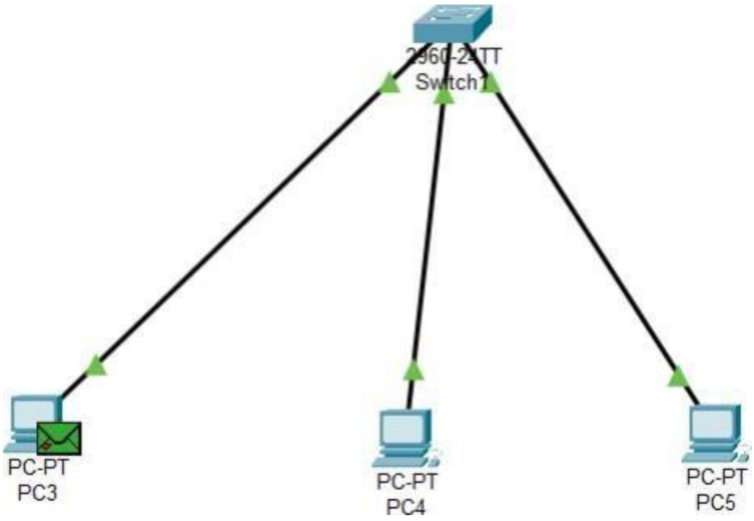
```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 60.92.10.117

Pinging 60.92.10.117 with 32 bytes of data:

Reply from 60.92.10.117: bytes=32 time<1ms TTL=128
Reply from 60.92.10.117: bytes=32 time<1ms TTL=128
Reply from 60.92.10.117: bytes=32 time<1ms TTL=128
Reply from 60.92.10.117: bytes=32 time<1ms TTL=128

Ping statistics for 60.92.10.117:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>
```

Steps to perform the experiment –

1. Launch Cisco Packet Tracer application
2. Select two computers and drag them on the space
3. Select the Copper crossover cable and connect it with the two computers
4. Select the first computer and then go to Desktop, then go into the IP Configuration and in the IP address enter an address ranging from 0.0.0.0 to 255.255.255.0 and press Enter
5. Similarly enter the IP address for the second computer and press Enter
6. Now select the first computer and in Desktop, go to Command Prompt and enter the following command: ping <IP address of the second computer> and press Enter.
7. It will show you how many packets are lost or not during the message transfer.

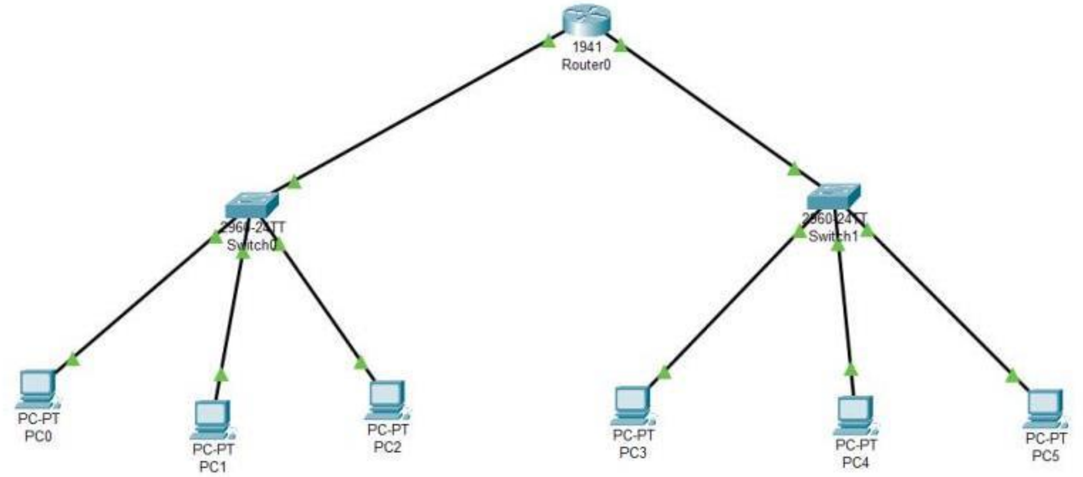

2960-24TT
Switch0
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2

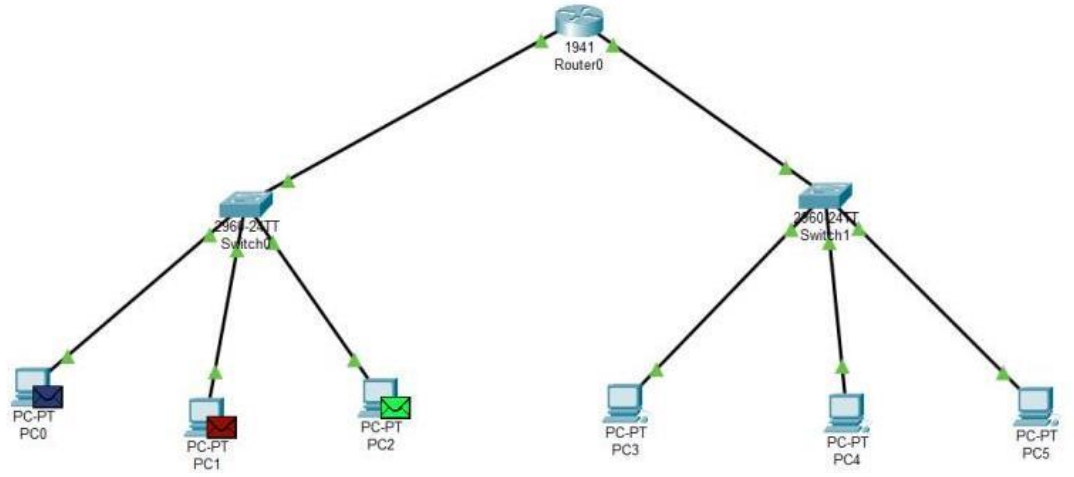

2960-24TT
Switch1
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5

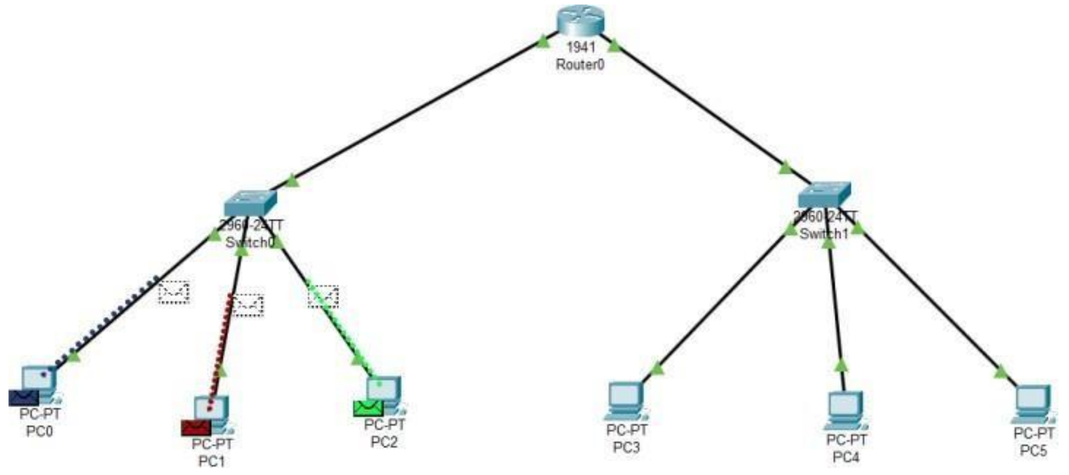

2960-24TT
Switch1
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5

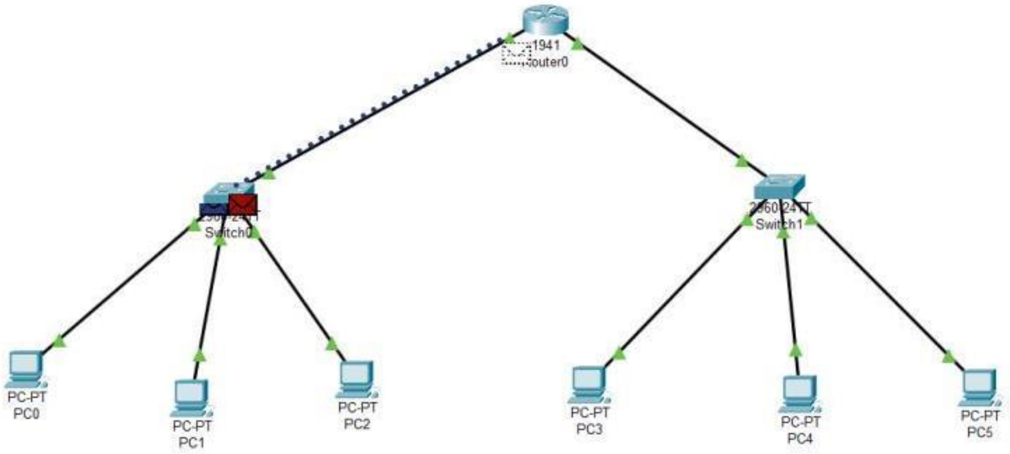
Steps for performing first sub-experiment –

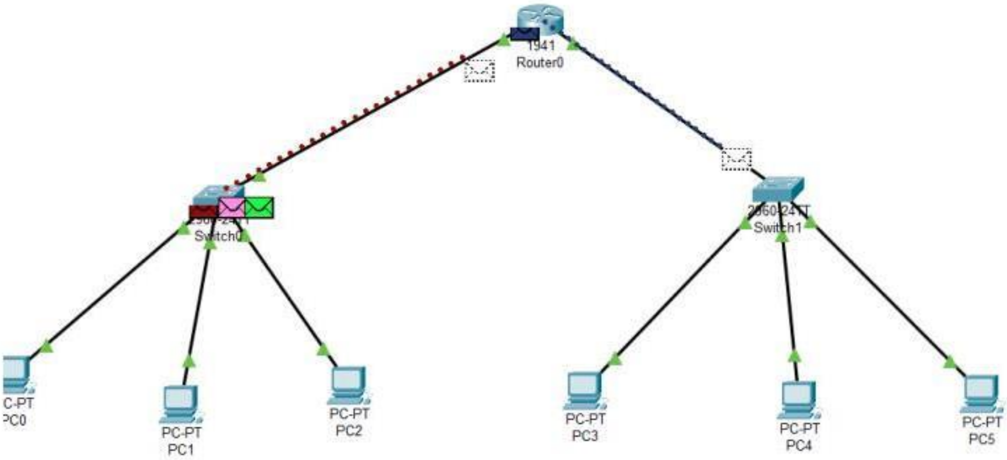
1. Launch the Cisco Packet Tracer application
2. Select three computers and a 2960 Switch
3. Connect the three computers with the switch
4. Configure the IP addresses of the three computers
5. Now you can ping any computer with the other two and see the simulation of transferring of the messages

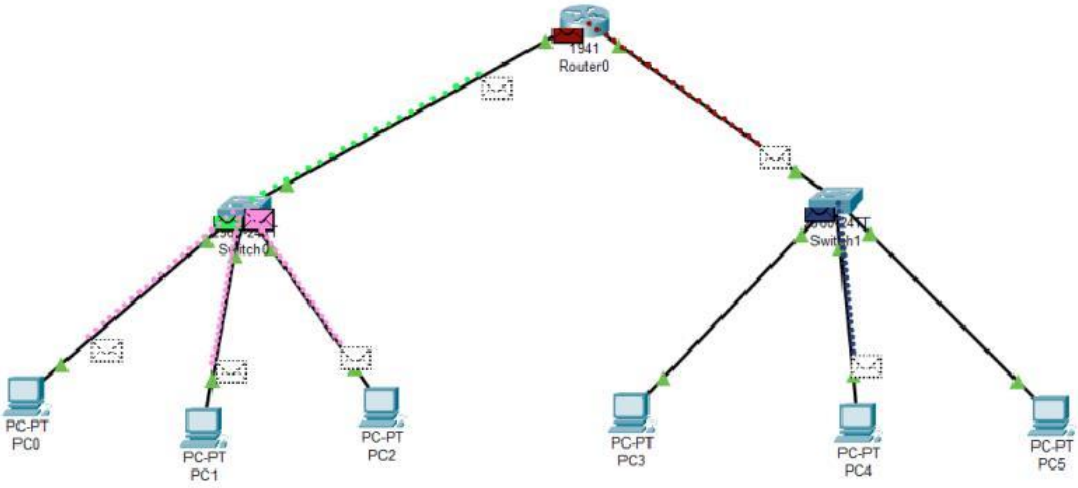


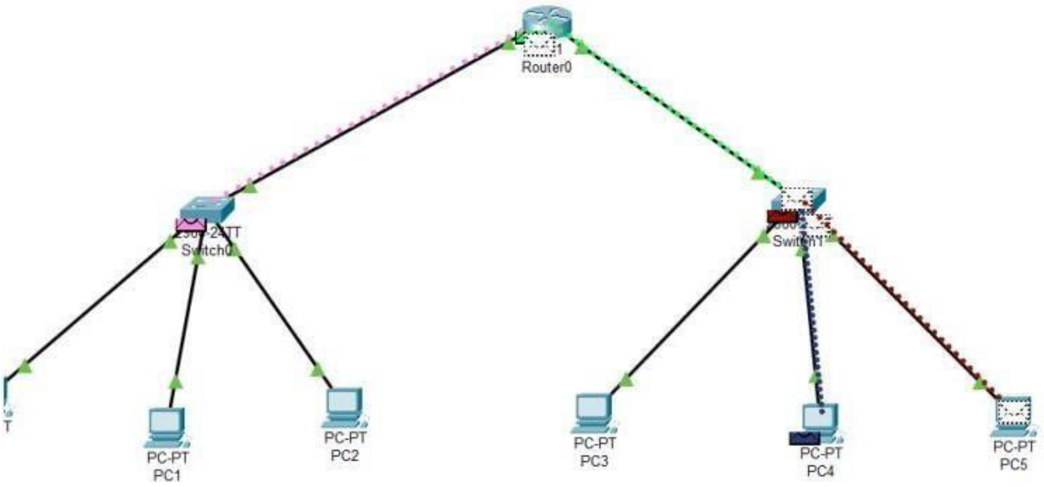


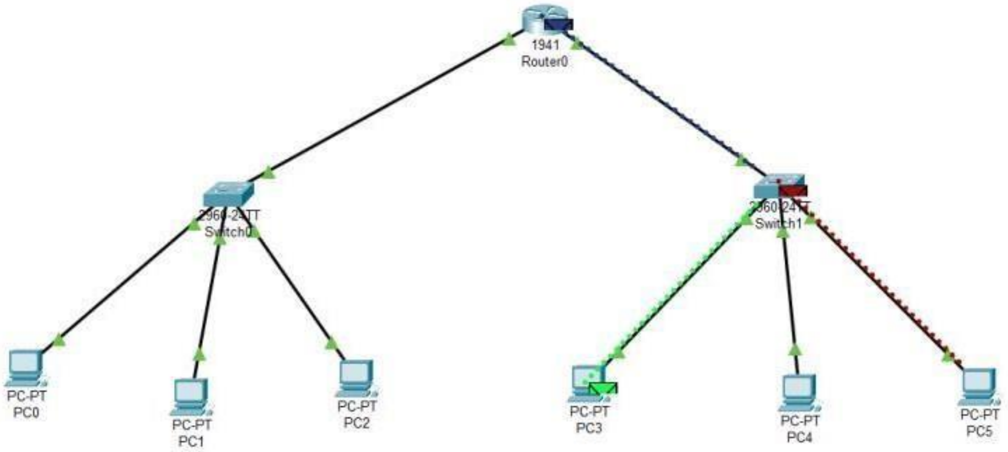


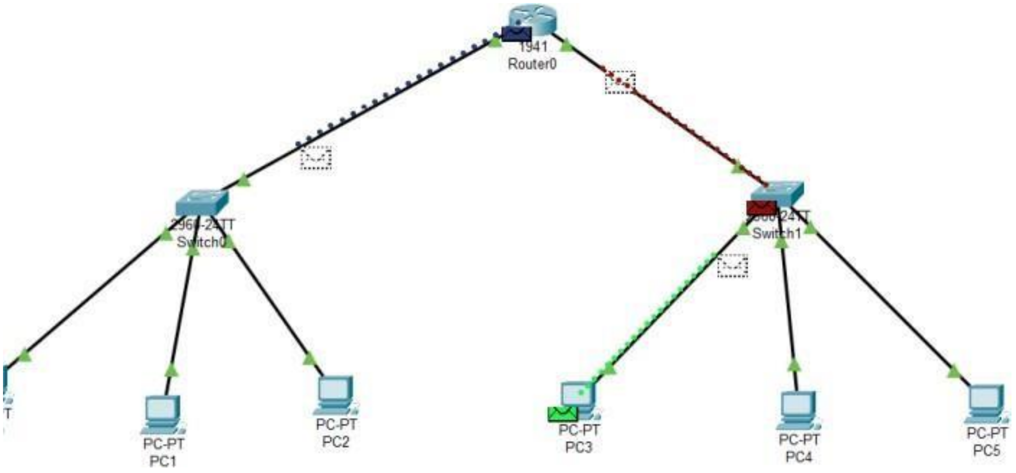

1941
Router0
2960-24TT
Switch0
2960-24TT
Switch1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5

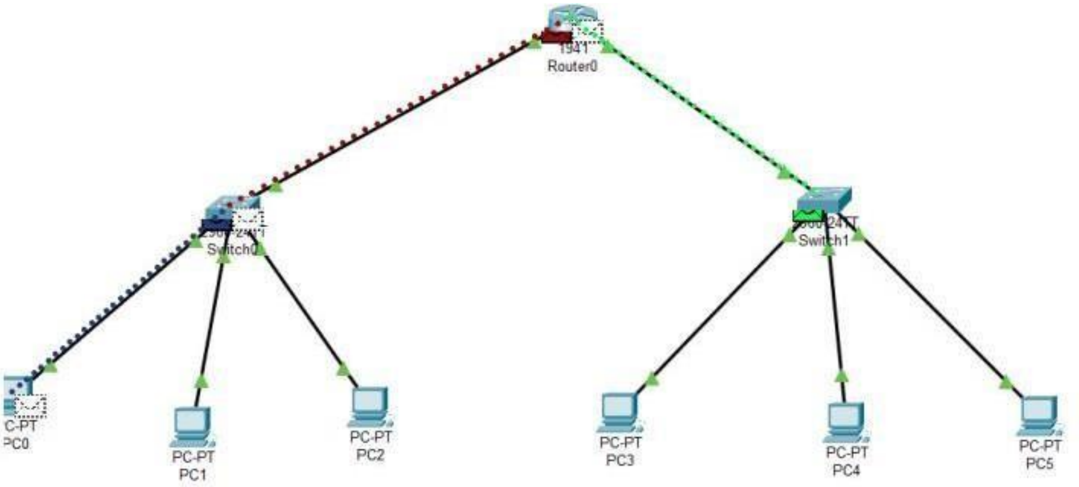

1941
Router0
2960-24TT
Switch0
2960-24TT
Switch1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5

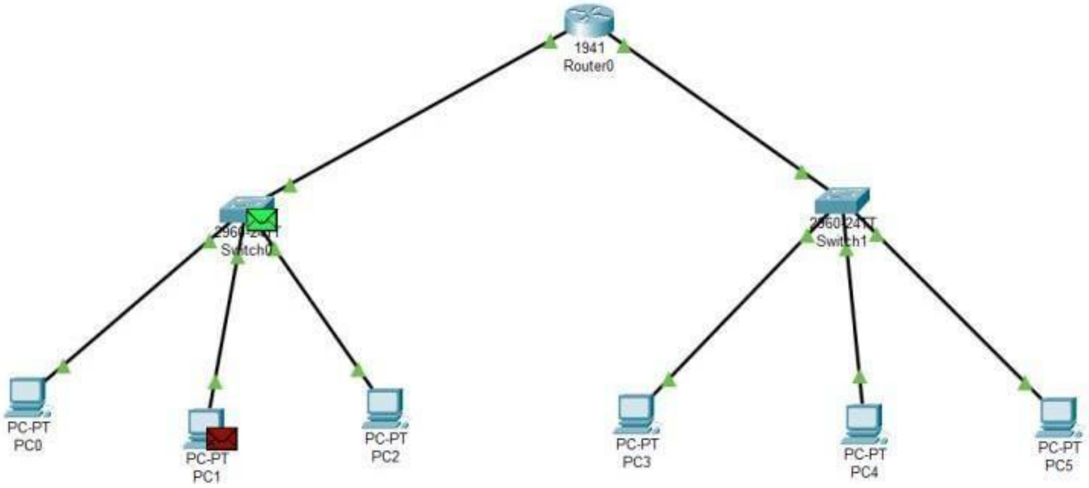

1941
Router0
2960-24TT
Switch0
2960-24TT
Switch1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5

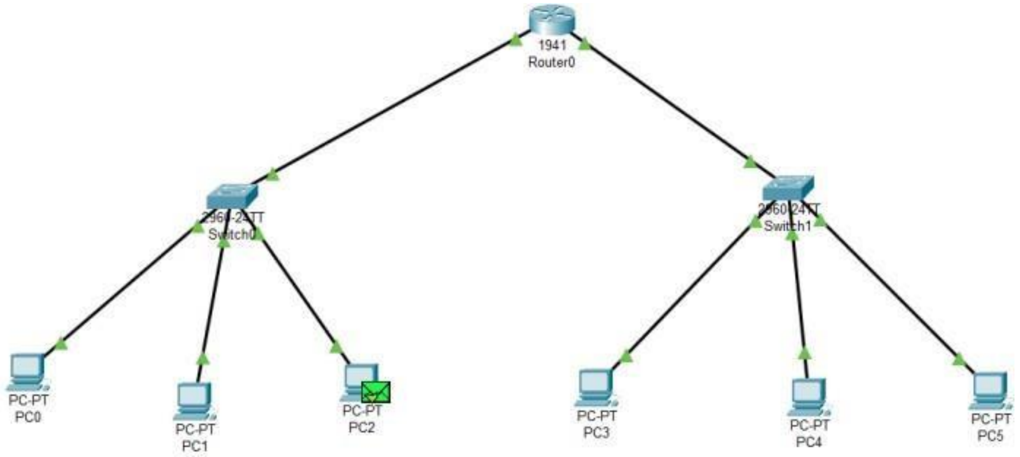

Router0
2960-24TT
Switch0
Switch1
PC-PT
PC1
PC-PT
PC2
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5

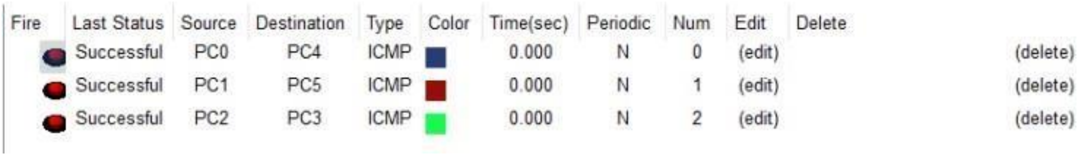

1941
Router0
2960-24TT
Switch0
2960-24TT
Switch1
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5


1941
Router0
2960-24TT
Switch0
2960-24TT
Switch1
PC-PT
PC1
PC-PT
PC2
PC-PT
PC3
PC-PT
PC4
PC-PT
PC5

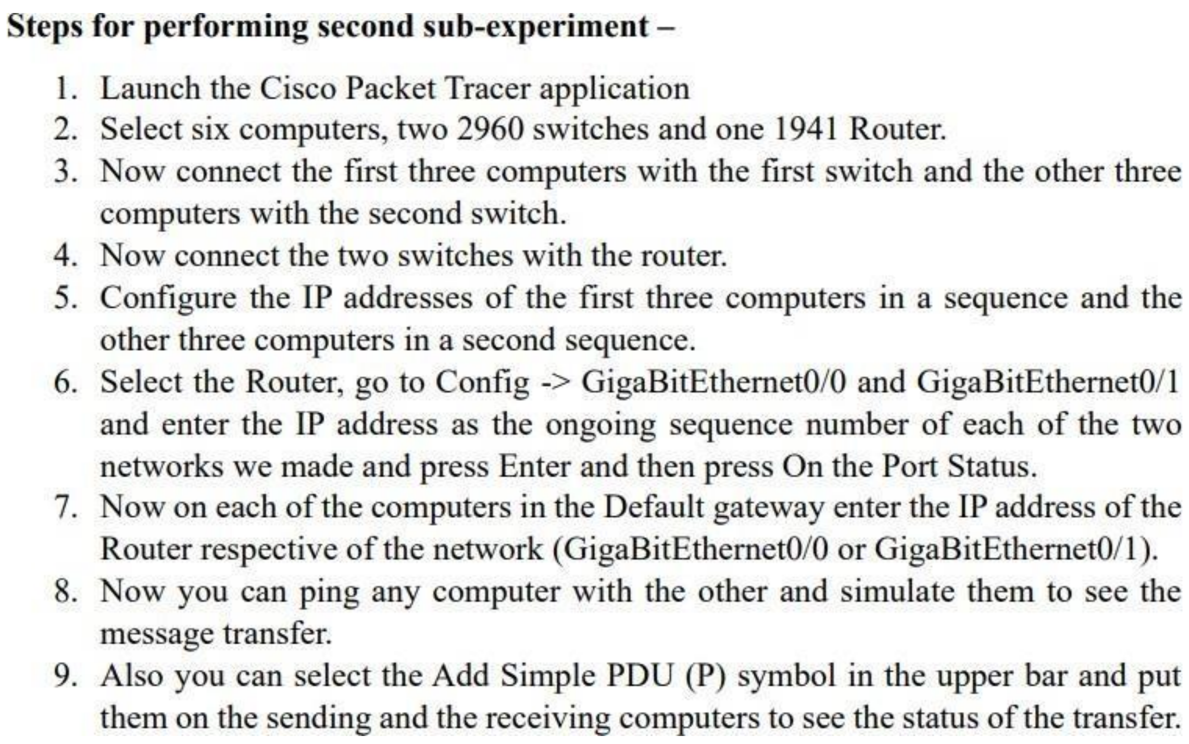








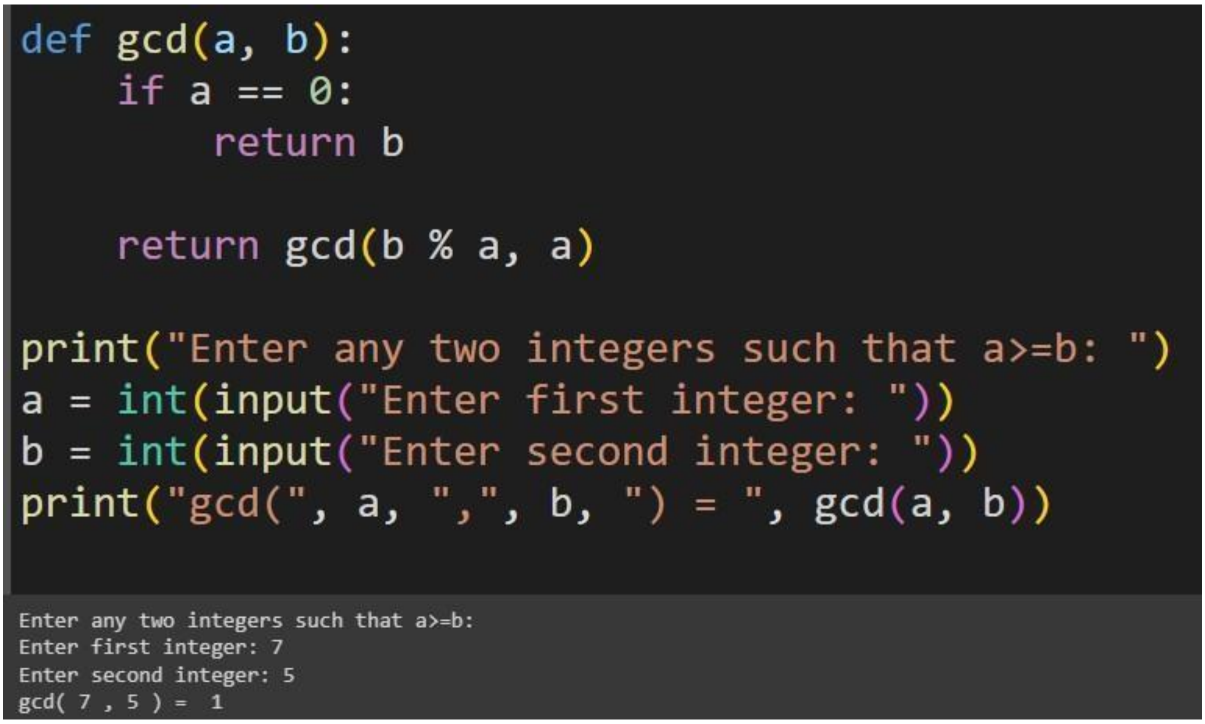
Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	Successful	PC0	PC4	ICMP	🟦	0.000	N	0	(edit)	(delete)
●	Successful	PC1	PC5	ICMP	🟥	0.000	N	1	(edit)	(delete)
●	Successful	PC2	PC3	ICMP	🟩	0.000	N	2	(edit)	(delete)



Steps for performing second sub-experiment –

1. Launch the Cisco Packet Tracer application
2. Select six computers, two 2960 switches and one 1941 Router.
3. Now connect the first three computers with the first switch and the other three computers with the second switch.
4. Now connect the two switches with the router.
5. Configure the IP addresses of the first three computers in a sequence and the other three computers in a second sequence.
6. Select the Router, go to Config -> GigaBitEthernet0/0 and GigaBitEthernet0/1 and enter the IP address as the ongoing sequence number of each of the two networks we made and press Enter and then press On the Port Status.
7. Now on each of the computers in the Default gateway enter the IP address of the Router respective of the network (GigaBitEthernet0/0 or GigaBitEthernet0/1).
8. Now you can ping any computer with the other and simulate them to see the message transfer.
9. Also you can select the Add Simple PDU (P) symbol in the upper bar and put them on the sending and the receiving computers to see the status of the transfer.

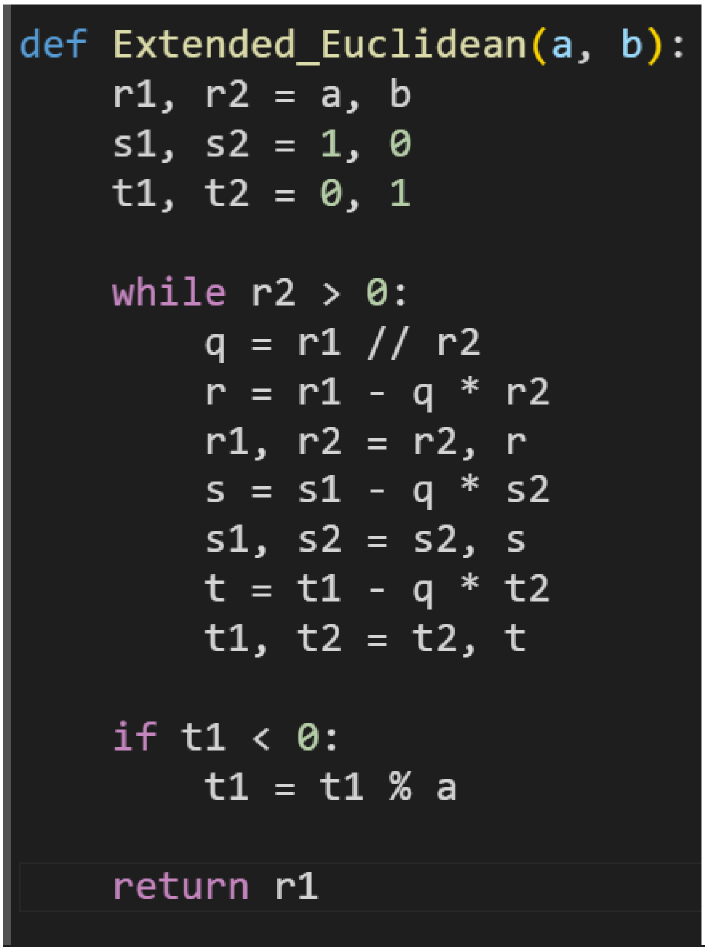
EUCLIDEAN ALGORITHM –



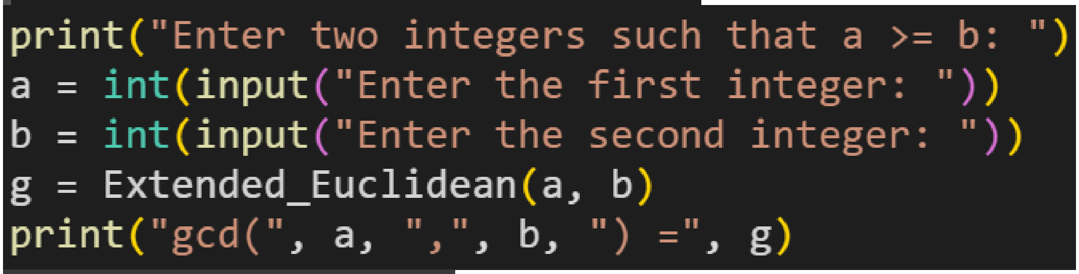
```python
def gcd(a, b):
    if a == 0:
        return b

    return gcd(b % a, a)

print("Enter any two integers such that a>=b: ")
a = int(input("Enter first integer: "))
b = int(input("Enter second integer: "))
print("gcd(", a, ",", b, ") = ", gcd(a, b))
```

```
Enter any two integers such that a>=b:
Enter first integer: 7
Enter second integer: 5
gcd( 7 , 5 ) =  1
```

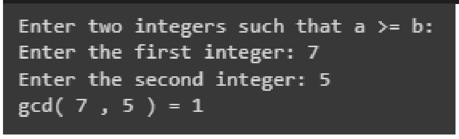
EXTENDED EUCLIDEAN ALGORITHM –



```python
def Extended_Euclidean(a, b):
    r1, r2 = a, b
    s1, s2 = 1, 0
    t1, t2 = 0, 1

    while r2 > 0:
        q = r1 // r2
        r = r1 - q * r2
        r1, r2 = r2, r
        s = s1 - q * s2
        s1, s2 = s2, s
        t = t1 - q * t2
        t1, t2 = t2, t

    if t1 < 0:
        t1 = t1 % a

    return r1
```

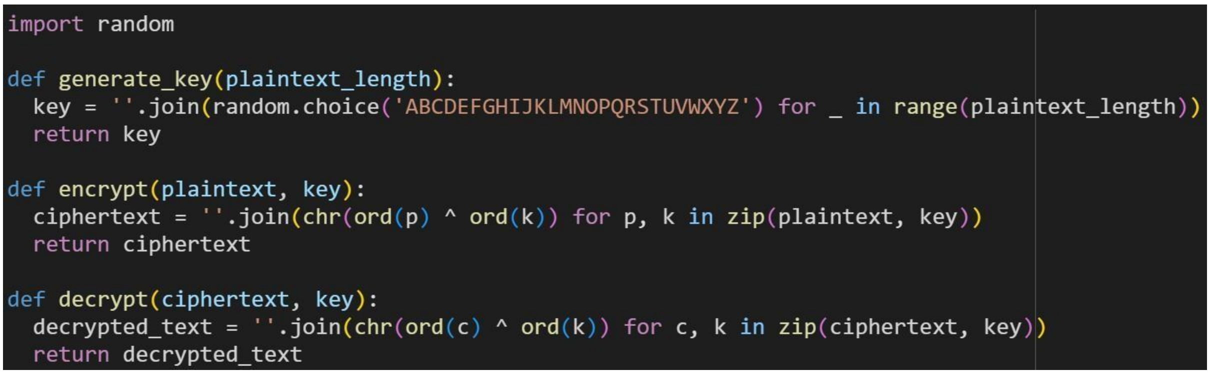


```python
print("Enter two integers such that a >= b: ")
a = int(input("Enter the first integer: "))
b = int(input("Enter the second integer: "))
g = Extended_Euclidean(a, b)
print("gcd(", a, ",", b, ") =", g)
```

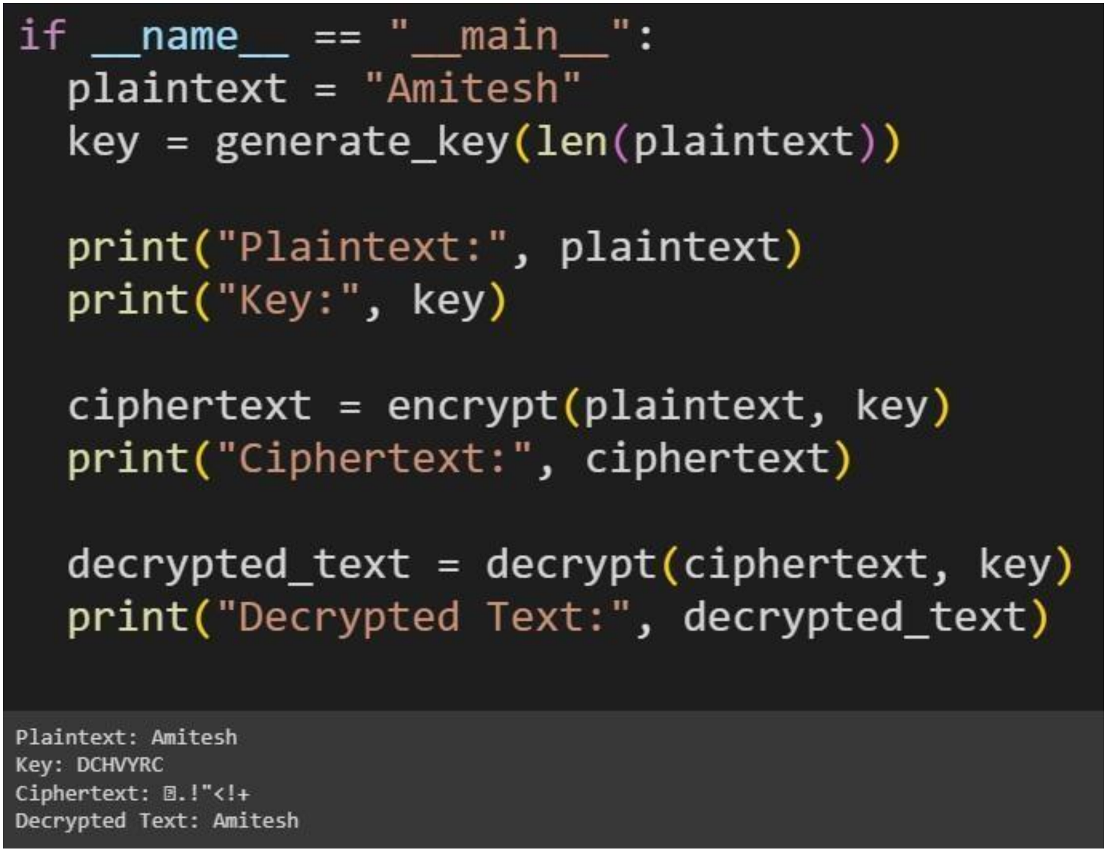


```
Enter two integers such that a >= b:
Enter the first integer: 7
Enter the second integer: 5
gcd( 7 , 5 ) = 1
```

VERNAM CIPHER –



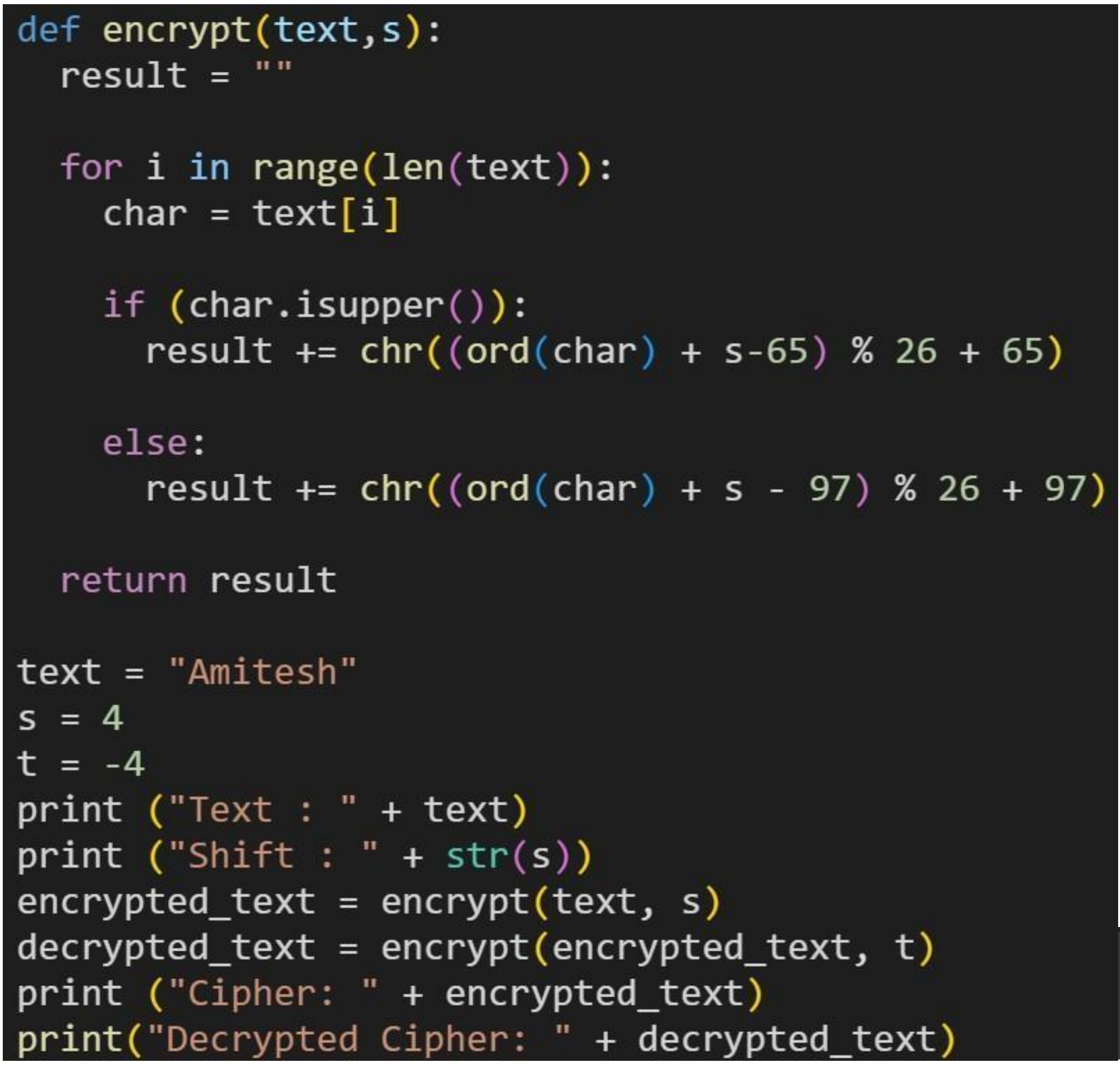
```python
import random

def generate_key(plaintext_length):
  key = ''.join(random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ') for _ in range(plaintext_length))
  return key

def encrypt(plaintext, key):
  ciphertext = ''.join(chr(ord(p) ^ ord(k)) for p, k in zip(plaintext, key))
  return ciphertext

def decrypt(ciphertext, key):
  decrypted_text = ''.join(chr(ord(c) ^ ord(k)) for c, k in zip(ciphertext, key))
  return decrypted_text
```



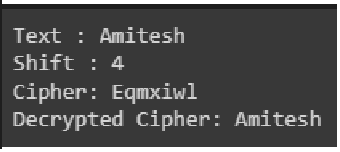
```python
if __name__ == "__main__":
  plaintext = "Amitesh"
  key = generate_key(len(plaintext))

  print("Plaintext:", plaintext)
  print("Key:", key)

  ciphertext = encrypt(plaintext, key)
  print("Ciphertext:", ciphertext)

  decrypted_text = decrypt(ciphertext, key)
  print("Decrypted Text:", decrypted_text)
```

```
Plaintext: Amitesh
Key: DCHVYRC
Ciphertext: ⍰.!"<!+
Decrypted Text: Amitesh
```

CAESAR CIPHER –

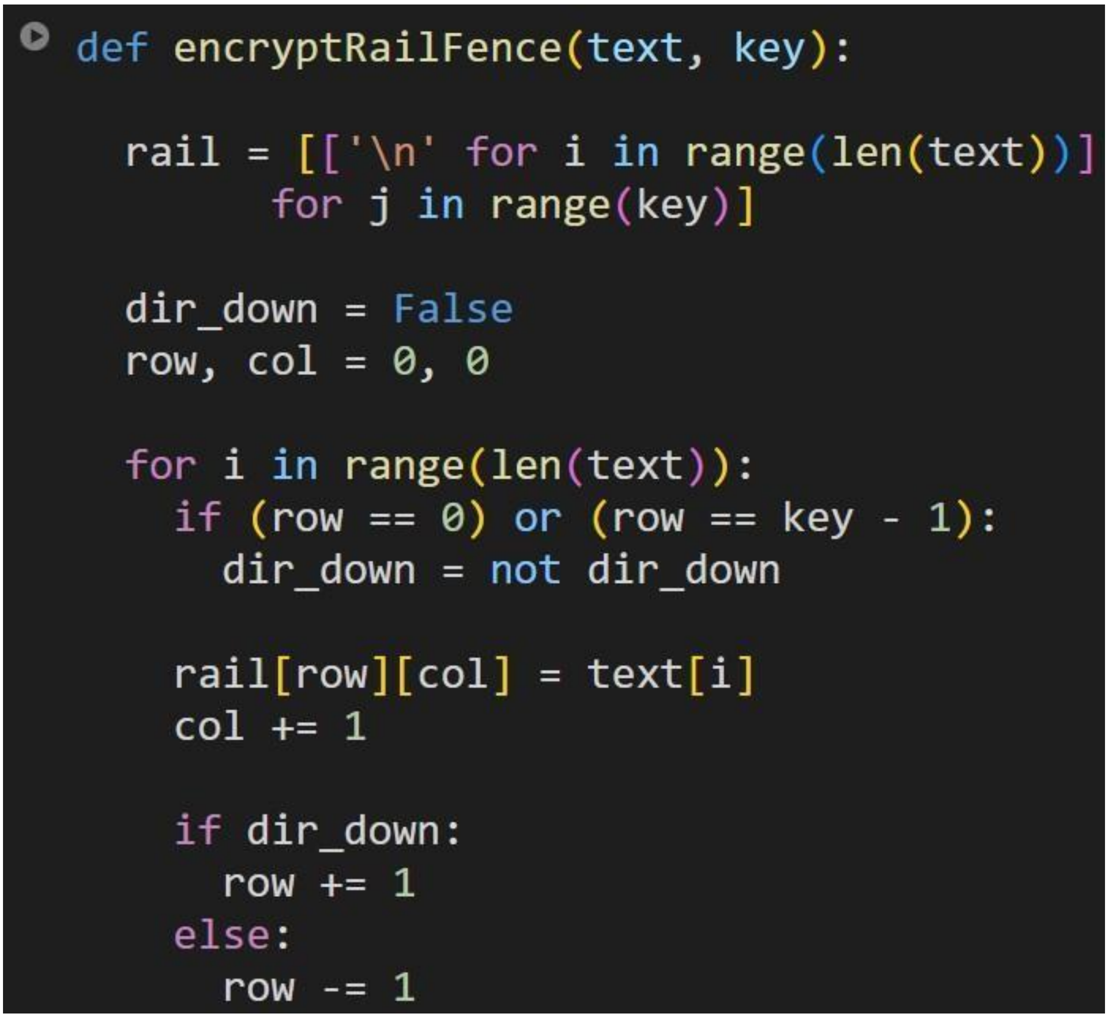


```
def encrypt(text,s):
  result = ""

  for i in range(len(text)):
    char = text[i]

    if (char.isupper()):
      result += chr((ord(char) + s-65) % 26 + 65)

    else:
      result += chr((ord(char) + s - 97) % 26 + 97)

  return result

text = "Amitesh"
s = 4
t = -4
print ("Text : " + text)
print ("Shift : " + str(s))
encrypted_text = encrypt(text, s)
decrypted_text = encrypt(encrypted_text, t)
print ("Cipher: " + encrypted_text)
print("Decrypted Cipher: " + decrypted_text)
```

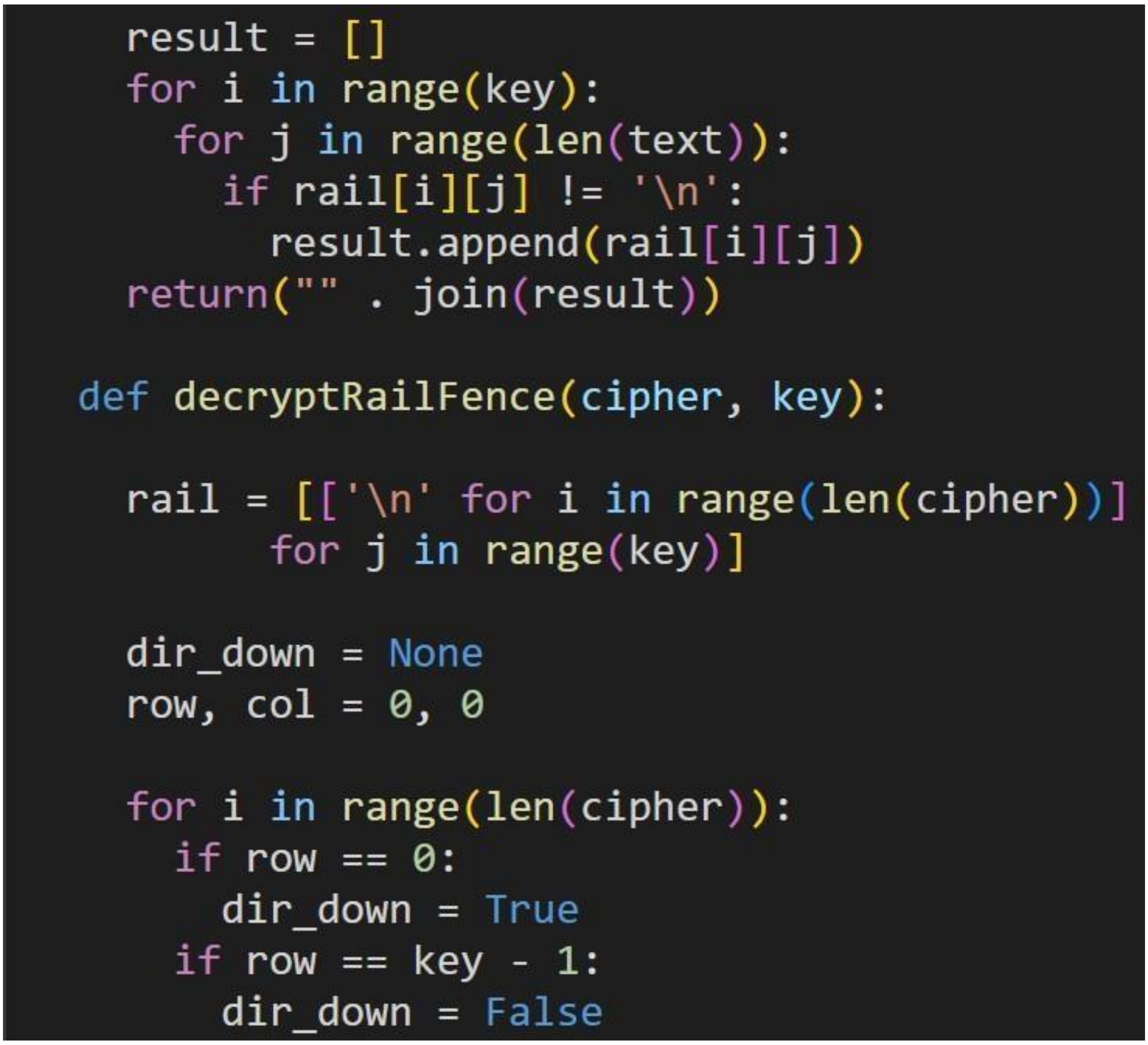


```
Text : Amitesh
Shift : 4
Cipher: Eqmxiwl
Decrypted Cipher: Amitesh
```

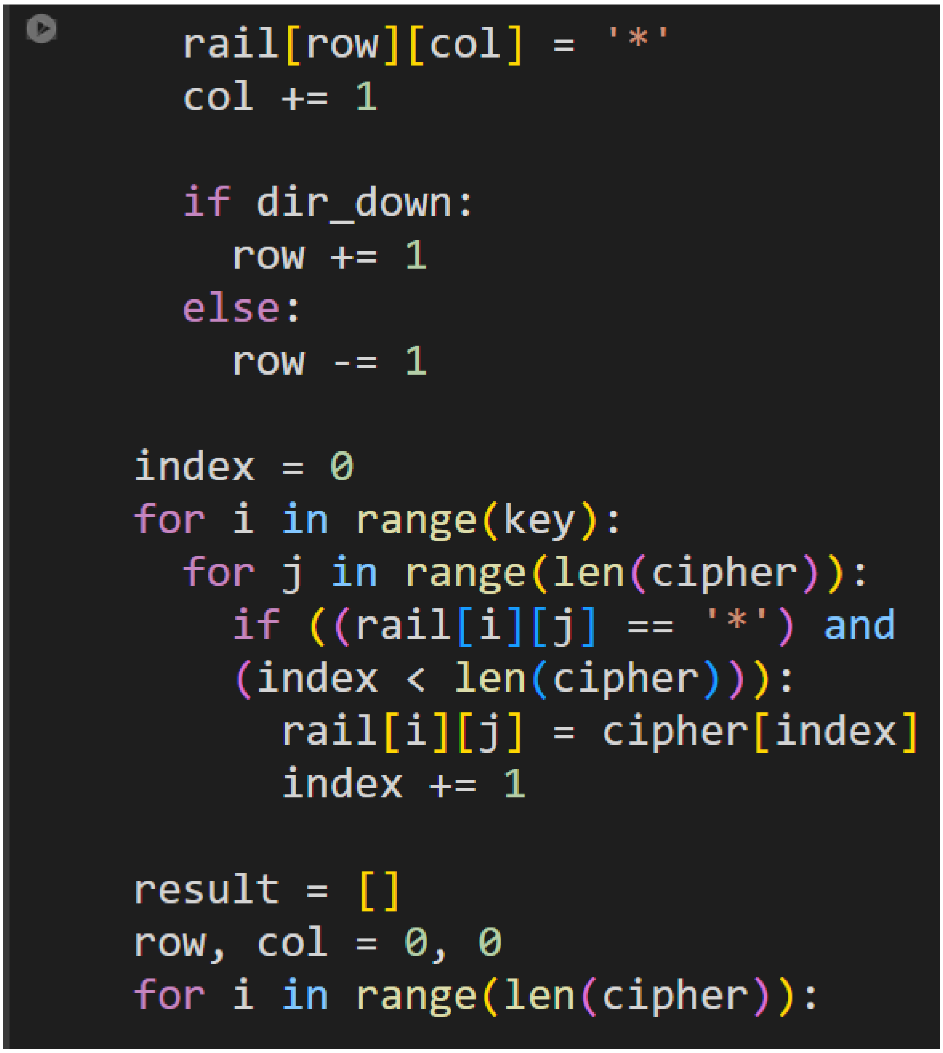
RAILFENCE ALGORITHM –



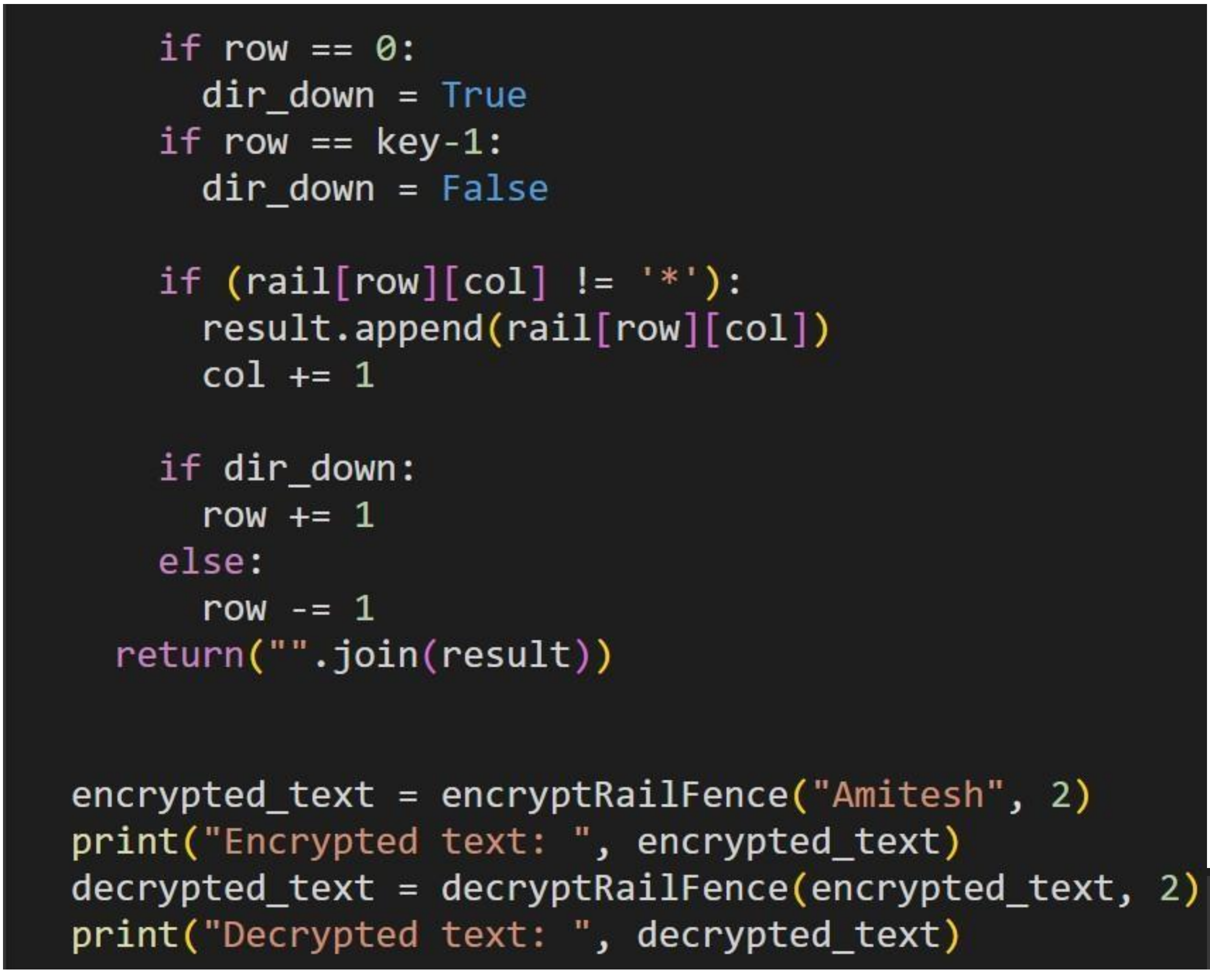
```python
def encryptRailFence(text, key):

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

  dir_down = False
  row, col = 0, 0

  for i in range(len(text)):
    if (row == 0) or (row == key - 1):
      dir_down = not dir_down

    rail[row][col] = text[i]
    col += 1

    if dir_down:
      row += 1
    else:
      row -= 1
```



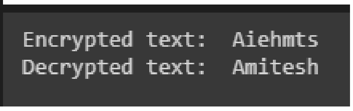
```
  result = []
  for i in range(key):
    for j in range(len(text)):
      if rail[i][j] != '\n':
        result.append(rail[i][j])
  return("" . join(result))

def decryptRailFence(cipher, key):

  rail = [['\n' for i in range(len(cipher))]
        for j in range(key)]

  dir_down = None
  row, col = 0, 0

  for i in range(len(cipher)):
    if row == 0:
      dir_down = True
    if row == key - 1:
      dir_down = False
```



```
    rail[row][col] = '*'
    col += 1

    if dir_down:
      row += 1
    else:
      row -= 1

  index = 0
  for i in range(key):
    for j in range(len(cipher)):
      if ((rail[i][j] == '*') and
      (index < len(cipher))):
        rail[i][j] = cipher[index]
        index += 1

  result = []
  row, col = 0, 0
  for i in range(len(cipher)):
```

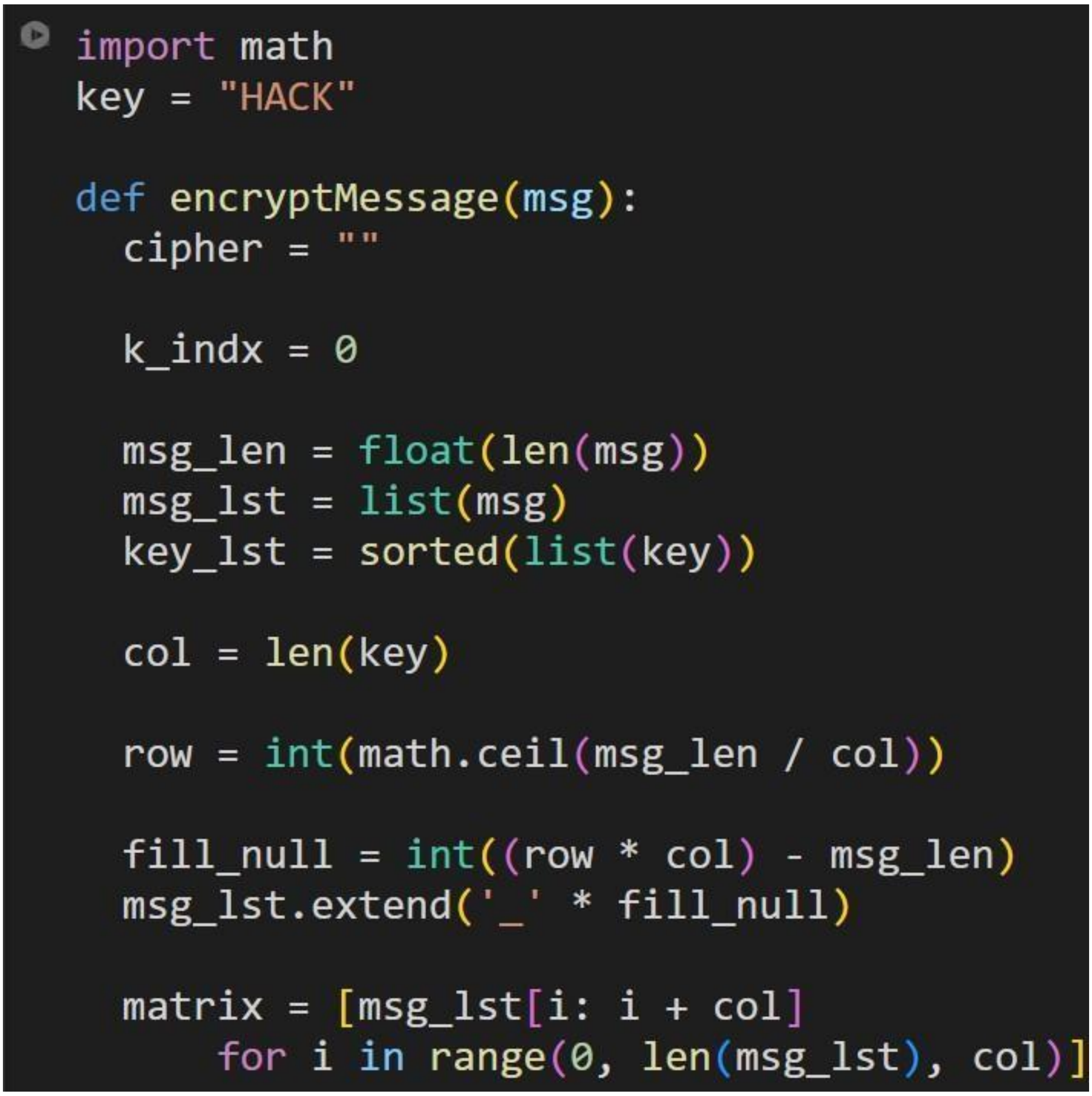


```
    if row == 0:
      dir_down = True
    if row == key-1:
      dir_down = False

    if (rail[row][col] != '*'):
      result.append(rail[row][col])
      col += 1

    if dir_down:
      row += 1
    else:
      row -= 1
  return("".join(result))


encrypted_text = encryptRailFence("Amitesh", 2)
print("Encrypted text: ", encrypted_text)
decrypted_text = decryptRailFence(encrypted_text, 2)
print("Decrypted text: ", decrypted_text)
```

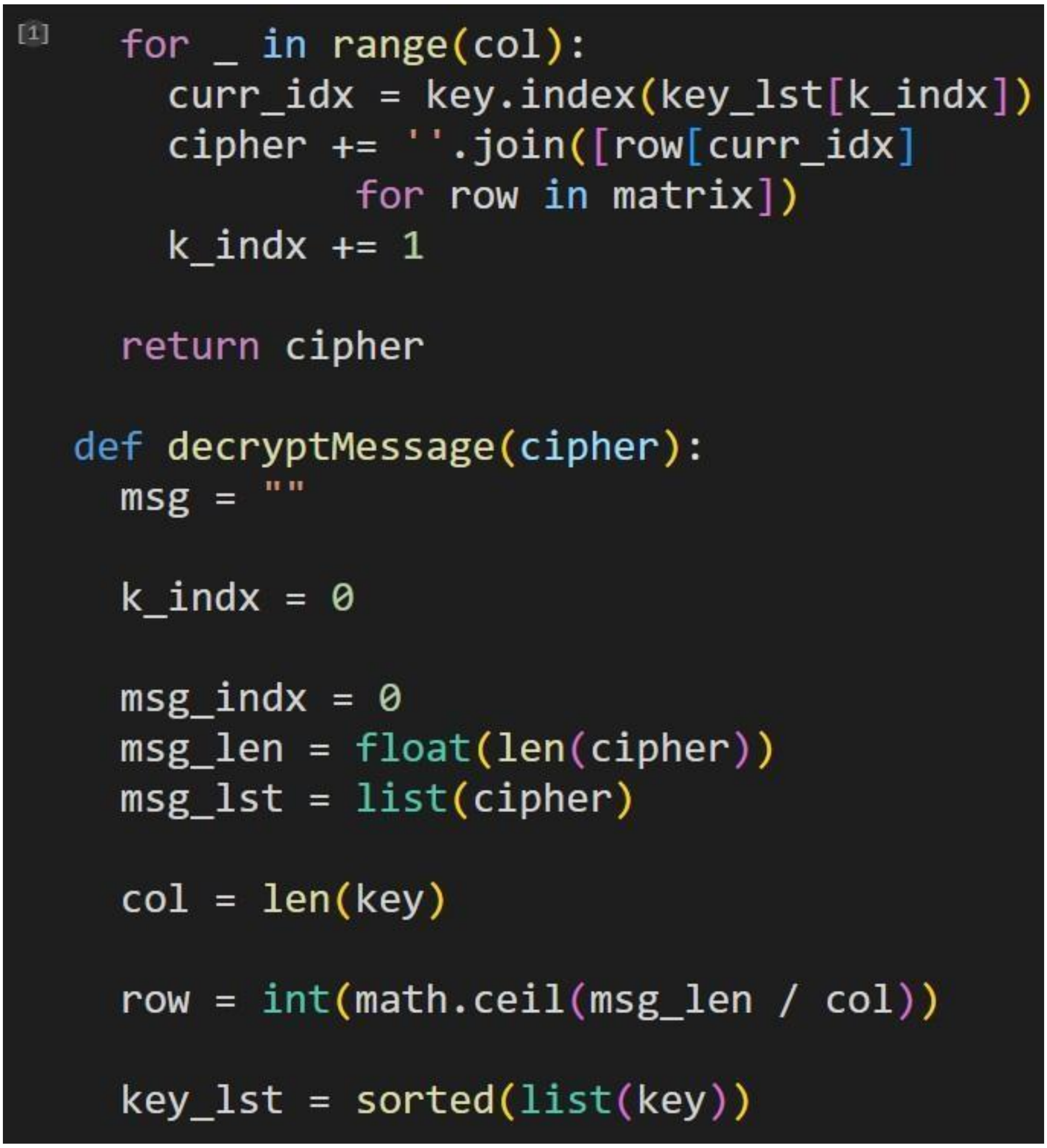


```
Encrypted text:  Aiehmts
Decrypted text:  Amitesh
```

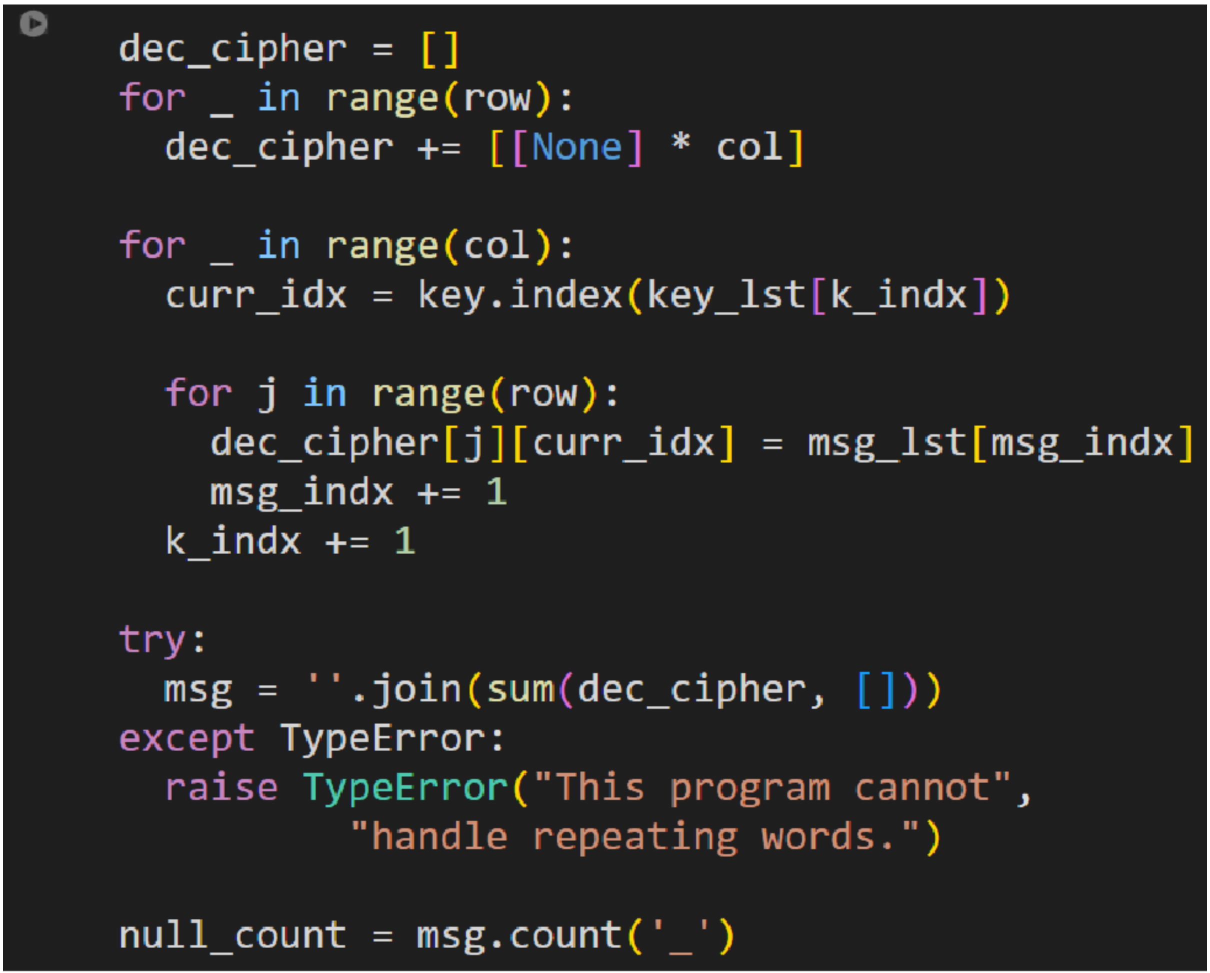
COLUMNAR TRANSPOSITION –



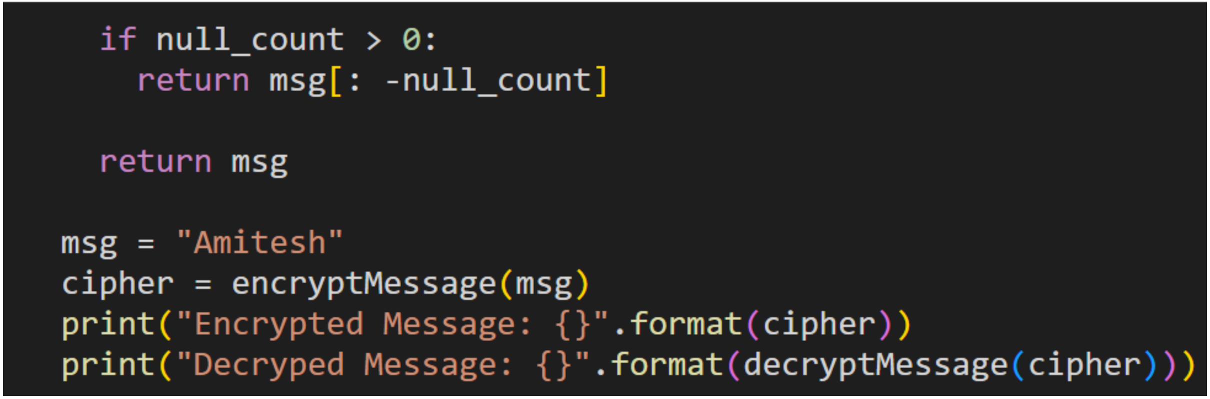
```python
import math
key = "HACK"

def encryptMessage(msg):
  cipher = ""

  k_indx = 0

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

  col = len(key)

  row = int(math.ceil(msg_len / col))

  fill_null = int((row * col) - msg_len)
  msg_lst.extend('_' * fill_null)

  matrix = [msg_lst[i: i + col]
      for i in range(0, len(msg_lst), col)]
```



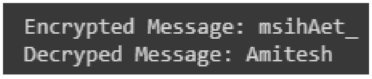
```
  for _ in range(col):
    curr_idx = key.index(key_lst[k_indx])
    cipher += ''.join([row[curr_idx]
            for row in matrix])
    k_indx += 1

  return cipher

def decryptMessage(cipher):
  msg = ""

  k_indx = 0

  msg_indx = 0
  msg_len = float(len(cipher))
  msg_lst = list(cipher)

  col = len(key)

  row = int(math.ceil(msg_len / col))

  key_lst = sorted(list(key))
```



```
  dec_cipher = []
  for _ in range(row):
    dec_cipher += [[None] * col]

  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

  try:
    msg = ''.join(sum(dec_cipher, []))
  except TypeError:
    raise TypeError("This program cannot",
            "handle repeating words.")

  null_count = msg.count('_')
```

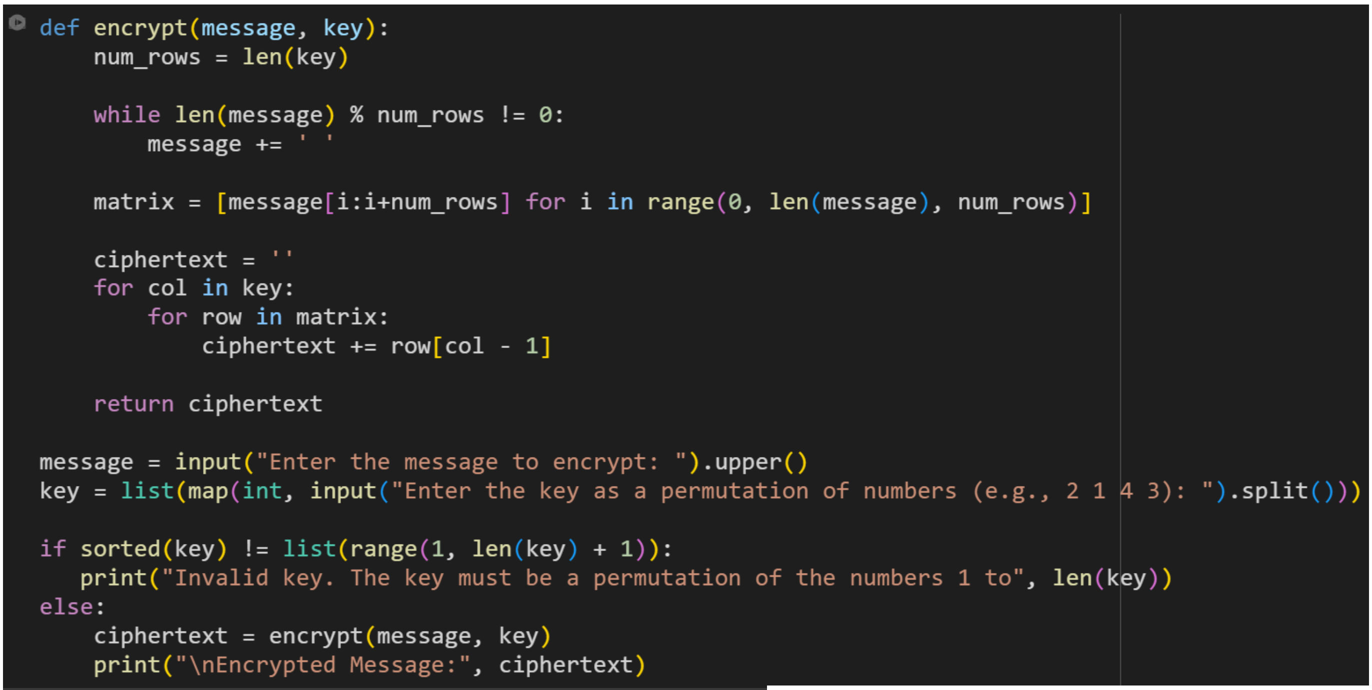


```
  if null_count > 0:
    return msg[: -null_count]

  return msg

msg = "Amitesh"
cipher = encryptMessage(msg)
print("Encrypted Message: {}".format(cipher))
print("Decryped Message: {}".format(decryptMessage(cipher)))
```

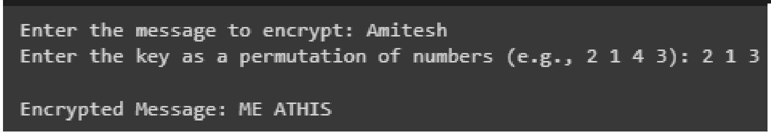


```
Encrypted Message: msihAet_
Decryped Message: Amitesh
```

ROW TRANSPOSITION –

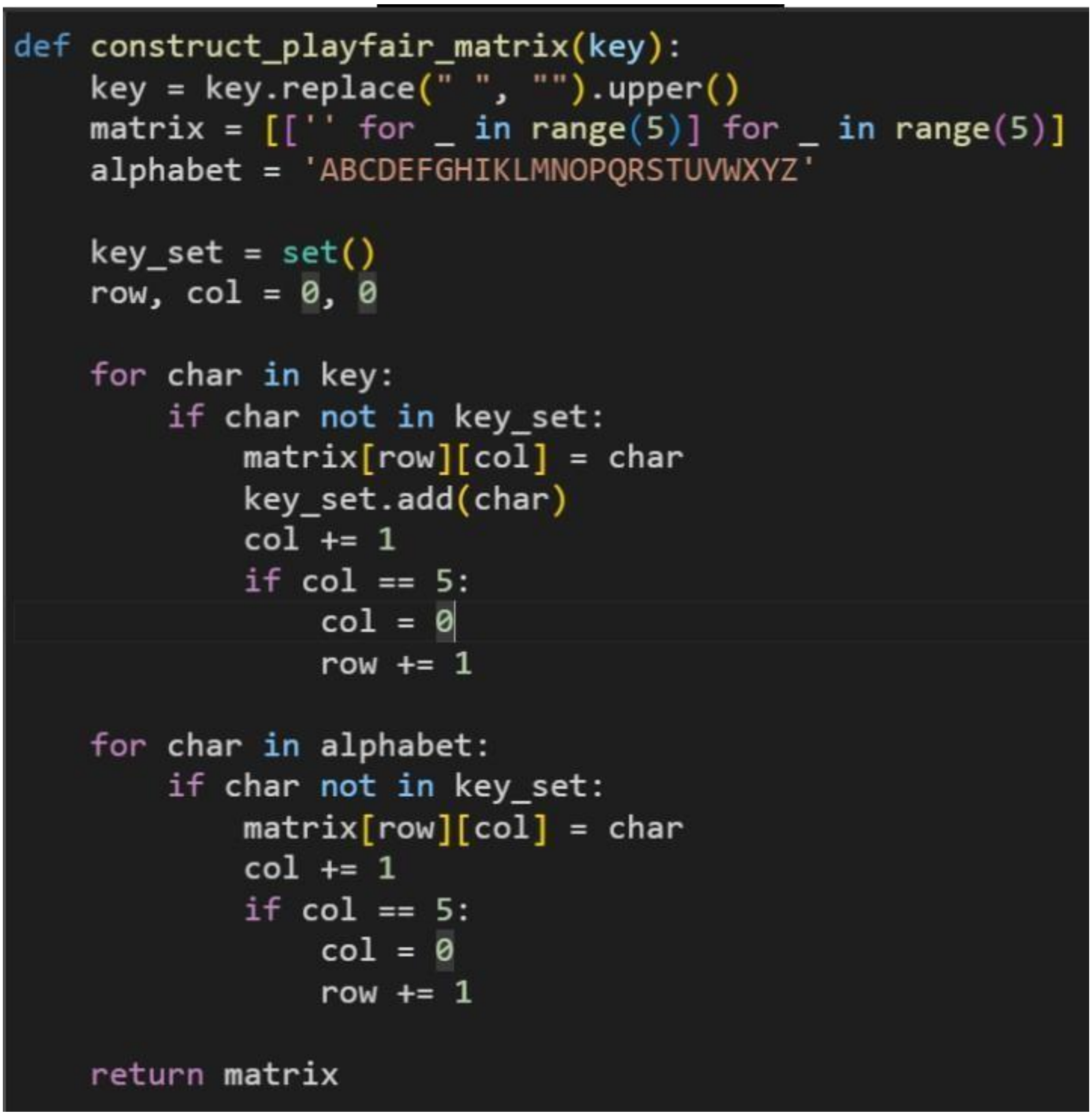


```python
def encrypt(message, key):
    num_rows = len(key)

    while len(message) % num_rows != 0:
        message += ' '

    matrix = [message[i:i+num_rows] for i in range(0, len(message), num_rows)]

    ciphertext = ''
    for col in key:
        for row in matrix:
            ciphertext += row[col - 1]

    return ciphertext

message = input("Enter the message to encrypt: ").upper()
key = list(map(int, input("Enter the key as a permutation of numbers (e.g., 2 1 4 3): ").split()))

if sorted(key) != list(range(1, len(key) + 1)):
   print("Invalid key. The key must be a permutation of the numbers 1 to", len(key))
else:
    ciphertext = encrypt(message, key)
    print("\nEncrypted Message:", ciphertext)
```

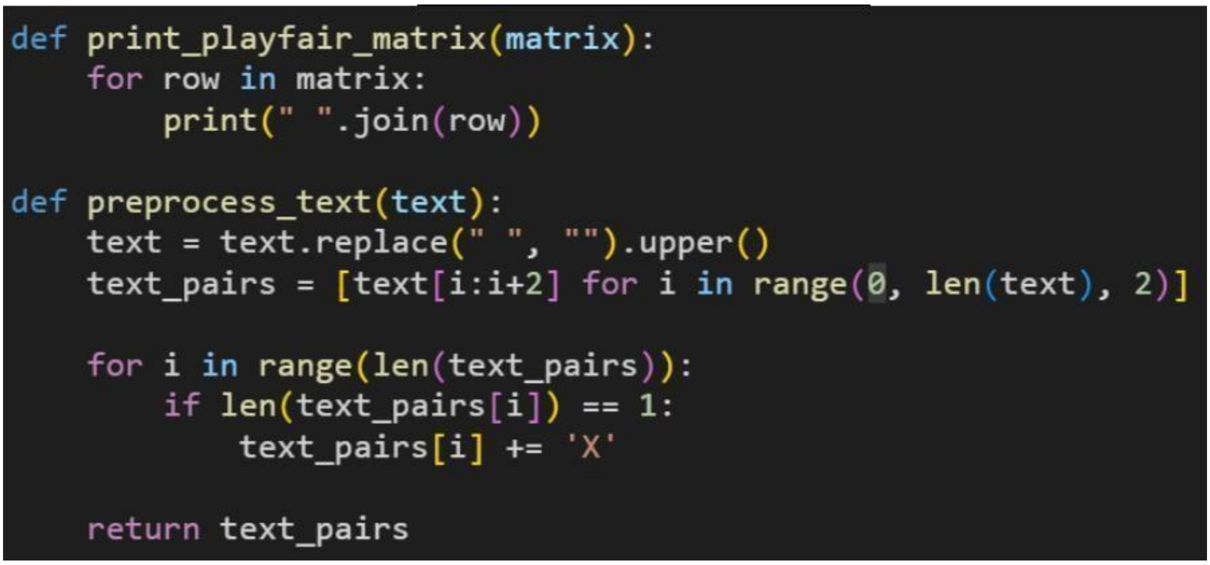


```
Enter the message to encrypt: Amitesh
Enter the key as a permutation of numbers (e.g., 2 1 4 3): 2 1 3

Encrypted Message: ME ATHIS
```

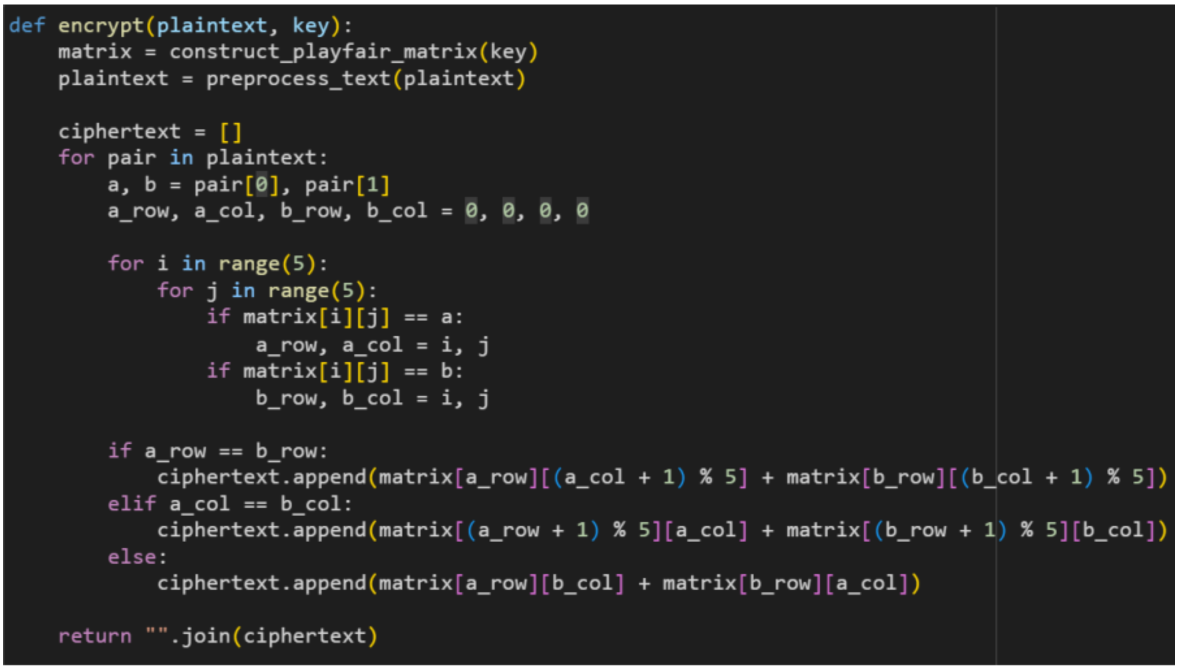
PLAYFAIR CIPHER –



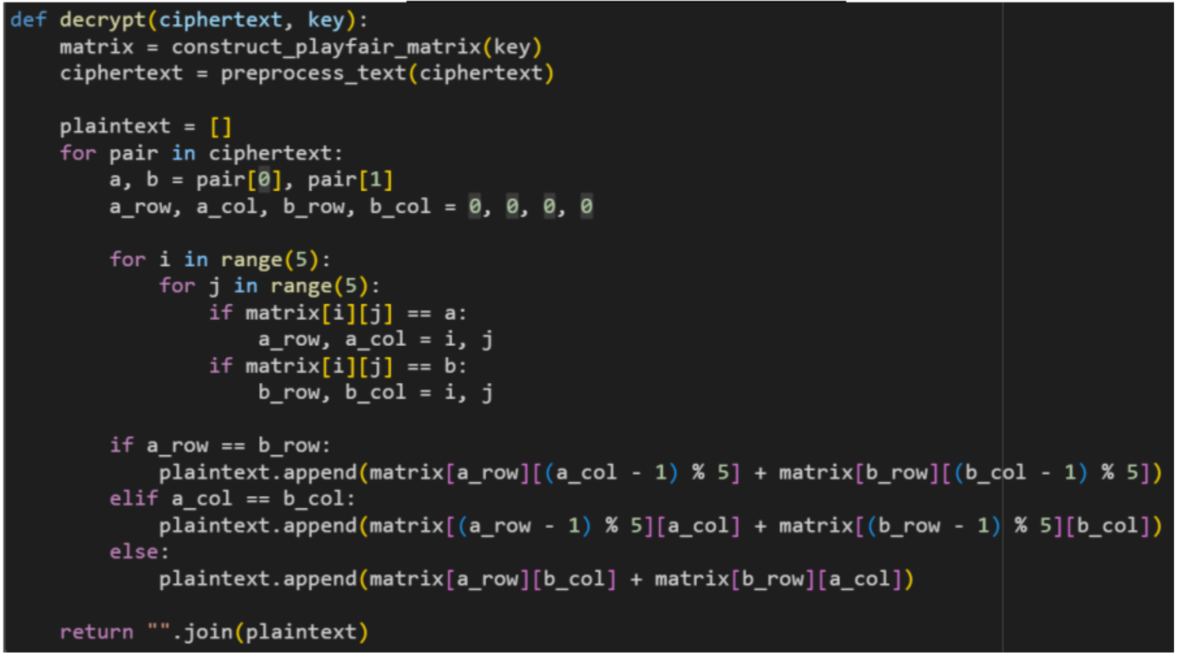
```python
def construct_playfair_matrix(key):
    key = key.replace(" ", "").upper()
    matrix = [['' for _ in range(5)] for _ in range(5)]
    alphabet = 'ABCDEFGHIKLMNOPQRSTUVWXYZ'

    key_set = set()
    row, col = 0, 0

    for char in key:
        if char not in key_set:
            matrix[row][col] = char
            key_set.add(char)
            col += 1
            if col == 5:
                col = 0
                row += 1

    for char in alphabet:
        if char not in key_set:
            matrix[row][col] = char
            col += 1
            if col == 5:
                col = 0
                row += 1

    return matrix
```



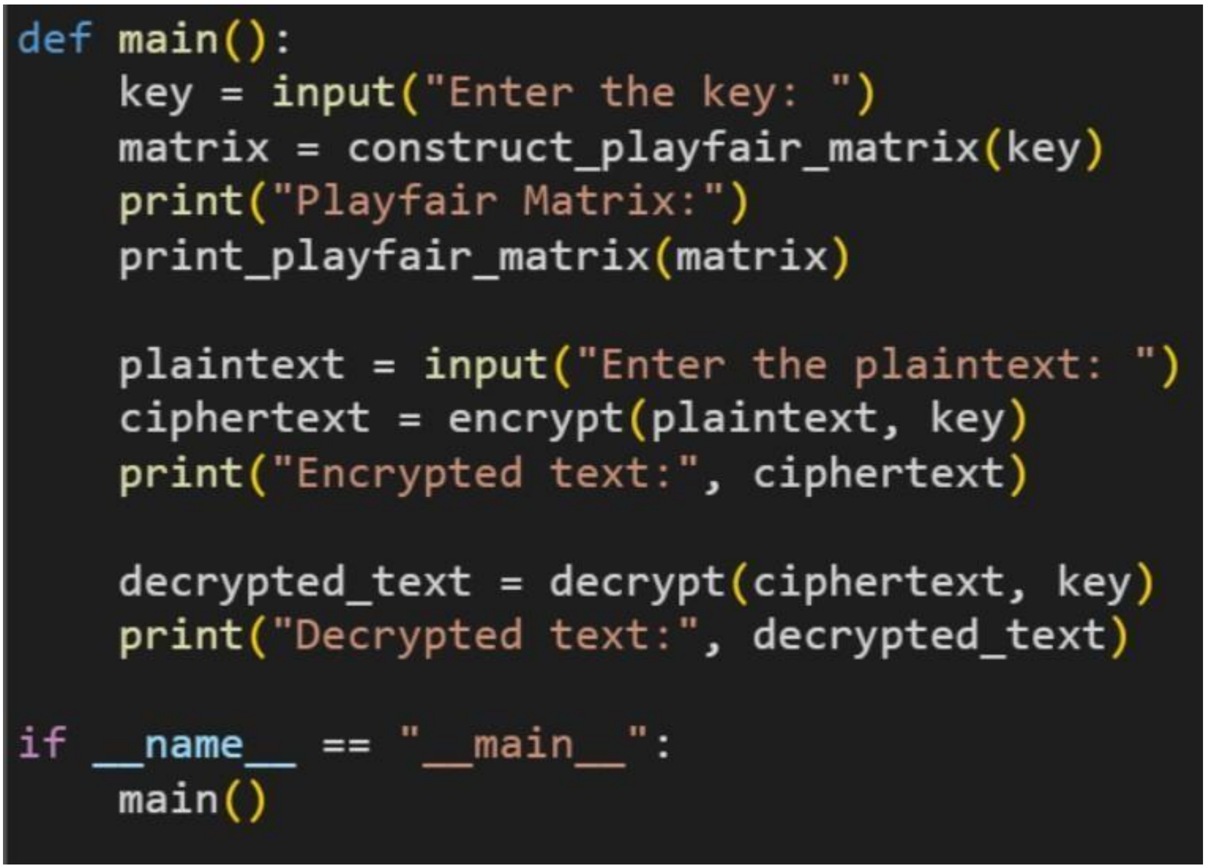
```python
def print_playfair_matrix(matrix):
    for row in matrix:
        print(" ".join(row))

def preprocess_text(text):
    text = text.replace(" ", "").upper()
    text_pairs = [text[i:i+2] for i in range(0, len(text), 2)]

    for i in range(len(text_pairs)):
        if len(text_pairs[i]) == 1:
            text_pairs[i] += 'X'

    return text_pairs
```



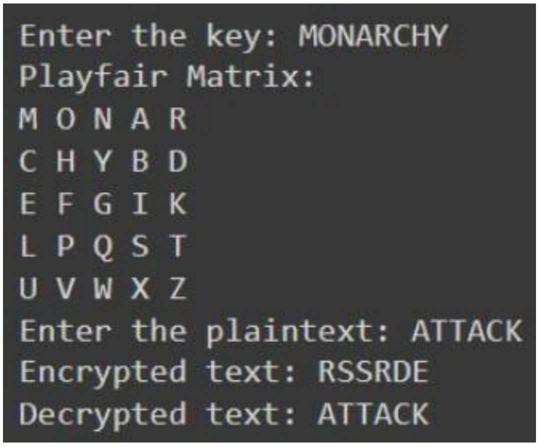
```python
def encrypt(plaintext, key):
    matrix = construct_playfair_matrix(key)
    plaintext = preprocess_text(plaintext)

    ciphertext = []
    for pair in plaintext:
        a, b = pair[0], pair[1]
        a_row, a_col, b_row, b_col = 0, 0, 0, 0

        for i in range(5):
            for j in range(5):
                if matrix[i][j] == a:
                    a_row, a_col = i, j
                if matrix[i][j] == b:
                    b_row, b_col = i, j

        if a_row == b_row:
            ciphertext.append(matrix[a_row][(a_col + 1) % 5] + matrix[b_row][(b_col + 1) % 5])
        elif a_col == b_col:
            ciphertext.append(matrix[(a_row + 1) % 5][a_col] + matrix[(b_row + 1) % 5][b_col])
        else:
            ciphertext.append(matrix[a_row][b_col] + matrix[b_row][a_col])

    return "".join(ciphertext)
```



```python
def decrypt(ciphertext, key):
    matrix = construct_playfair_matrix(key)
    ciphertext = preprocess_text(ciphertext)

    plaintext = []
    for pair in ciphertext:
        a, b = pair[0], pair[1]
        a_row, a_col, b_row, b_col = 0, 0, 0, 0

        for i in range(5):
            for j in range(5):
                if matrix[i][j] == a:
                    a_row, a_col = i, j
                if matrix[i][j] == b:
                    b_row, b_col = i, j

        if a_row == b_row:
            plaintext.append(matrix[a_row][(a_col - 1) % 5] + matrix[b_row][(b_col - 1) % 5])
        elif a_col == b_col:
            plaintext.append(matrix[(a_row - 1) % 5][a_col] + matrix[(b_row - 1) % 5][b_col])
        else:
            plaintext.append(matrix[a_row][b_col] + matrix[b_row][a_col])

    return "".join(plaintext)
```

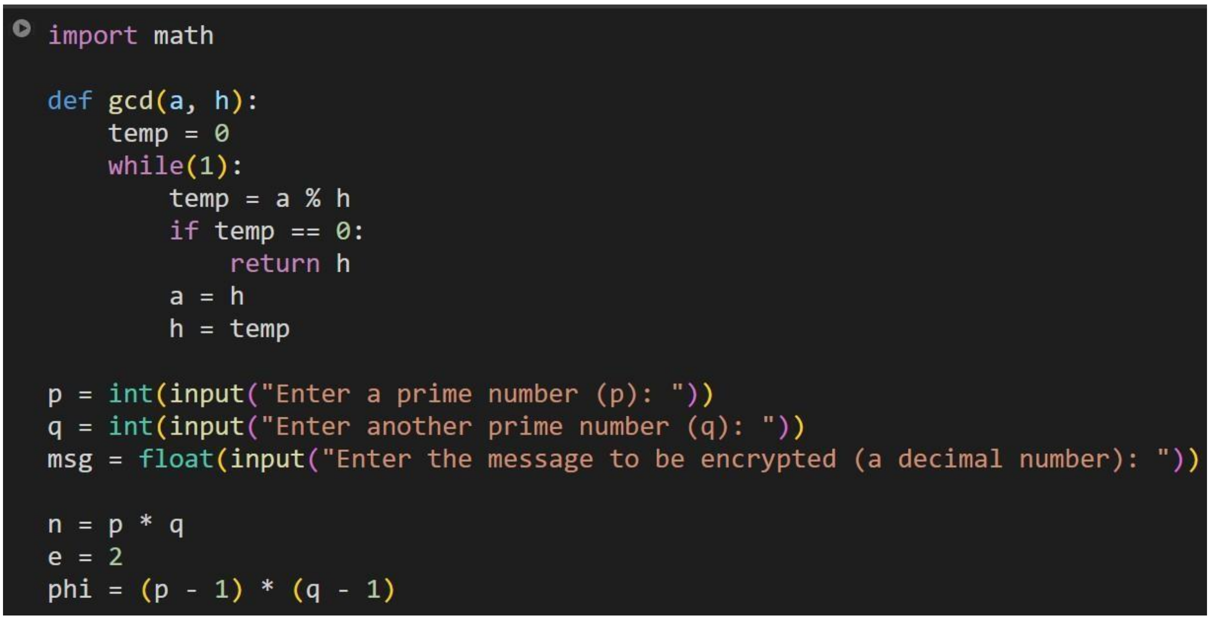


```python
def main():
    key = input("Enter the key: ")
    matrix = construct_playfair_matrix(key)
    print("Playfair Matrix:")
    print_playfair_matrix(matrix)

    plaintext = input("Enter the plaintext: ")
    ciphertext = encrypt(plaintext, key)
    print("Encrypted text:", ciphertext)

    decrypted_text = decrypt(ciphertext, key)
    print("Decrypted text:", decrypted_text)

if __name__ == "__main__":
    main()
```

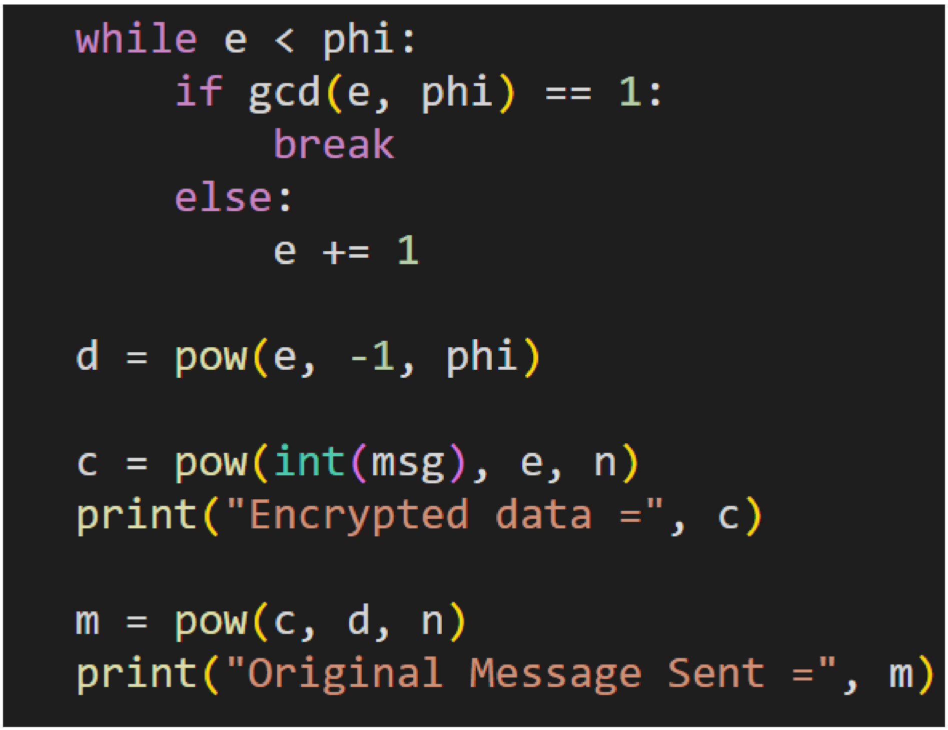


```
Enter the key: MONARCHY
Playfair Matrix:
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z
Enter the plaintext: ATTACK
Encrypted text: RSSRDE
Decrypted text: ATTACK
```

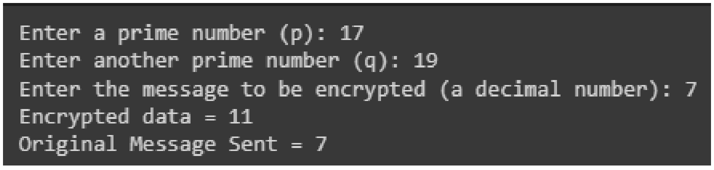
RSA –



```
import math

def gcd(a, h):
    temp = 0
    while(1):
        temp = a % h
        if temp == 0:
            return h
        a = h
        h = temp

p = int(input("Enter a prime number (p): "))
q = int(input("Enter another prime number (q): "))
msg = float(input("Enter the message to be encrypted (a decimal number): "))

n = p * q
e = 2
phi = (p - 1) * (q - 1)
```

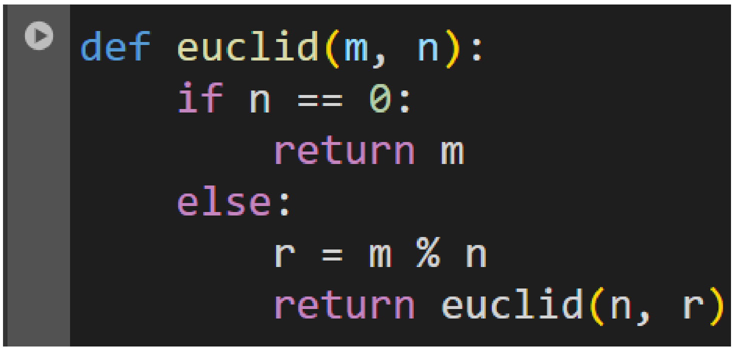


```
while e < phi:
    if gcd(e, phi) == 1:
        break
    else:
        e += 1

d = pow(e, -1, phi)

c = pow(int(msg), e, n)
print("Encrypted data =", c)

m = pow(c, d, n)
print("Original Message Sent =", m)
```

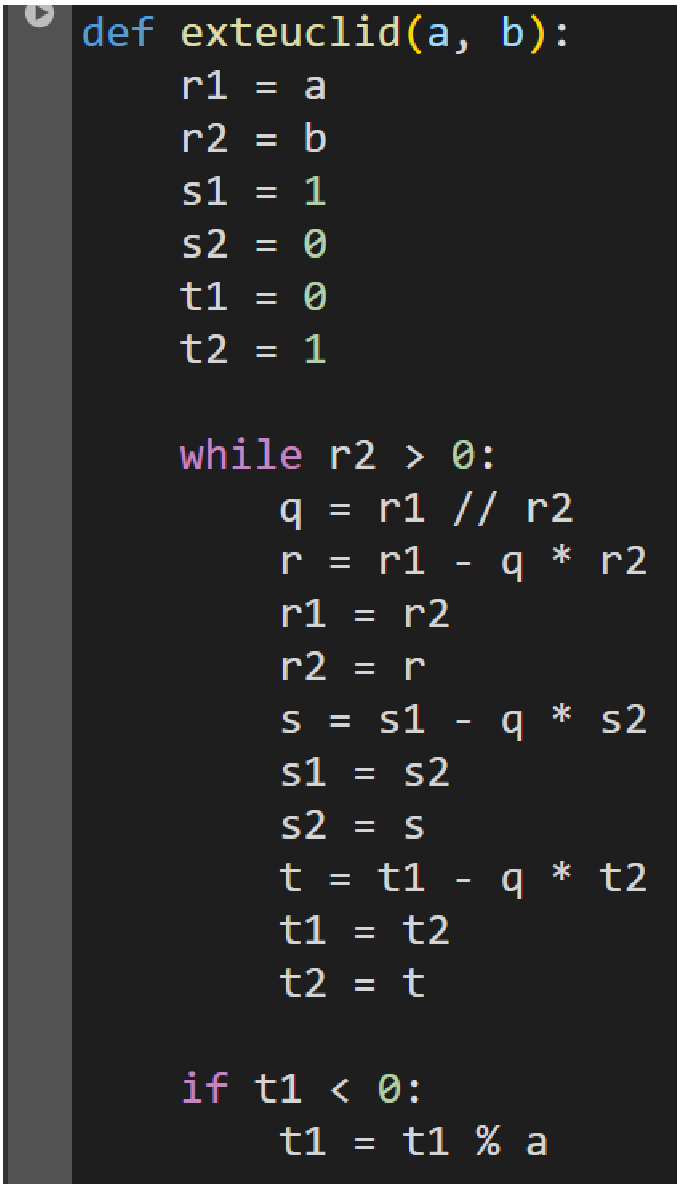


```
Enter a prime number (p): 17
Enter another prime number (q): 19
Enter the message to be encrypted (a decimal number): 7
Encrypted data = 11
Original Message Sent = 7
```

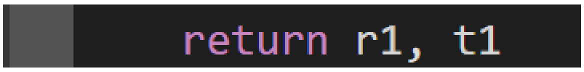
RSA – DIGITAL SIGNATURE –



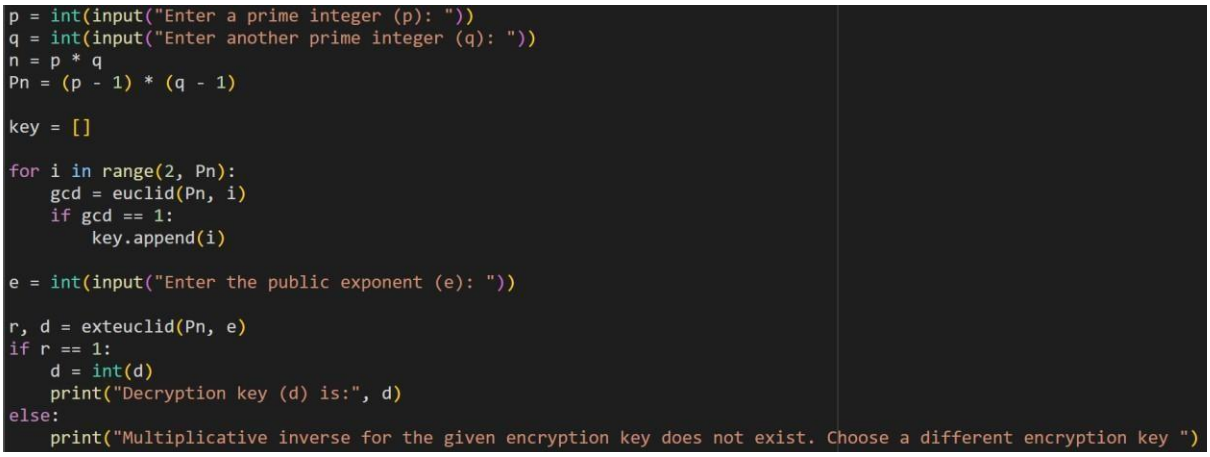
```
def euclid(m, n):
    if n == 0:
        return m
    else:
        r = m % n
        return euclid(n, r)
```



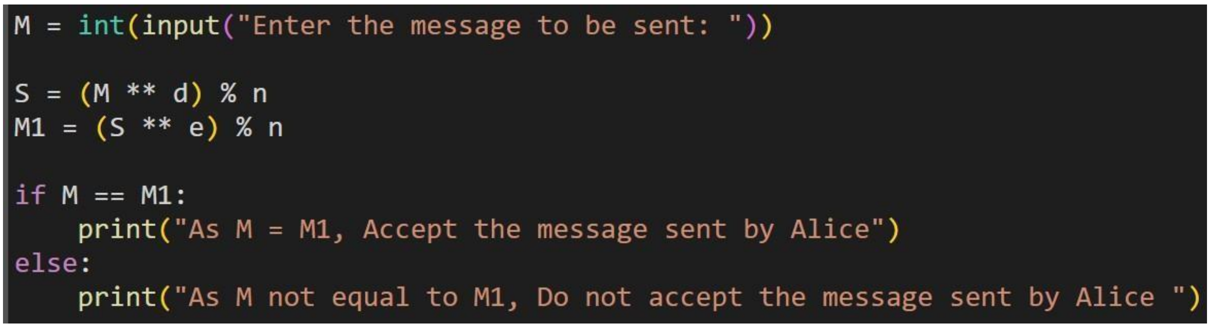
```
def exteuclid(a, b):
    r1 = a
    r2 = b
    s1 = 1
    s2 = 0
    t1 = 0
    t2 = 1

    while r2 > 0:
        q = r1 // r2
        r = r1 - q * r2
        r1 = r2
        r2 = r
        s = s1 - q * s2
        s1 = s2
        s2 = s
        t = t1 - q * t2
        t1 = t2
        t2 = t

    if t1 < 0:
        t1 = t1 % a
```



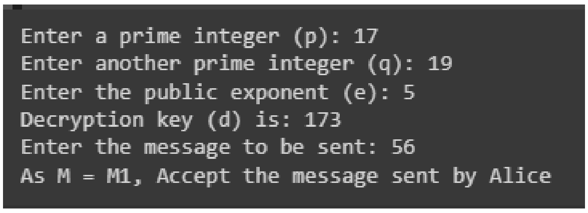
```
    return r1, t1
```



```
p = int(input("Enter a prime integer (p): "))
q = int(input("Enter another prime integer (q): "))
n = p * q
Pn = (p - 1) * (q - 1)

key = []

for i in range(2, Pn):
    gcd = euclid(Pn, i)
    if gcd == 1:
        key.append(i)

e = int(input("Enter the public exponent (e): "))

r, d = exteuclid(Pn, e)
if r == 1:
    d = int(d)
    print("Decryption key (d) is:", d)
else:
    print("Multiplicative inverse for the given encryption key does not exist. Choose a different encryption key ")
```

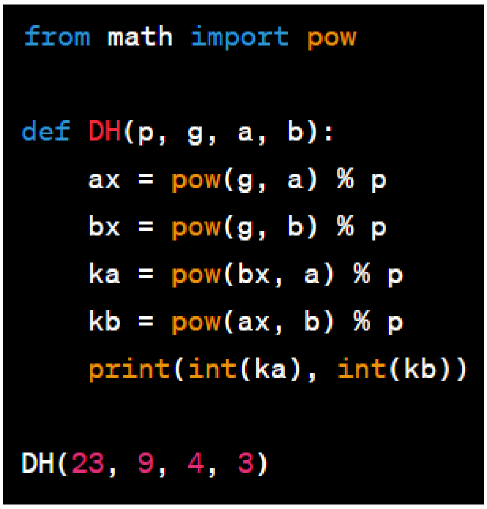


```
M = int(input("Enter the message to be sent: "))

S = (M ** d) % n
M1 = (S ** e) % n

if M == M1:
    print("As M = M1, Accept the message sent by Alice")
else:
    print("As M not equal to M1, Do not accept the message sent by Alice ")
```

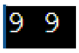


```
Enter a prime integer (p): 17
Enter another prime integer (q): 19
Enter the public exponent (e): 5
Decryption key (d) is: 173
Enter the message to be sent: 56
As M = M1, Accept the message sent by Alice
```

DIFFIE-HELLMAN KEY EXCHANGE –

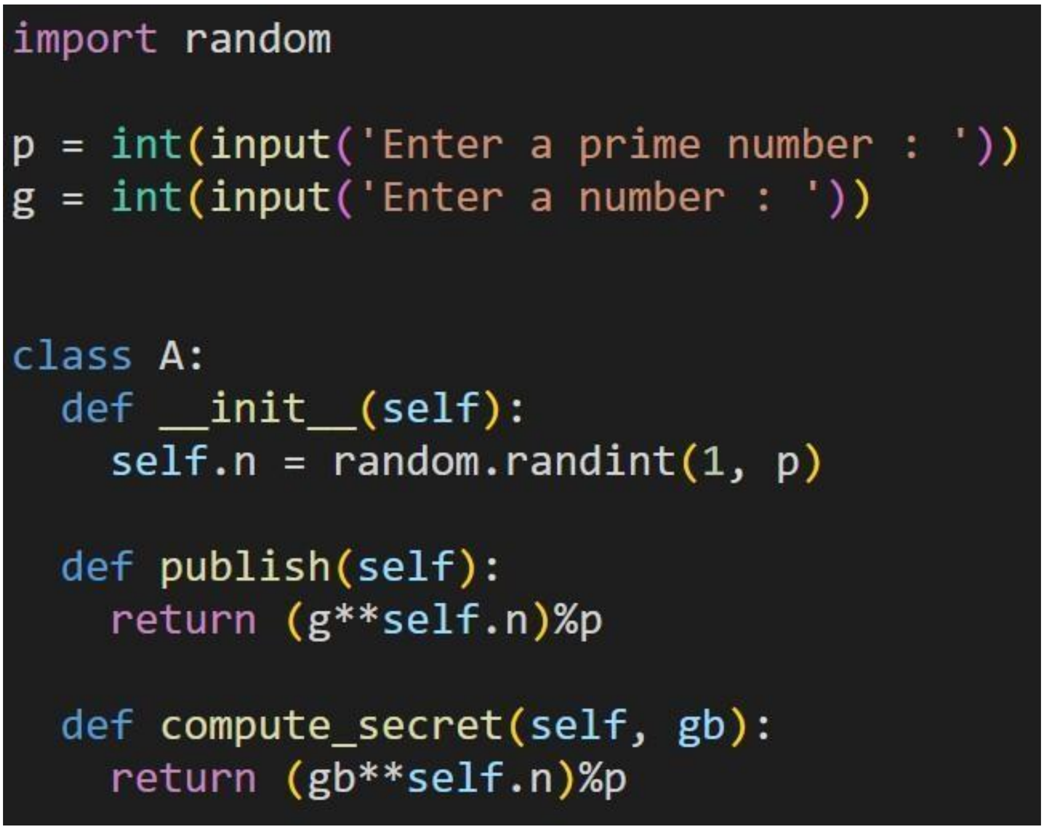


```
from math import pow

def DH(p, g, a, b):
    ax = pow(g, a) % p
    bx = pow(g, b) % p
    ka = pow(bx, a) % p
    kb = pow(ax, b) % p
    print(int(ka), int(kb))

DH(23, 9, 4, 3)
```

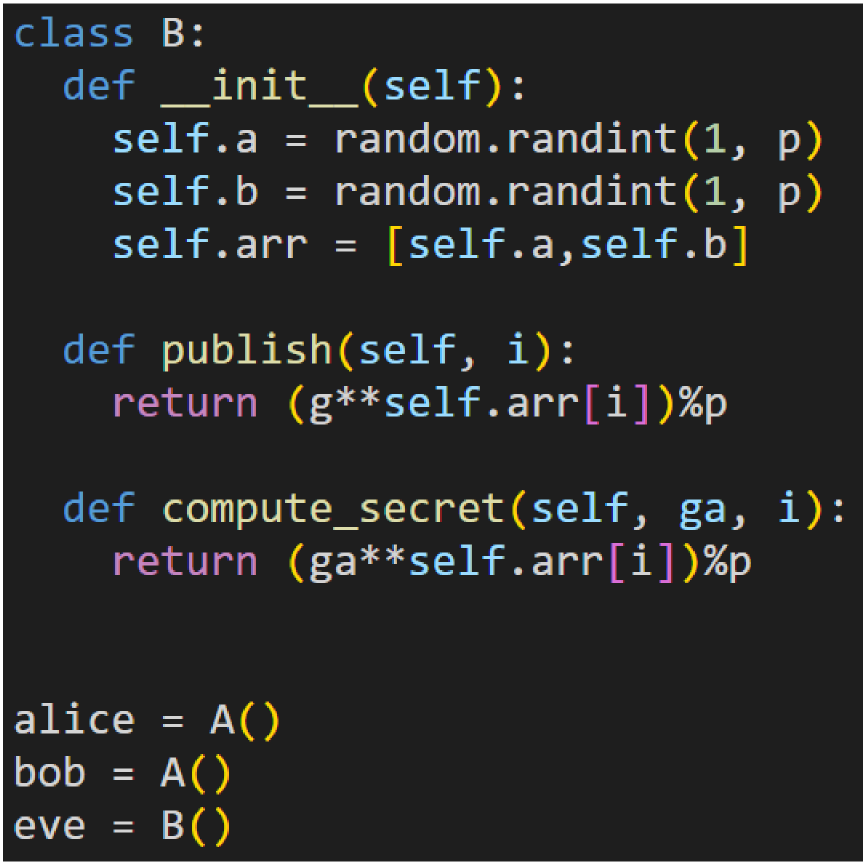


```
9 9
```

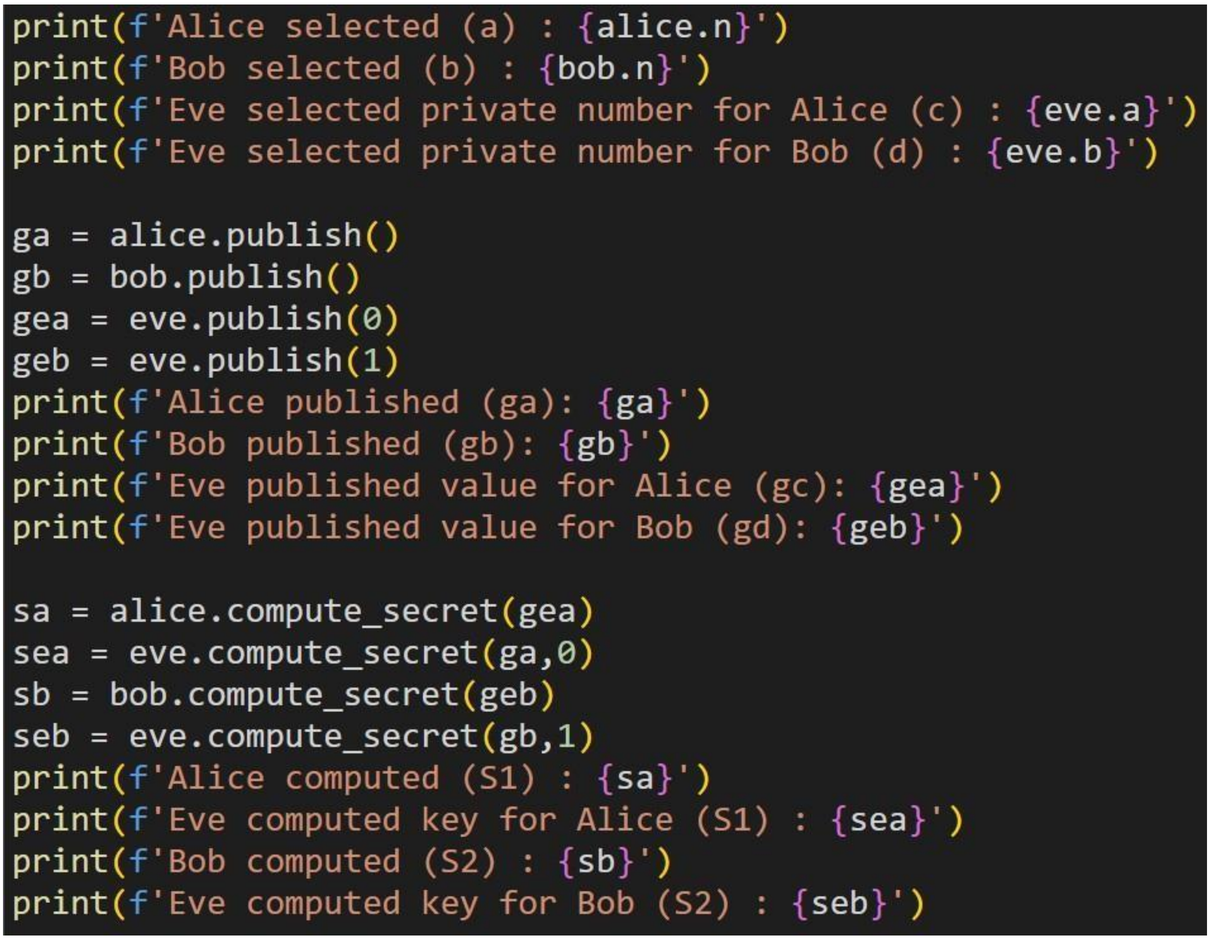
MAN IN THE MIDDLE ATTACK – DIFFIE-HELLMAN



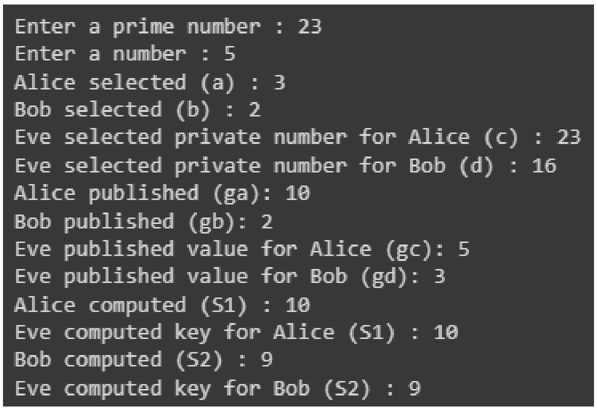
```python
import random

p = int(input('Enter a prime number : '))
g = int(input('Enter a number : '))


class A:
  def __init__(self):
    self.n = random.randint(1, p)

  def publish(self):
    return (g**self.n)%p

  def compute_secret(self, gb):
    return (gb**self.n)%p
```



```python
class B:
  def __init__(self):
    self.a = random.randint(1, p)
    self.b = random.randint(1, p)
    self.arr = [self.a,self.b]

  def publish(self, i):
    return (g**self.arr[i])%p

  def compute_secret(self, ga, i):
    return (ga**self.arr[i])%p


alice = A()
bob = A()
eve = B()
```



```python
print(f'Alice selected (a) : {alice.n}')
print(f'Bob selected (b) : {bob.n}')
print(f'Eve selected private number for Alice (c) : {eve.a}')
print(f'Eve selected private number for Bob (d) : {eve.b}')

ga = alice.publish()
gb = bob.publish()
gea = eve.publish(0)
geb = eve.publish(1)
print(f'Alice published (ga): {ga}')
print(f'Bob published (gb): {gb}')
print(f'Eve published value for Alice (gc): {gea}')
print(f'Eve published value for Bob (gd): {geb}')

sa = alice.compute_secret(gea)
sea = eve.compute_secret(ga,0)
sb = bob.compute_secret(geb)
seb = eve.compute_secret(gb,1)
print(f'Alice computed (S1) : {sa}')
print(f'Eve computed key for Alice (S1) : {sea}')
print(f'Bob computed (S2) : {sb}')
print(f'Eve computed key for Bob (S2) : {seb}')
```



```
Enter a prime number : 23
Enter a number : 5
Alice selected (a) : 3
Bob selected (b) : 2
Eve selected private number for Alice (c) : 23
Eve selected private number for Bob (d) : 16
Alice published (ga): 10
Bob published (gb): 2
Eve published value for Alice (gc): 5
Eve published value for Bob (gd): 3
Alice computed (S1) : 10
Eve computed key for Alice (S1) : 10
Bob computed (S2) : 9
Eve computed key for Bob (S2) : 9
```