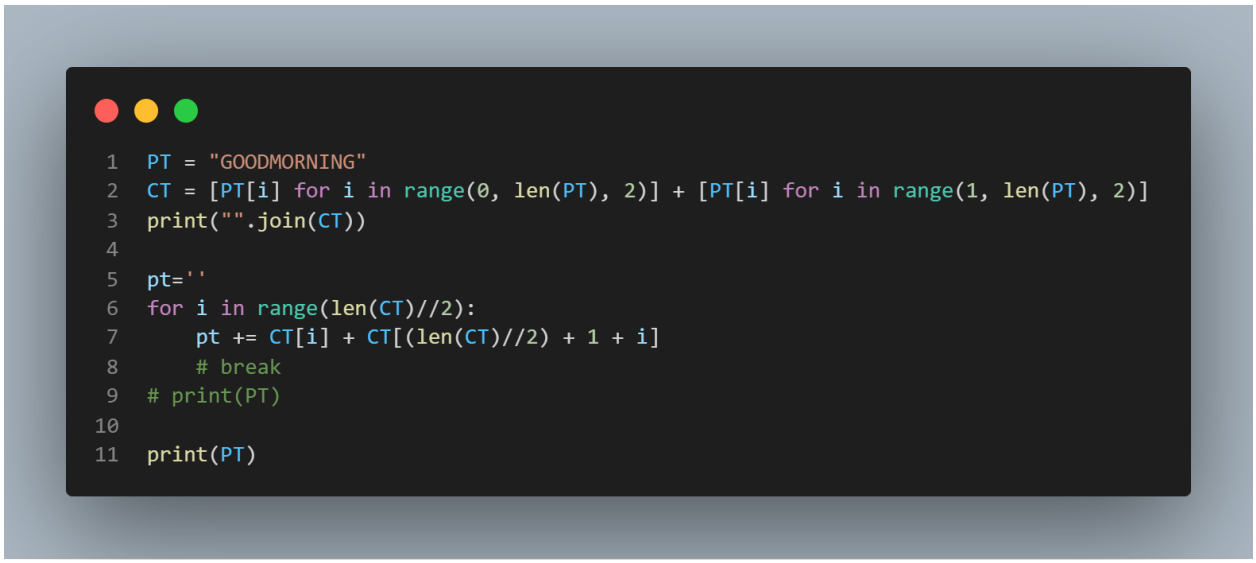
Hill Cip

```python
import numpy as np
from string import ascii_uppercase

def modInverse(a, m):
    for x in range(1, m):
        if (((a%m) * (x%m)) % m == 1):
            return x
    return -1

K = np.array([
        [6, 24, 1],
        [13, 16, 10],
        [20, 17, 15]
    ])
P = np.array([[0], [2], [19]])

arr = np.matmul(K, P) % 26

#encryption

for i in arr:
    print(ascii_uppercase[i[0]])
inv = np.round(np.linalg.inv(K) * np.linalg.det(K) * modInverse(np.linalg.det(K), 26)) % 26
print(inv)
arr1 = np.matmul(inv, arr) % 26
# arr1 = P
# decryption
for i in arr1:
    print(ascii_uppercase[int(i[0])])
```

```python
LETTERS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
translated = []
message = "MOTH"
key = "ASHW"
mode="encrypt"
keyIndex = 0 # 3
message = message.upper()
key = key.upper()

for symbol in message:
    num = LETTERS.find(symbol.upper()) # 0
    if num != -1:
        if mode == "encrypt":
            num += LETTERS.find(key[keyIndex])  # 1 + 1 2
        elif mode == "decrypt":
            num -= LETTERS.find(key[keyIndex])

        num %= len(LETTERS)

        translated.append(LETTERS[num]) # A, C, E, G

        keyIndex += 1
        if keyIndex == len(key):
            keyIndex = 0
    else:
        translated.append(symbol)
print(" ".join(translated))
```

Rail fence



```python
PT = "GOODMORNING"
CT = [PT[i] for i in range(0, len(PT), 2)] + [PT[i] for i in range(1, len(PT), 2)]
print("".join(CT))

pt=''
for i in range(len(CT)//2):
    pt += CT[i] + CT[(len(CT)//2) + 1 + i]
    # break
# print(PT)

print(PT)
```

```
<html>
   <head>
      <title>RSA Encryption</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
   </head>
   <body>
      <center>
         <h1>RSA Algorithm</h1>
         <h2>Implemented Using HTML & Javascript</h2>
         <hr>
         <table>
            <tr>
               <td>Enter First Prime Number:</td>
               <td><input type="number" value="53" id="p"></td>
            </tr>
            <tr>
               <td>Enter Second Prime Number:</td>
               <td><input type="number" value="59" id="q"></p>
               </td>
            </tr>
            <tr>
               <td>Enter the Message(cipher text):<br>[A=1, B=2,...]</td>
               <td><input type="number" value="89" id="msg"></p>
               </td>
            </tr>
            <tr>
               <td>Public Key:</td>
               <td>
                  <p id="publickey"></p>
               </td>
            </tr>
            <tr>
               <td>Exponent:</td>
               <td>
                  <p id="exponent"></p>
               </td>
            </tr>
            <tr>
               <td>Private Key:</td>
               <td>
                  <p id="privatekey"></p>
               </td>
            </tr>
            <tr>
               <td>Cipher Text:</td>
               <td>
                  <p id="ciphertext"></p>
               </td>
            </tr>
            <tr>
               <td><button onclick="RSA();">Apply RSA</button></td>
```

```
            </tr>
         </table>
      </center>
   </body>
   <script type="text/javascript">
      function RSA() {
      var gcd, p, q, no, n, t, e, i, x;
      gcd = function (a, b) { return (!b) ? a : gcd(b, a % b); };
      p = document.getElementById('p').value;
      q = document.getElementById('q').value;
      no = document.getElementById('msg').value;
      n = p * q;
      t = (p - 1) * (q - 1);
      for (e = 2; e < t; e++) {
      if (gcd(e, t) == 1) {
      break;
      }
      }
      for (i = 0; i < 10; i++) {
      x = 1 + i * t
      if (x % e == 0) {
      d = x / e;
      break;
      }
      }
      ctt = Math.pow(no, e).toFixed(0);
      ct = ctt % n;
      dtt = Math.pow(ct, d).toFixed(0);
      dt = dtt % n;
      document.getElementById('publickey').innerHTML = n;
      document.getElementById('exponent').innerHTML = e;
      document.getElementById('privatekey').innerHTML = d;
      document.getElementById('ciphertext').innerHTML = ct;
      }
   </script>
</html>
```

```
def caesar(text, key):
  result = ""
  for char in text:
    result += chr((ord(char) + key - 65) % 26 + 65)

  print(result)
  return result

pt = input()
ct = ""
key = int(input())

ct = caesar(pt, key)
caesar(ct, -key)
```

```
import numpy as np
pt="BALLOON"
key="MONARCHY"
ct=""

Alpha="ABCDEFGHIKLMNOPQRSTUVWXYZ"
keystr=""

for char in key:
    if char in Alpha:
        keystr += char
        Alpha = Alpha.replace(char,"")
keystr +=Alpha

keymat =[]
for i in range(5):
    keymat.append([])
    for j in range(5):
        keymat[i].append(keystr[0])
        keystr = keystr[1:]

for i in range (5):
    for j in range(5):
        print(keymat[i][j],end=" ")
    print("")

#digrams

dia = []
temp = pt

while temp != "":
    if temp[0] == temp[1]: # LL OO N
        dia.append(temp[0]+"X")
        temp = temp[1:]
    else:
        dia.append(temp[:2])
        temp=temp[2:]
print("Digrams",dia)
```

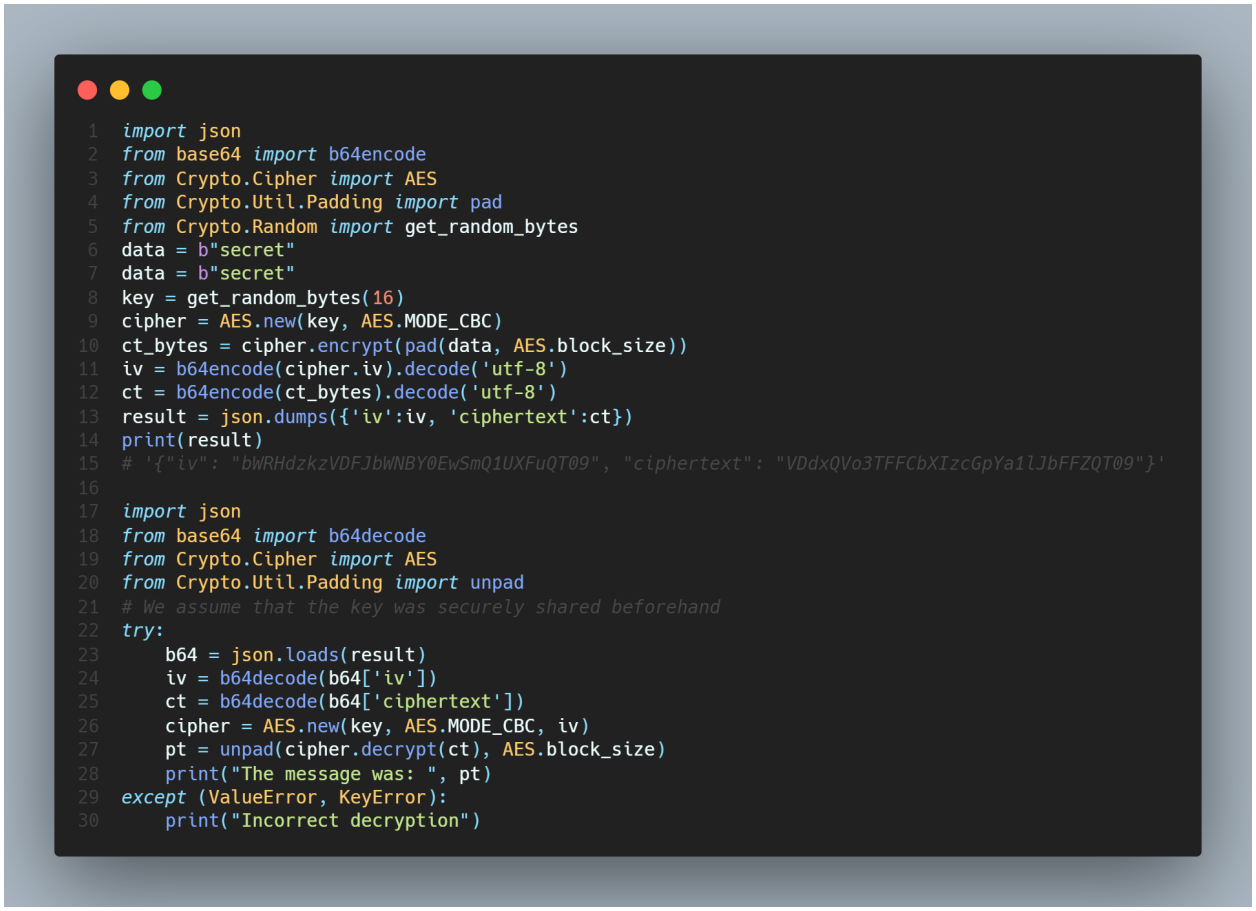
```
keyMatrixNP = np.array(keymat)
for digram in dia:
  # Step 1: Finding index of digrams

  # i and j for first letter
  a = np.where(keyMatrixNP==digram[0])

  # i and j for second letter
  b = np.where(keyMatrixNP==digram[1])

  # extracting i and j
  a = [a[0][0], a[1][0]]
  b = [b[0][0], b[1][0]]

  # Step 2: Applying encryption rules

  # if row equal
  if a[0] == b[0]:
    ct += keymat[a[0]][a[1]+1] + keymat[b[0]][b[1]+1]

  # if column equal
  elif a[1] == b[1]:
    ct += keymat[a[0]+1][a[1]] + keymat[b[0]+1][b[1]]

  # if both not equal
  else:
    ct += keymat[a[0]][b[1]] + keymat[b[0]][a[1]]

print(ct)

print(pt)
```

Playfair Cipher

```python
import numpy as np
pt="BALLOON"
key="MONARCHY"
ct=""

Alpha="ABCDEFGHIKLMNOPQRSTUVWXYZ"
keystr=""

for char in key:
    if char in Alpha:
        keystr += char
        Alpha = Alpha.replace(char,"")
keystr +=Alpha

keymat =[]
for i in range(5):
    keymat.append([])
    for j in range(5):
        keymat[i].append(keystr[0])
        keystr = keystr[1:]

for i in range (5):
    for j in range(5):
        print(keymat[i][j],end=" ")
    print("")

#digrams

dia = []
temp = pt

while temp != "":
    if temp[0] == temp[1]: # LL OO N
        dia.append(temp[0]+"X")
        temp = temp[1:]
    else:
        dia.append(temp[:2])
        temp=temp[2:]
print("Digrams",dia)
```

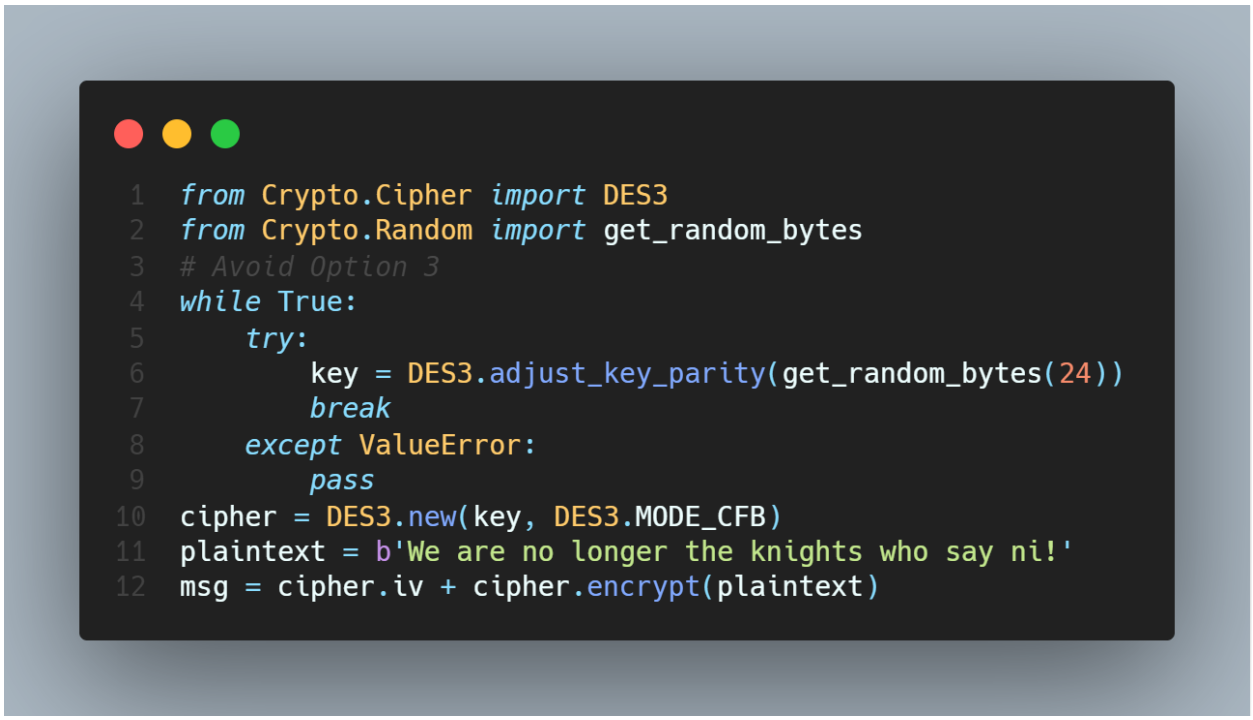
```
1
2  keyMatrixNP = np.array(keymat)
3  for digram in dia:
4    # Step 1: Finding index of digrams
5
6    # i and j for first letter
7    a = np.where(keyMatrixNP==digram[0])
8
9    # i and j for second letter
10   b = np.where(keyMatrixNP==digram[1])
11
12   # extracting i and j
13   a = [a[0][0], a[1][0]]
14   b = [b[0][0], b[1][0]]
15
16   # Step 2: Applying encryption rules
17
18   # if row equal
19   if a[0] == b[0]:
20     ct += keymat[a[0]][a[1]+1] + keymat[b[0]][b[1]+1]
21
22   # if column equal
23   elif a[1] == b[1]:
24     ct += keymat[a[0]+1][a[1]] + keymat[b[0]+1][b[1]]
25
26   # if both not equal
27   else:
28     ct += keymat[a[0]][b[1]] + keymat[b[0]][a[1]]
29
30 print(ct)
31
32 print(pt)
```

AES:



```python
import json
from base64 import b64encode
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from Crypto.Random import get_random_bytes
data = b"secret"
data = b"secret"
key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_CBC)
ct_bytes = cipher.encrypt(pad(data, AES.block_size))
iv = b64encode(cipher.iv).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
result = json.dumps({'iv':iv, 'ciphertext':ct})
print(result)
# '{"iv": "bWRHdzkzVDFJbWNBY0EwSmQ1UXFuQT09", "ciphertext": "VDdxQVo3TFFCbXIzcGpYa1lJbFFZQT09"}'

import json
from base64 import b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
# We assume that the key was securely shared beforehand
try:
    b64 = json.loads(result)
    iv = b64decode(b64['iv'])
    ct = b64decode(b64['ciphertext'])
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    print("The message was: ", pt)
except (ValueError, KeyError):
    print("Incorrect decryption")
```

DES:



```python
from Crypto.Cipher import DES3
from Crypto.Random import get_random_bytes
# Avoid Option 3
while True:
    try:
        key = DES3.adjust_key_parity(get_random_bytes(24))
        break
    except ValueError:
        pass
cipher = DES3.new(key, DES3.MODE_CFB)
plaintext = b'We are no longer the knights who say ni!'
msg = cipher.iv + cipher.encrypt(plaintext)
```