```python
def ceaser_encrypt(text, n):
    ans = ""
    for ch in text:
        if ch.isupper():
            ans += chr((ord(ch)-65 + n) % 26 + 65)
        elif ch.islower():
            ans += chr((ord(ch)-97 + n) % 26 + 97)
        else:
            ans += ch
    return ans

def ceaser_decrypt(cipher, n):
    return ceaser_encrypt(text, -n)

if __name__ == "__main__":
    text = input("Enter the text to encrypt: ")
    n = int(input("Enter the shift value: "))
    encrypted = ceaser_encrypt(text, n)
    print("Encrypted text: ", encrypted)
    print("Decrypted text: ", ceaser_decrypt(encrypted, n))
```

```python
def ceaser_encrypt(text, key):
    ans = ""
    shift = ord(key.lower()) - 97

    for ch in text:
        if ch.isupper():
            ans += chr((ord(ch) + shift - 65) % 26 + 65)
        elif ch.islower():
            ans += chr((ord(ch) + shift - 97) % 26 + 97)
        else:
            ans += ch
    return ans

text = input("Enter text to encrypt: ")
key = input("Enter the key: ")

if not (key.isalpha() and len(key) == 1):
    print("Invalid key")
else:
    print("encrytped: ", ceaser_encrypt(text, key))
```

```python
def StringEncryption(text, key):
    cipherText = ""
    cipher = []
    for i in range(len(key)):
        shift = (ord(text[i]) - ord('A') + ord(key[i]) - ord('A'))
        cipher.append(shift)
    for val in cipher:
        cipherText += chr(val + ord('A'))
    return cipherText
plainText = "HelloTYCS"
key = "MONEYBANK"
encryptedText = StringEncryption(plainText.upper(), key.upper())
print("cipherText: ", encryptedText)
```

```python
string = input("Enter a string: ")

def RailFence(text):
    result = ""
    for i in range(len(string)):
        if(i % 2 == 0):
            result += string[i]
    for i in range(len(string)):
        if(i % 2 != 0):
            result += string[i]
    return result

print(RailFence(string))
```

```python
import math

def encrypt_columnar(message, key):
    msg = message.replace(" ", "")
    col = len(key)
    row = math.ceil(len(msg)/col)
    pad = row * col - len(msg)
    msg += '_' * pad

    matrix = [list(msg[i:i+col]) for i in range(0, len(msg), col)]
    key_order = sorted([(k,i) for i,k in enumerate(key)])

    cipher = ""
    for k, col_idx in key_order:
        for r in range(row):
            cipher += matrix[r][col_idx]
    return cipher

msg = "network security"
key = "tycs"
print("Original:", msg)
print("Key:", key)
print("Encryption:", encrypt_columnar(msg, key))
```

```python
import math

p, q = 3, 7
n, phi = p * q, (p - 1) * (q - 1)
print("n =", n)

e = next(i for i in range(2, phi) if math.gcd(i, phi) == 1)
print("e =", e)

k, d = 2, ((2 * phi) + 1) / e
print("d =", d)

print(f"public key: {e},{n}")
print(f"private key: {d},{n}")

msg = 11
print("original message:", msg)

c = math.fmod(pow(msg, e), n)
print("encrypted message:", c)

M = math.fmod(pow(c, d), n)
print("decrypted message:", M)
```

```python
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
data = datasets.load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
model = DecisionTreeClassifier(criterion="entropy")
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Decision Tree Structure:")
print(export_text(model, feature_names=data.feature_names))
print("\nAccuracy of the Decision Tree model:", round(accuracy * 100, 2), "%")
```

```python
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256
from Crypto import Random

key = RSA.generate(2048, Random.new().read)
private_key = key.export_key()
public_key = key.publickey().export_key()

message = "Hello World"

hashed = SHA256.new(message.encode())

signer = PKCS1_v1_5.new(RSA.import_key(private_key))
signature = signer.sign(hashed)

verifier = PKCS1_v1_5.new(RSA.import_key(public_key))
is_valid = verifier.verify(hashed, signature)

print("Digital Signature:", signature)
print("Is the signature valid?", is_valid)
```

```
from random import randint

if __name__ == '__main__':
    P = 23
    G = 9

    print('The value of P is: %d' % (P))
    print('The value of G is: %d' % (G))

    a = 4
    print('Secret number for Alice is: %d' % (a))
    x = int(pow(G, a, P))

    b = 3
    print('Secret number for Bob: %d' % (b))
    y = int(pow(G, b, P))

    ka = int(pow(y, a, P))
    kb = int(pow(x, b, P))

    print('Secret key for Alice is: %d' % (ka))
    print('Secret key for Bob is: %d' % (kb))
```