Assignment 1:

```
#program to store string "helloworld" and performs logical operations
#like AND,OR XOR
#between each character of string and 127.

str1="Helloworld"
for ch in str1:
   s1=(ord(ch)&127)
   s2=(ord(ch)|127)
   s3=(ord(ch)^127)

   print(ch+" & 127 :"+chr(s1)+'\t'+
        ch+" | 127 :"+chr(s2)+'\t'+
        ch+" ^ 127 :"+chr(s3)+'\t')
```

Output:

```
H & 127 :H      H | 127 :      H ^ 127 :7
e & 127 :e      e | 127 :      e ^ 127 :
l & 127 :l      l | 127 :      l ^ 127 :
l & 127 :l      l | 127 :      l ^ 127 :
o & 127 :o      o | 127 :      o ^ 127 :
w & 127 :w      w | 127 :      w ^ 127
o & 127 :o      o | 127 :      o ^ 127 :
 & 127 :r       r | 127 :      r ^ 127 :
l & 127 :l      l | 127 :      l ^ 127 :
d & 127 :d      d | 127 :      d ^ 127 :
```

Assignment 2:

```
import math
plaintext="transposition technique using python"
key=8
ciphertext=['']*key
for colum in range(key):
    pointer=colum
    while pointer<len(plaintext):
     ciphertext[colum]+=plaintext[pointer]
     print(ciphertext)
     pointer+=key

cipher=''.join(ciphertext)
print(cipher)

numOfColumns = math.ceil(len(cipher) / key)
print(numOfColumns )
numOfRows = key
numOfShadedBoxes = (numOfColumns * numOfRows) - len(cipher)

pt = [''] * numOfColumns
col=0
row=0
for sym in cipher:
   pt[col]+=sym
   col+=1
   if (col == numOfColumns) or (col == numOfColumns - 1 and row >= numOfRows -
numOfShadedBoxes):
      col=0
      row=row+1
```

```
    print(pt)
print(''.join(pt))
```

Output:

```
>>> %Run Assignment2_SourceFile.py
['t', '', '', '', '', '', '', '']
['ti', '', '', '', '', '', '', '']
['tic', '', '', '', '', '', '', '']
['ticu', '', '', '', '', '', '', '']
['ticut', '', '', '', '', '', '', '']
['ticut', 'r', '', '', '', '', '', '']
['ticut', 'rt', '', '', '', '', '', '']
['ticut', 'rth', '', '', '', '', '', '']
['ticut', 'rths', '', '', '', '', '', '']
['ticut', 'rthsh', '', '', '', '', '', '']
['ticut', 'rthsh', 'a', '', '', '', '', '']
['ticut', 'rthsh', 'ai', '', '', '', '', '']
['ticut', 'rthsh', 'ain', '', '', '', '', '']
['ticut', 'rthsh', 'aini', '', '', '', '', '']
['ticut', 'rthsh', 'ainio', '', '', '', '', '']
['ticut', 'rthsh', 'ainio', 'n', '', '', '', '']
['ticut', 'rthsh', 'ainio', 'no', '', '', '', '']
['ticut', 'rthsh', 'ainio', 'noi', '', '', '', '']
['ticut', 'rthsh', 'ainio', 'noin', '', '', '', '']
['ticut', 'rthsh', 'ainio', 'noinn', '', '', '', '']
['ticut', 'rthsh', 'ainio', 'noinn', 's', '', '', '']
['ticut', 'rthsh', 'ainio', 'noinn', 'sn', '', '', '']
['ticut', 'rthsh', 'ainio', 'noinn', 'snq', '', '', '']
```

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', '', '', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p', '', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p ', '', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u', '', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', '', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', 'o', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', 'ot', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', 'ote', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', 'otep', '']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', 'otep', 's']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', 'otep', 'se']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', 'otep', 'se ']

['ticut', 'rthsh', 'ainio', 'noinn', 'snqg', 'p u ', 'otep', 'se y']

ticutrthshainionoinnsnqgp u otepse y

5

['t', 'i', 'c', 'u', 't']

['tr', 'it', 'ch', 'us', 'th']

['tra', 'iti', 'chn', 'usi', 'tho']

['tran', 'itio', 'chni', 'usin', 'thon']

['trans', 'ition', 'chniq', 'using', 'thon']

['transp', 'ition ', 'chniqu', 'using ', 'thon']

['transpo', 'ition t', 'chnique', 'using p', 'thon']

['transpos', 'ition te', 'chnique ', 'using py', 'thon']

transposition technique using python

Assignment 3:

```
from Crypto.Cipher import DES
def pad(text):
 n = len(text) % 8
 print(b"text to encrypt:"+text + (b' ' * n))
 return text + (b' ' * n)
key = b'hello123'
text1 = b'Python is the Best Language!'
des = DES.new(key, DES.MODE_ECB)
padded_text = pad(text1)
encrypted_text = des.encrypt(padded_text)
print(encrypted_text)
print(des.decrypt(encrypted_text))
```

Output:

```
b'text to encrypt:Python is the Best Language!    '
b'{\x01^\xfe\xd5\xd1\x8fM\x1a\xcc\xd5\xbc\x04\x1c\x0em$,\xc1\xc7w-H1\xe6>\x08%!\xab\xd0X'
b'Python is the Best Language!    '
```

Assignment 4:

```
from Crypto.Cipher import AES
key = b'6487264824evcnvk'
cipher = AES.new(key, AES.MODE_EAX)
data = "Hello welcome to P town".encode()

nonce = cipher.nonce
ciphertext = cipher.encrypt(data)

print("Plain text",data)
print("Cipher text: ",ciphertext)
cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)

plaintext = cipher.decrypt(ciphertext)
print("Plain text", plaintext)
```

Output:

```
>>> %Run AES7.py
Plain text b'Hello welcome to P town'
Cipher text:  b'\xc3\xcb1ms\xea\x15\x84\x12b\x8bc\x18\xd5\xfac\xb2\x14\xba\x9a\xbb\xe2\xef'
Plain text b'Hello welcome to P town'
```

Assignment 5:

```
import random
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def generate_primes():
    p = random.randint(100, 1000)
    while not is_prime(p):
        p = random.randint(100, 1000)
    q = random.randint(100, 1000)
    while not is_prime(q) or p == q:
        q = random.randint(100, 1000)
    return p, q

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def lcm(a, b):
    return a * b // gcd(a, b)

def generate_keys():
    p, q = generate_primes()
    n = p * q
```

```
    phi_n = lcm(p - 1, q - 1)
    e = random.randint(2, phi_n - 1)
    while gcd(e, phi_n) != 1:
        e = random.randint(2, phi_n - 1)
    d = pow(e, -1, phi_n)
    return (e, n), (d, n)

def encrypt(message, public_key):
    e, n = public_key
    return pow(message, e, n)

def decrypt(ciphertext, private_key):
    d, n = private_key
    return pow(ciphertext, d, n)
# Example usage
public_key, private_key = generate_keys()
message = 123
ciphertext = encrypt(message, public_key)
plaintext = decrypt(ciphertext, private_key)
print(f"Public key: {public_key}")
print(f"Private key: {private_key}")
print(f"Message: {message}")
print(f"Ciphertext: {ciphertext}")
print(f"Plaintext: {plaintext}")
```

Output:

```
Public key: (52189, 239651)
Private key: (22213, 239651)
Message: 123
Ciphertext: 233934
Plaintext: 123
```

Assignment 6:

```
import math
n=int(input("n = "))
g=int(input("g = "))
for i in range(2,n):
    if(n%i==0):
        print("Invalid")
        break
for i in range(2,g):
    if(g%i==0):
        print("Invalid")
        break

x=int(input("No. selected by Alice: "))
y=int(input("No. selected by Bob: "))

A=(math.pow(g,x))%n
B=(math.pow(g,y))%n

print("\nA=",A)
print("B=",B)

k1=(math.pow(B,x))%n
k2=(math.pow(A,y))%n
print("\nk1=",k1)
print("k2=",k2)
if(k1==k2):
    print("Algorithm is correct")
else:
    print("Algorithm is wrong")
```

Output:

%Run 'Python source file.py'

n = 23

g = 5

No. selected by Alice: 4

No. selected by Bob: 3

A= 4.0

B= 10.0

k1= 18.0

k2= 18.0

Algorithm is correct