Sandbox | CMU CS Academy

```
import copy
import math

alphabet = {'A':0, 'B':1, 'C':2, 'D':3, 'E':4, 'F':5, 'G': 6, 'H':7, 'I':8, 'J':9,
'K':10, 'L':11, 'M':12, 'N':13, 'O':14, 'P':15, 'Q':16, 'R':17, 'S':18, 'T':19,
'U':20, 'V':21, 'W':22, 'X':23, 'Y':24, 'Z':25}
### To convert an inverted matrix into an inverted modular matrix, you must multipl
y the matrix
### by the modular inverse of its determinant. Using the greatest common divisor th
eorem and its
### extension, you can find the modular inverse of a number. This number when multi
plied by the determinant will produce 1 after
### applying modular division.
### Credit for algorithm: https://techiedelight.com/extended-euclidean-algorithm-im
plementation/
### Credit for concept: http://www-math.ucdenver.edu/~wcherowi/courses/m5410/exeuca
lg.html
def extendedGCD(a, b):
    if a == 0:
        return b, 0, 1
    else:
        gcd, x, y = extendedGCD(b % a, a)
        return gcd, y - (b//a) * x, x

### The determinant of a matrix can determine its invertability in a modular field.
The function implements Gaussian
### Elimination. This method manipulates a matrix using row swapping, scaling, and
row-to-row arthimatic into an upper
### triangular/row echelon form.
### Credit for algorithm: https://integratedmlai.com/find-the-determinant-of-a-matr
ix-with-pure-python-without-numpy-or-scipay/
def determinant(key):
    matrix = copy.deepcopy(key)
    n = len(matrix)
    for fd in range(n):
        for i in range(fd+1, n):
            if matrix[fd][fd] == 0:
                matrix[fd][fd] = 1*10^-18
            crScaler = matrix[i][fd] / matrix[fd][fd]
            for j in range(n):
                matrix[i][j] = matrix[i][j] - crScaler * matrix[fd][j]
    product = 1
    for i in range(n):
        product *= matrix[i][i]
    return product

def identityMatrix(length):
    identity = []
    for i in range(length):
        identity.append([])
        for j in range(length):
            if i == j:
                identity[i].append(1)

```

```
            else:
                identity[i].append(0)
    return identity

### To decrypt an encrypted message, you require the modular inverse of your key matrix.
### If password * key == encryption, then password == encryption / key. However, the Hill cipher is in module 26
### to represent the indexs of the alphabets,so division is instead multiplication with a number modular inverse
### This function applies Gauss-Jordan elimination on an augmented key matrix. It multipies the manipulated identity
### matrix by the determinant and the inverse determinant of the key and then modularly divides by 26.
### Credit for inverse algorithm: https://integratedmlai.com/matrixinverse/
### Credit for converting matrix into its modular inverse: South Florida Journal of Development, Miami, v.3, n.3 p.
### 3100-3111, may./jun., 2022. ISSN 2675-5459
def modularInverse(key, modulo):
    d = rounded(determinant(key))
    inverseD = extendedGCD(d, modulo)[1]
    n = len(key)
    I = identityMatrix(n)
    for fd in range(n):
        fdScaler = 1/ key[fd][fd]
        for j in range(n):
            key[fd][j] = key[fd][j] * fdScaler
            I[fd][j] = I[fd][j] * fdScaler
        for i in range(n):
            crScaler = key[i][fd]
            if i != fd:
                for j in range(n):
                    key[i][j] = key[i][j] - key[fd][j] * crScaler
                    I[i][j] = I[i][j] - I[fd][j] * crScaler
    for i in range(n):
        for j in range(n):
            key[i][j] *= d
            I[i][j] *= d

    for i in range(n):
        for j in range(n):
            key[i][j] = rounded((key[i][j] * inverseD) % modulo)
            I[i][j] = rounded((I[i][j] * inverseD) % modulo)
    return I

def encryption(password, key):
    isValid = determinant(key)
    if isValid == 0 or isValid % 2 == 0 or isValid % 13 == 0:
        print('Your key can not be inverted! Please restart the program!')
        app.stop()
        return None
    encryptedPassword = ''
    index = 0
    for cluster in password:
        for i in range(len(key)):
            for j in range(len(key[0])):
                index += key[i][j] * cluster[j % len(cluster)]
```

```
            index %= 26
            encryptedPassword = encryptedPassword + chr(index + 65)
            index = 0
    print('Your encrypted text is ' + encryptedPassword + '.')
    return encryptedPassword

def decryption(password, key):
    decryptedPassword = ''
    key = modularInverse(key, 26)
    index = 0
    for cluster in password:
        for i in range(len(key)):
            for j in range(len(key[0])):
                index += key[i][j] * cluster[j % len(cluster)]
            index %= 26
            decryptedPassword = decryptedPassword + chr(index + 65)
            index = 0
    print('Your decrypted text is ' + decryptedPassword + '.')
    return decryptedPassword

password = app.getTextInput('Enter in a password! (Only capital letters!)')
key = app.getTextInput('Enter in a key! (Only capital letters!)')
password = password.upper()
key = key.upper()
print('Your password/message is ' + password + '.')
print('Your key is ' + key + '.')

lengthMatrix = rounded(math.sqrt(len(key)))
key = [alphabet[letter] for letter in key]
key = [key[i:i+lengthMatrix] for i in range(0, len(key), lengthMatrix)]
if len(password) % lengthMatrix != 0:
    password = password + 'A' * (lengthMatrix - (len(password) % lengthMatrix))
password = [alphabet[letter] for letter in password]
password = [password[i:i+lengthMatrix] for i in range(0, len(password), lengthMatri
x)]

selector = app.getTextInput('Do you wish to encrypt or decrypt? (1 or 2)')
if selector == '1':
    encryption(password, key)
if selector == '2':
    decryption(password, key)
```