1, 2

```
s=input()
print('xor with 0')
for i in s:#1st
    print(chr(ord(i)^0),end='')


print('xor with 127')
for i in s:#2nd
    print(chr(ord(i)^127),end='')
print('and with 127')
for i in s:
    print(chr(ord(i)&127),end='')
print('or with 127')
for i in s:
    print(chr(ord(i)|127),end='')
```

3. caeser cipher

```
def encrypt(string,s):
    cipher=''
    for i in string:
        if i==' ':
            cipher+=' '
        elif i.isupper():
            cipher+=chr((ord(i)+s-65)%26+65)
        else:
            cipher+=chr((ord(i)+s-97)%26+97)
    return cipher

s=int(input("Enter the shift: "))
string=input("Enter the string: ")
cipher = encrypt(string,s)
print("The encrypted text is : ",cipher)
 #for decryption just change the shift number sign to minus
print('The decrypted text is: ',encrypt(cipher,-s))
```

4. substitution cipher

```
def encrypt(plain,key):
    ciper=''
    for i in range(len(plain)):
        if plain[i]==' ':
                ciper+=' '
                continue
            ciper+=chr((ord(plain[i])+ord(key[i])-2*97)%26+97)
        return ciper
def decrypt(cipher,key):
    plain=''
    for i in range(len(cipher)):
        if cipher[i]==' ':
            plain+=' '
            continue
        plain+=chr((ord(cipher[i])-ord(key[i]))%26+97)
    return plain


plaintext=input()
key=input()
cipher=encrypt(plaintext.lower(),key)
print('Encrypted text:',cipher)
print('Decrypted text:',decrypt(cipher.lower(),key))
```

5. playfair cipher

```
def getindex(a):
    for i in range(5):
        for j in range(5):
            if a in matrix[i][j]:
                return [i,j]

def is_same_row(x,y):
    return x[0]==y[0]
def is_same_col(x,y):
    return x[1]==y[1]

key='bhai'
matrix=[['b','h','a','ij','c'],['d','e','f','g','k'],
        ['l','m','n','o','p'],['q','r','s','t','u'],['v','w','x','y','z']]
plaintext='bmfgqw' #hlgkrv
cipher=""
for i in range(0,len(plaintext),2):
    a,b=plaintext[i],plaintext[i+1]
    x=getindex(a)
    y=getindex(b)
    if is_same_row(x,y):
        cipher+=matrix[x[0]][(x[1]+1)%5]+matrix[y[0]][(y[1]+1)%5]   #turn + to -
for decryption
    elif is_same_col(x,y):
        cipher+=matrix[(x[0]+1)%5][x[1]]+matrix[(y[0]+1)%5][y[1]]   #turn + to -
for decryption
    else:
        cipher+=matrix[x[0]][y[1]]+matrix[y[0]][x[1]]

print(cipher)

#decryption
decipher=""
for i in range(0,len(cipher),2):
    a,b=cipher[i],cipher[i+1]
    x=getindex(a)
    y=getindex(b)
    if is_same_row(x,y):
        decipher+=matrix[x[0]][(x[1]-1)%5]+matrix[y[0]][(y[1]-1)%5]
    elif is_same_col(x,y):
        decipher+=matrix[(x[0]-1)%5][x[1]]+matrix[(y[0]-1)%5][y[1]]
    else:
        decipher+=matrix[x[0]][y[1]]+matrix[y[0]][x[1]]

print(decipher)
```

6. hill cipher

```
import numpy as np
def encrypt(plaintext,key):
    cipher=""
    if len(plaintext)%2:
        plaintext+='x'
    for i in range(0,len(plaintext),len(key)):
        pair=[]
        for j in range(len(key)):
            pair.append(ord(plaintext[i+j])-97)
        result = np.dot(key,pair)%26
        for j in range(len(key)):
            cipher += chr((result[j])+97)

    return cipher
```

```python
def get_inverse_mod(a,m):
    for i in range(1,m):
        if (a*i)%m==1:
            return i
    return None

def decrypt(cipher,key):
    key=np.array(key)
    det=int(np.linalg.det(key))
    det = det%26
    inv_det=get_inverse_mod(det,26)
    print(inv_det)
    if inv_det is None:
        return ValueError("Key matrix is not invertible module 26")
    inverse_matrix = np.linalg.inv(key)
    adjoint_matrix = inverse_matrix * det * inv_det
    adjoint_matrix %= 26
    key=np.round(adjoint_matrix,0).astype(int)
    key %= 26
    decipher=''
    plain=''
    for i in range(0,len(cipher),len(key)):
        pair=[]
        for j in range(len(key)):
            pair.append(ord(cipher[i+j])-97)
        result = np.dot(key,pair)%26
        for j in range(len(key)):
            plain += chr((result[j])+97)
    return plain

keymatrix=[[3,2],[5,7]]
plaintext="avasaramaaa"
ciphertext=encrypt(plaintext,keymatrix)
print("cipher text:",ciphertext)
decrypted_text=decrypt(ciphertext,keymatrix)
print("decrypted: ",decrypted_text)
```

7. des

```python
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad,unpad

def encryption(plain,key):
    cipher = DES.new(key,DES.MODE_ECB)
    padded_text = pad(plain,DES.block_size)
    encrypted=cipher.encrypt(padded_text)
    return encrypted

def decryption(cipher,key):
    decipher=DES.new(key,DES.MODE_ECB)
    unpadded=decipher.decrypt(cipher)
    decrypted=unpad(unpadded,DES.block_size)
    return decrypted

key=get_random_bytes(8)

plaintext=b"hello world"
ciphertext=encryption(plaintext,key)
print('encrypted text:', ciphertext,'\n')
decrypted=decryption(ciphertext,key)
print("decrypted text: ",decrypted.decode('utf-8'))
```

8. aes

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

key=get_random_bytes(16)

cipher = AES.new(key, AES.MODE_EAX)

data=b"hello world!"
nonce = cipher.nonce
ciphertext=cipher.encrypt(data)

print("Cipher text: ",ciphertext)

decipher = AES.new(key,AES.MODE_EAX, nonce=nonce)

plaintext=decipher.decrypt(ciphertext)

print("Decrypted: ",plaintext.decode('utf-8'))
```

9. blowfish

```
from Crypto.Cipher import Blowfish
from Crypto.Random import get_random_bytes
```

```python
from Crypto.Util.Padding import pad, unpad

def encryption(key,plaintext):
    cipher = Blowfish.new(key,Blowfish.MODE_ECB)
    padded_text = pad(plaintext,Blowfish.block_size)
    ciphertext=cipher.encrypt(padded_text)
    return ciphertext

def decryption(key,ciphertext):
    decipher=Blowfish.new(key,Blowfish.MODE_ECB)
    unpadded=decipher.decrypt(ciphertext)
    decrypted = unpad(unpadded,Blowfish.block_size)
    return decrypted

key = get_random_bytes(8)
plaintext=b'endhi bro idhi'
encrypted_text=encryption(key,plaintext)
print('Encrypted text: ', encrypted_text,'\n')
decrypted_text=decryption(key,encrypted_text)
print('Decrypted text:', decrypted_text.decode('utf-8'))
```

10. RC4

```python
from Crypto.Cipher import ARC4
from Crypto.Random import get_random_bytes

def encryption(key,plaintext):
    cipher=ARC4.new(key)
    ciphertext=cipher.encrypt(plaintext.encode('utf-8'))
    return ciphertext

def decryption(key,ciphertext):
    decipher=ARC4.new(key)
    plaintext=decipher.decrypt(ciphertext)
    return plaintext.decode('utf-8')

key=get_random_bytes(16)
plaintext = 'nen potha'
ciphertext=encryption(key,plaintext)
print("Ciphertext: ",ciphertext)
decryptedtext=decryption(key,ciphertext)
print("decrypted: ",decryptedtext)
```

11. RSA

```python
import math
def gcd(a,h):
    while h:
        a,h=h,a%h
    return a

p=7
q=11
n=p*q
phi=(p-1)*(q-1)
e=2
d=2
while e<phi:
    if gcd(e,phi)==1:
        break
    e+=1
while d<phi:
    if (((d*e)%phi)==1):
        break
    d+=1
print(e,n)
print(d,n)
msg=75
print(msg)
c=(msg**e)%n
print(c)
m=(c**d)%n
print(m)
```

12. Diffie hellman

```
#deffie hellman
import math
#check for prime
def prime(n):
    if n==1:
        return False
    if n==2 or n==3:
        return True
    for i in range(2,int(math.sqrt(n))+1):
        if n%i==0:
            return False
    return True

q = int(input())

if not prime(q):
    print("Enter a prime number!!!")
#primtive root
def primitive(n):

    for i in range(1,n):
        x=i
        s = set()
        for j in range(1,n):
            s.add(math.fmod(x**j,n))
        if len(s)==n-1:
            return x
    return -1

print(primitive(q))

alpha = primitive(q)

#public key of A
Xa = 3
Ya = math.fmod(alpha**Xa,q)

#public key of B
Xb = 5
Yb = math.fmod(alpha**Xb,q)

#secret keys
ka = math.fmod(Yb**Xa,q)
kb = math.fmod(Ya**Xb,q)

print(ka,kb)
```

13. Simple columner

```
plain_text = "helloworld"

key = "head"
s = [i for i in key]
s.sort()
k1 = [s.index(i)+1 for i in key]
# print(k1)
l = len(k1)
t = []
idx = 0
while idx<len(plain_text):
    if idx+l>len(plain_text):
        t.append(list(plain_text[idx:]))
    else:
        t.append(list(plain_text[idx:idx+l]))
    idx+=l
if len(t[-1])<l:
    for i in range(l-len(t[-1])):
        t[-1].append("_")

# print(t)
tr = [["_" for i in range(len(t))]for j in range(len(t[0]))]
```

```
for i in range(len(tr)):
    for j in range(len(tr[0])):
        tr[i][j] = t[j][i]
# print(tr)
cipher = ""
for i in range(1,len(k1)+1):
    cipher+="".join(tr[k1.index(i)])
print(cipher)

print(t)
ans = ""
for i in t:
    ans+="".join(i)
print(ans)
```

14. euclidiean and adv Euclidean

```
def gcd(a,b):
    while b:
        a,b=b,a%b
    return a

a=10
b=40
print(gcd(a,b))


def gcdExtended(a,b):
    if a==0:
        return b,0,1
    gcd,x1,y1=gcdExtended(b%a,a)
    x=y1-(b//a)*x1
    y=x1
    return gcd,x,y
a=10
b=40
print(gcdExtended(a,b))
```

15. md5

#builtin vala

import hashlib

text = "Your text goes here"

# Convert the text to bytes

text_bytes = text.encode('utf-8')

# Create an MD5 hash object

md5_hash = hashlib.md5()

# Update the hash object with the text bytes

md5_hash.update(text_bytes)

# Get the MD5 digest in hexadecimal format

digest = md5_hash.hexdigest()

print("MD5 Digest:", digest)

#not builtin vala

```
import struct
def left_rotate(x, c):
    return (x << c) | (x >> (32 - c))
def F(x, y, z):
    return (x & y) | ((~x) & z)
def G(x, y, z):
    return (x & z) | (y & (~z))
def H(x, y, z):
    return x ^ y ^ z
def I(x, y, z):
    return y ^ (x | (~z))
def md5(data):
    A = 0x67452301
    B = 0xEFCDAB89
    C = 0x98BADCFE
    D = 0x10325476
    K = [0xD76AA478, 0xE8C7B756, 0x242070DB, 0xC1BDCEEE,
         0xF57C0FAF, 0x4787C62A, 0xA8304613, 0xFD469501,
         0x698098D8, 0x8B44F7AF, 0xFFFF5BB1, 0x895CD7BE,
         0x6B901122, 0xFD987193, 0xA679438E, 0x49B40821,
         0xF61E2562, 0xC040B340, 0x265E5A51, 0xE9B6C7AA,
         0xD62F105D, 0x02441453, 0xD8A1E681, 0xE7D3FBC8,
         0x21E1CDE6, 0xC33707D6, 0xF4D50D87, 0x455A14ED,
         0xA9E3E905, 0xFCEFA3F8, 0x676F02D9, 0x8D2A4C8A,
         0xFFFA3942, 0x8771F681, 0x6D9D6122, 0xFDE5380C,
         0xA4BEEA44, 0x4BDECFA9, 0xF6BB4B60, 0xBEBFBC70,
         0x289B7EC6, 0xEAA127FA, 0xD4EF3085, 0x04881D05,
         0xD9D4D039, 0xE6DB99E5, 0x1FA27CF8, 0xC4AC5665,
         0xF4292244, 0x432AFF97, 0xAB9423A7, 0xFC93A039,
         0x655B59C3, 0x8F0CCC92, 0xFFEFF47D, 0x85845DD1,
         0x6FA87E4F, 0xFE2CE6E0, 0xA3014314, 0x4E0811A1,
         0xF7537E82, 0xBD3AF235, 0x2AD7D2BB, 0xEB86D391]
    bit_len = len(data) * 8
    data += b'\x80'
    while (len(data) + 8) % 64 != 0:
        data += b'\x00'
    data += struct.pack('<Q', bit_len)
    for i in range(0, len(data), 64):
        chunk = data[i:i + 64]
        M = struct.unpack('<16I', chunk)
        AA, BB, CC, DD = A, B, C, D
        for j in range(16):
            A = B + left_rotate((A + F(B, C, D) + M[j] + K[j]), 7)
            A, B, C, D = D, A, B, C
        for j in range(16):
            k = (1 + 5 * j) % 16
            A = B + left_rotate((A + G(B, C, D) + M[k] + K[j + 16]), 12)
            A, B, C, D = D, A, B, C
        for j in range(16):
            k = (5 + 3 * j) % 16
            A = B + left_rotate((A + H(B, C, D) + M[k] + K[j + 32]), 17)
            A, B, C, D = D, A, B, C
        for j in range(16):
            k = (7 * j) % 16
            A = B + left_rotate((A + I(B, C, D) + M[k] + K[j + 48]), 22)
            A, B, C, D = D, A, B, C

        A, B, C, D = (A + AA) & 0xFFFFFFFF, (B + BB) & 0xFFFFFFFF, (C + CC) & 0xFFFFFFFF,
(D + DD) & 0xFFFFFFFF
    result = ''.join(f'{x:08x}' for x in (A, B, C, D))
    return result
message = input("Enter the string: ")
hashed = md5(message.encode('utf-8'))
print("MD5 Hash:", hashed)
```

16. sha

#builtin vala

import hashlib

text = "Your text goes here"

# Convert the text to bytes

text_bytes = text.encode('utf-8')

# Create a SHA-1 hash object

sha1_hash = hashlib.sha1()

# Update the hash object with the text bytes

sha1_hash.update(text_bytes)

# Get the SHA-1 digest in hexadecimal format

digest = sha1_hash.hexdigest()

print("SHA-1 Digest:", digest)

#notbuiltin vala

```
def sha1(data):
    bytes = ""
    h0 = 0x67452301
    h1 = 0xEFCDAB89
    h2 = 0x98BADCFE
    h3 = 0x10325476
    h4 = 0xC3D2E1F0
    for n in range(len(data)):
        bytes+='{0:08b}'.format(ord(data[n]))
    bits = bytes+"1"
    pBits = bits
    while len(pBits)%512 != 448:
        pBits+="0"
    pBits+='{0:064b}'.format(len(bits)-1)
    def chunks(l, n):
        return [l[i:i+n] for i in range(0, len(l), n)]
    def rol(n, b):
        return ((n << b) | (n >> (32 - b))) & 0xffffffff
    for c in chunks(pBits, 512):
        words = chunks(c, 32)
        w = [0]*80
        for n in range(0, 16):
            w[n] = int(words[n], 2)
        for i in range(16, 80):
            w[i] = rol((w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16]), 1)
        a = h0
        b = h1
        c = h2
        d = h3
        e = h4
        for i in range(0, 80):
            if 0 <= i <= 19:
                f = (b & c) | ((~b) & d)
                k = 0x5A827999
            elif 20 <= i <= 39:
                f = b ^ c ^ d
                k = 0x6ED9EBA1
            elif 40 <= i <= 59:
                f = (b & c) | (b & d) | (c & d)
                k = 0x8F1BBCDC
            elif 60 <= i <= 79:
                f = b ^ c ^ d
                k = 0xCA62C1D6
            temp = rol(a, 5) + f + e + k + w[i] & 0xffffffff
            e = d
            d = c
            c = rol(b, 30)
            b = a
            a = temp
        h0 = h0 + a & 0xffffffff
        h1 = h1 + b & 0xffffffff
        h2 = h2 + c & 0xffffffff
        h3 = h3 + d & 0xffffffff
        h4 = h4 + e & 0xffffffff
    return '%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4)
str=input("Enter the string: ")
print (sha1(str))
```