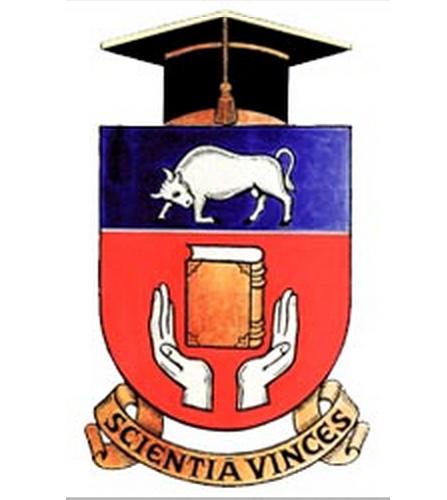
Universitatea de Stat din Tiraspol

Facultatea de Fizica Matematica si Informatica



Portofoliu

**Elaborat de**: Dordea Pavel, student gr.3i

**Verificat**: Globa Angela

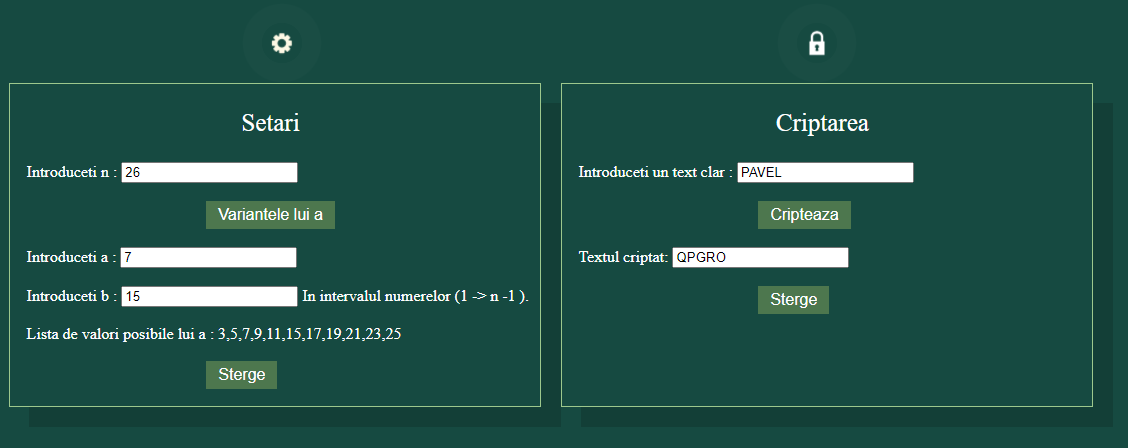
**Chișinău, 2021**

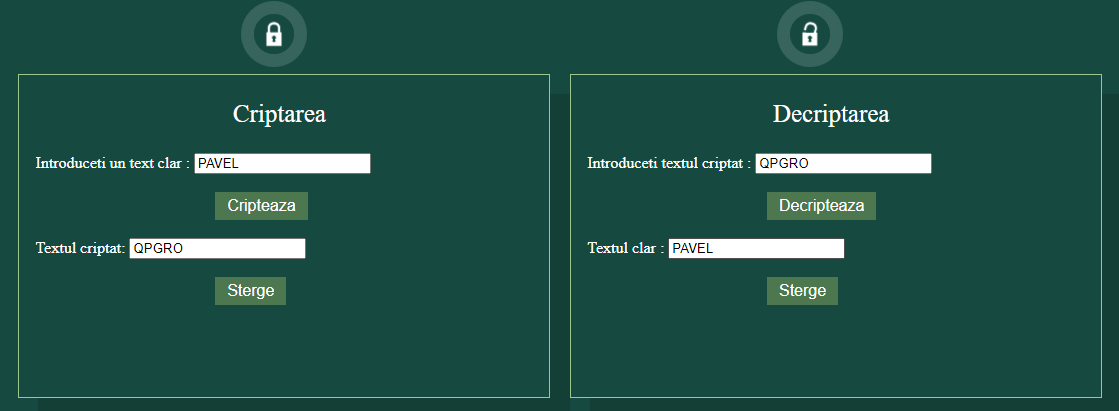
***#Despre***

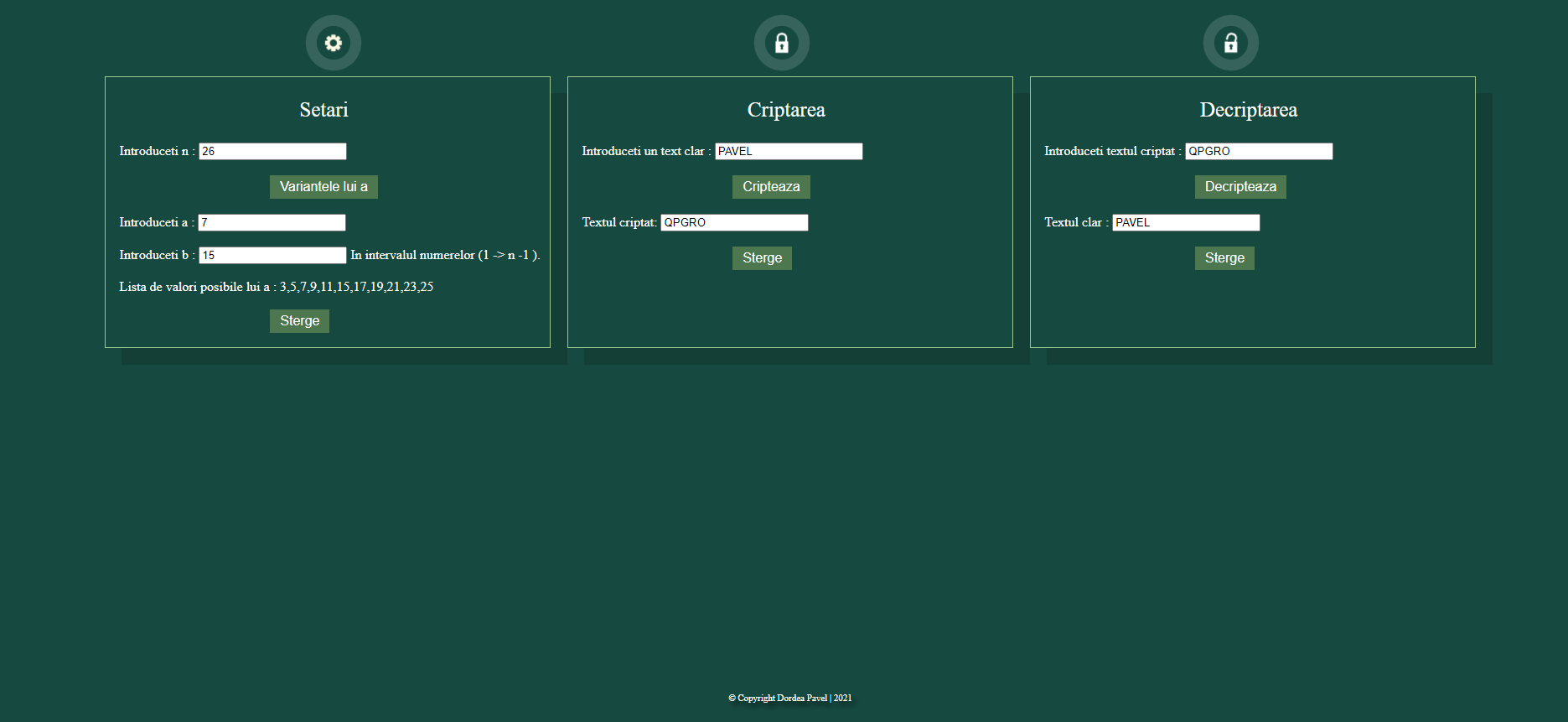
Cifrul afin este o generalizare a cifrului Cezar, având cheia k = {(a, b) | a, b ∈ Z26 = {0, 1, 2, …, 25}, cmmdc(a, 26) = 1}, iar funcţiile de criptare şi decriptare pentru o cheie k = (a, b) sunt ek(x) = ax + b (mod 26) și dk(y) = a −1(y+26 − b) (mod 26), unde a·a −1 =1 modn. Condiţia ca a să fie prim cu 26 asigură existenţa lui a −1 în Z26. Pentru calcularea lui a −1 poate fi aplicat algoritmul Euclid extins (vezi compartimentul 7), sau se verifică pa rând toate numerele reciproc prime cu a în clasa modulo n până când găsim acel număr pentru care a·a −1 =1 modn, acesta fiind și unicul cu o astfel de proprietate. De exemplu, pentru a = 7, b = 16 funcţia de criptare este ek(x) = 7x + 16. În acest caz pentru m =„cifrul afin”, obținem c =„ euzfap qzud”. Deoarece 7·15 =1 (mod26), avem 7−1 = 15 (mod 26), iar decriptarea se realizează matematic folosind funcţia 8 dk(y)= 15· (y+26−15)(mod 26)=15y+15·11 (mod 26)=15y + 9 (mod 26). Condiţia cmmdc(a, 26)=1 asigură de asemenea injectivitatea funcţiei ek, necesară pentru un proces corect de criptare-decriptare. De exemplu, pentru ek(x) = 10x + 1, A şi N se transformă ambele în B, iar O nu apare ca imagine în alfabetul substituţiei. În acest caz avem: cmmdc(a, 26)= cmmdc(10, 26)=2≠1. Astfel de cazuri au loc întotdeauna atunci când a nu respectă condiția de mai sus. Orice cheie k a cifrului afin este determinată complet de valorile întregi (a, b) pentru care cmmdc(a, 26) = 1. Sunt posibile 12 valori pentru a: 1, 3, 5, 7, 9, 11, 15, 19, 21, 23, 25 şi 26 valori pentru b, care se iau independent de a, cu o singură excepţie a = 1, b = 0 (care se exclude deoarece nu conduce la nici o criptare). Aşadar mulţimea cheilor în acest caz este alcătuită din 12·26–1=311 chei diferite, număr suficient de mic pentru atacul prin forţa brută, adică prin verificarea tuturor cheilor posibile. Trebuie de avut în vedere că în cazul în care este necesar de a cripta mesaje în limba română sau rusă (sau oricare altă limbă care are un alfabet) procedeul rămâne același, doar că trebuie să ținem cont de faptul că alfabetele respective au 31 și 33 de litere, adică n=31 și respectiv n=33.

***#Screenshots***









**Sistemul Caesar**

***#Despre***

În criptografie, **cifrul lui Cezar**, numit și **cifru cu deplasare**, **codul lui Cezar** sau **deplasarea lui Cezar**, este una dintre cele mai simple și mai cunoscute tehnici de criptare. Este un tip de cifru al substituției, în care fiecare literă din textul inițial este înlocuită cu o literă care se află în alfabet la o distanță fixă față de cea înlocuită. De exemplu, cu o deplasare de cinci poziții în alfabetul limbii române, A este înlocuit cu D, Ă devine E și așa mai departe. Această metodă este numită așa după Iulius Cezar, care o folosea pentru a comunica cu generalii săi.

Criptarea după cifrul Cezar poate fi reprezentată folosind aritmetică modulară prin transformarea literelor în numere conform schemei A = 0, Ă = 1,..., Z = 30. Astfel, alfabetul devine o secvență de 31 de numere, iar criptarea unei litere cu poziția din alfabet {\displaystyle x} printr-o deplasare spre dreapta cu *n* poziții poate fi descrisă matematic ca

{\displaystyle E\_{n}(x)=(x+n)\mod {31}.}

Decriptarea este făcută în mod similar:

{\displaystyle D\_{n}(x)=(x-n)\mod {31}.}

(Există mai multe definiții pentru operația modulo. În operația de mai sus, rezultatul se află în intervalul 0...30. Dacă *x+n* sau *x-n* nu se află în intervalul 0...30, atunci prin operația modulo se scad sau se adună 31 de atâtea ori până când condiția este îndeplinită).

***#COD***

alphabet = "abcdefghijklmnopqrstuvwxyz " #declaram alfabetul

letter\_to\_index = dict(zip(alphabet, range(len(alphabet))))

#facem conversia din litere in numarul de ordine din alfabet

index\_to\_letter = dict(zip(range(len(alphabet)), alphabet))

#facem conversia din numarul de ordine din alfabet in litere

Criptarea

def encrypt(message, shift=3): # functia pentru criptare cu parametrii mesaj si keye

cipher = "" # variabila pentru a salva mesajul criptat

for letter in message:#cream un ciclu pentru fiecare litera din mesajul clar

number = (letter\_to\_index[letter] + shift) % len(letter\_to\_index) #adunam numarul de ordine al literei cu keya apoi facem mod 26 (lungimea alfabetului)

letter = index\_to\_letter[number] #facem conversia numarului in litera corespunzatoare din alfabet

cipher += letter #o scrim in variabila noastra

return cipher #returnam mesajul criptat

Decriptarea

def decrypt(cipher, shift=3):# functia pentru decriptare cu parametrii mesaj si keye

decrypted = "" # variabila pentru a salva mesajul decriptat

for letter in cipher: #cream un ciclu pentru fiecare litera din mesajul criptat

number = (letter\_to\_index[letter] - shift) % len(letter\_to\_index) #scadem din numarul de ordine al literei keya apoi facem mod 26 (lungimea alfabetului)

letter = index\_to\_letter[number] #facem conversia numarului in litera corespunzatoare din alfabet

decrypted += letter #o scrim in variabila noastra

return decrypted #returnam mesajul clar

Corpul

def main():

message = input("Introduceti un mesaj:\n") # citim de la tastatura mesajul clar

key = int(input("Introduceti key-a:\n")) # citim de la tastatura key-a

cipher = encrypt(message, shift=key) # salvam in variabila cipher raspunsul returnat de functia de criptare (mesajul criptat)

decrypted = decrypt(cipher, shift=key) # salvam in variabila decrypted raspunsul returnat de functia de decriptare (mesajul decriptat)

print('Mesaj Original: ' + message) # Afisam mesajul clar

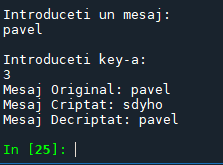
print('Mesaj Criptat: ' + cipher) #afisam mesajul criptat

print('Mesaj Decriptat: ' + decrypted) #afisam mesajul decriptat

main()

***#Screenshots***





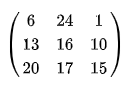
**Sistemul Hill**

***#Despre***

În criptografia clasică, **cifrul Hill** este un cifru al substituției poligrafic bazat pe algebră lineară. Inventat de către Lester S. Hill în 1929, a fost primul cifru poligrafic în care era practic posibil să se opereze cu mai mult de trei simboluri deodată. Pentru a înțelege discuția următoare sunt necesare cunoștințe de teoria matricelor.

***#Exemplu***

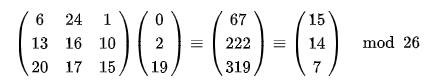
Fiecare literă este tratată ca o cifră din baza 26: A = 0, B =1 ș.a.m.d. Un bloc de *n* litere este considerat ca un vector cu n dimensiuni, și multiplicat cu o matrice n × n, modulo 26. Componentele matricei reprezintă cheia, care trebuie alese aleatoriu astfel încât matricea să fie inversabilă în {\displaystyle \mathbb {Z} \_{26}^{n}} (pentru a asigura posibilitatea decriptării). Considerăm mesajul 'ACT', și cheia de mai jos (sau GYBNQKURP în litere):



Deoarece 'A' este 0, 'C' este 2 și 'T' este 19, mesajul este vectorul:



Deci vectorul criptat este dat de:



care corespunde criptotextului 'POH'.

***#COD***

# -\*- coding: utf-8 -\*-

"""

Created on Tue Dec 28 15:59:08 2021

@author: Pavel

"""

keyMatrix = [[0] \* 3 for i in range(3)]

messageVector = [[0] for i in range(3)]

cipherMatrix = [[0] for i in range(3)]

def getKeyMatrix(key):

k = 0

for i in range(3):

for j in range(3):

keyMatrix[i][j] = ord(key[k]) % 65

k += 1

def encrypt(messageVector):

for i in range(3):

for j in range(1):

cipherMatrix[i][j] = 0

for x in range(3):

cipherMatrix[i][j] += (keyMatrix[i][x] \*

messageVector[x][j])

cipherMatrix[i][j] = cipherMatrix[i][j] % 26

def HillCipher(message, key):

getKeyMatrix(key)

for i in range(3):

messageVector[i][0] = ord(message[i]) % 65

encrypt(messageVector)

CipherText = []

for i in range(3):

CipherText.append(chr(cipherMatrix[i][0] + 65))

print("Ciphertext: ", "".join(CipherText))

def main():

message = "ACT"

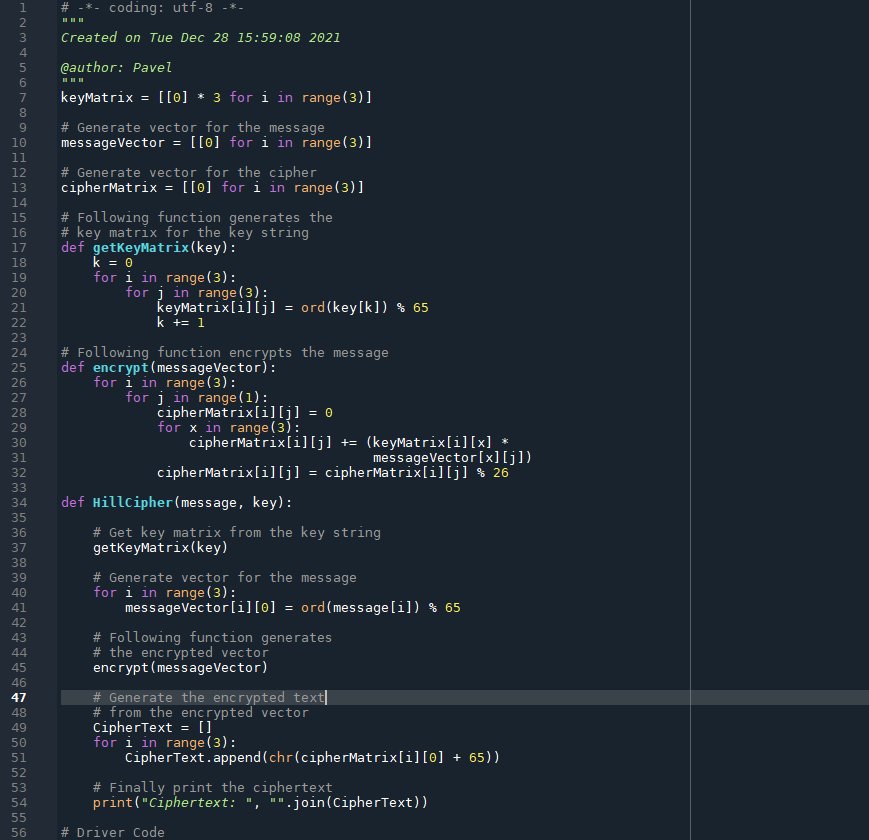
key = "GYBNQKURP"

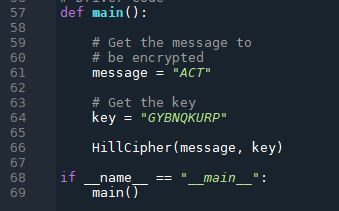
HillCipher(message, key)

if \_\_name\_\_ == "\_\_main\_\_":

main()

***#Screenshots***







***#Program Universal***

# -\*- coding: utf-8 -\*-

"""

Created on Tue Dec 28 16:34:26 2021

@author: Pavel

"""

import numpy as np

from egcd import egcd # pip install egcd

alphabet = "abcdefghijklmnopqrstuvwxyz "

letter\_to\_index = dict(zip(alphabet, range(len(alphabet))))

index\_to\_letter = dict(zip(range(len(alphabet)), alphabet))

def matrix\_mod\_inv(matrix, modulus):

"""Găsim modulul matricei invers cu

Pasul 1) Găsiți determinantul

Pasul 2) Găsiți valoarea determinantului într-un anumit modul (de obicei lungimea alfabetului)

Pasul 3) Luați acel det\_inv ori det\*matricea inversată (aceasta va fi apoi adjunctul) în mod 26

"""

det = int(np.round(np.linalg.det(matrix))) # Pas 1)

det\_inv = egcd(det, modulus)[1] % modulus # Pas 2)

matrix\_modulus\_inv = (

det\_inv \* np.round(det \* np.linalg.inv(matrix)).astype(int) % modulus

) # Pas 3)

return matrix\_modulus\_inv

def encrypt(message, K):

encrypted = ""

message\_in\_numbers = []

for letter in message:

message\_in\_numbers.append(letter\_to\_index[letter])

split\_P = [

message\_in\_numbers[i : i + int(K.shape[0])]

for i in range(0, len(message\_in\_numbers), int(K.shape[0]))

]

for P in split\_P:

P = np.transpose(np.asarray(P))[:, np.newaxis]

while P.shape[0] != K.shape[0]:

P = np.append(P, letter\_to\_index[' '])[:, np.newaxis]

numbers = np.dot(K, P) % len(alphabet)

n = numbers.shape[0] # lungimea mesajului criptat (în cifre)

# Harta înapoi pentru a obține text criptat

for idx in range(n):

number = int(numbers[idx, 0])

encrypted += index\_to\_letter[number]

return encrypted

def decrypt(cipher, Kinv):

decrypted = ""

cipher\_in\_numbers = []

for letter in cipher:

cipher\_in\_numbers.append(letter\_to\_index[letter])

split\_C = [

cipher\_in\_numbers[i : i + int(Kinv.shape[0])]

for i in range(0, len(cipher\_in\_numbers), int(Kinv.shape[0]))

]

for C in split\_C:

C = np.transpose(np.asarray(C))[:, np.newaxis]

numbers = np.dot(Kinv, C) % len(alphabet)

n = numbers.shape[0]

for idx in range(n):

number = int(numbers[idx, 0])

decrypted += index\_to\_letter[number]

return decrypted

def main():

message = input("Introduceti mesajul: ").lower()

K = np.matrix([[3,10,20],[20,19,17], [23,78,17]]) # for length of alphabet = 27

Kinv = matrix\_mod\_inv(K, len(alphabet))

encrypted\_message = encrypt(message, K)

decrypted\_message = decrypt(encrypted\_message, Kinv)

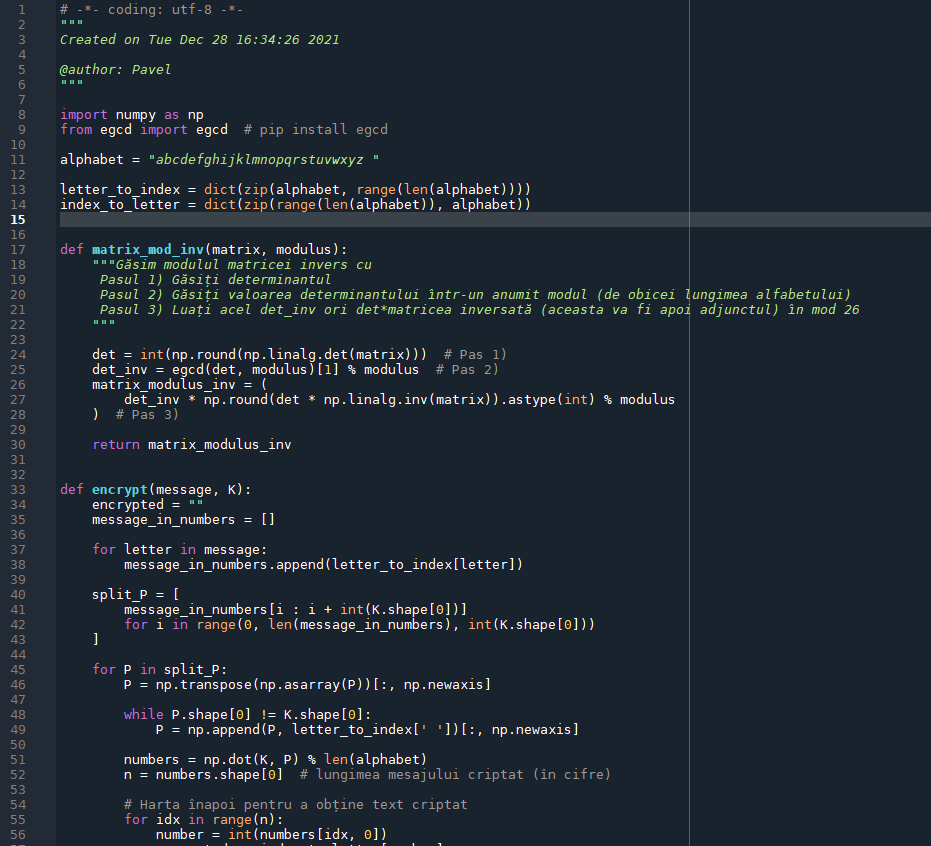
print("Mesaj Clar: " + message)

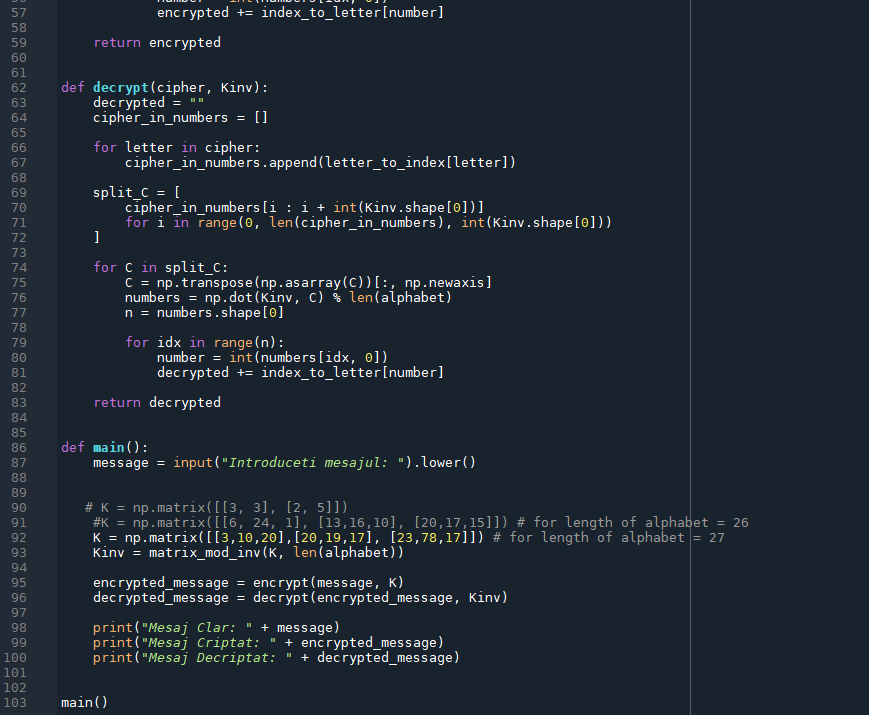
print("Mesaj Criptat: " + encrypted\_message)

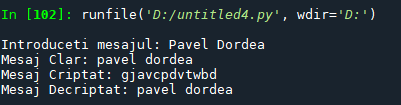
print("Mesaj Decriptat: " + decrypted\_message)

main()

***#Screenshots***



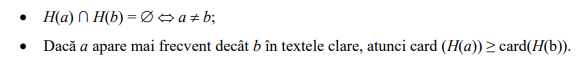




***Cifrul Omofonic***

***#Despre***

Cifrul omofonic (homophonic ciphers) este un cifru de substituţie în care un caracter al alfabetului mesajului clar (alfabet primar) poate să aibă mai multe reprezentări. Ideea utilizată în aceste cifruri este uniformizarea frecvenţelor de apariţie a caracterelor alfabetului textului cifrat (alfabet secundar), pentru a îngreuna atacurile criptanalitice. Astfel, litera A - cu cea mai mare frecvenţă de apariţie în alfabetul primar – poate fi înlocuită de exemplu cu H, # sau m. Cifrul omofonic este un cifru intermediar între sistemele mono şi cele polialfabetice. Principalul lui scop este de a evita atacul prin frecvenţa de apariţie a caracterelor. Se presupune că a fost utilizat prima oară în 1401 de către ducele de Mantua. În cifrul omofonic fiecărui caracter a∈m i se asociază o mulţime H(a) ⊂ c astfel încât:

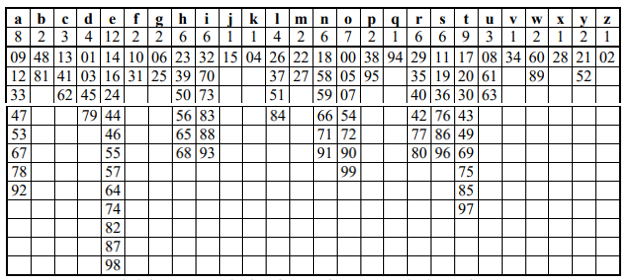


Criptarea unui caracter x ∈m se face cu un element ales aleator din H(a). Pentru decriptarea lui y ∈ c se caută o mulţime H(x) astfel încât y ∈ H(x).

Exemplu. Pentru limba engleză poate fi utilizat cifrul definit de tabelul 5.5, în primele două linii ale căruia sunt aşezate literele alfabetului latin şi frecvenţele rotunjite ale acestora. În coloanele de sub litera x este situat H(x). De exemplu



Pentru criptarea textului „ab” se poate folosi oricare din secvenţele 0948, 1248, 3348, 4748, 5348, 6748, 7848, 9248, 0981, 1281, 3381, 4781, 5381, 6781, 7881, 9281. Tabelul 5.5 a fost completat cu numerele 00, 01,…, 99, ele fiind în cantitate de 100 pentru a simplifica calculele legate de frecvența literelor (adică pentru a corespunde cu 100% a frecvențelor). Însă elementele acestui tabel pot fi oricare simboluri, iar cantitatea lor va determina coeficientul de proporționalitate pentru frecvențele fiecărei litere din alfabet.



Deşi mai greu de spart decât cifrurile de substituţie simple (monoalfabetice), cifrul omofonic nu maschează total proprietăţile statistice ale textului clar. În cazul unui atac cu text clar cunoscut, cifrul se sparge extrem de uşor. Atacul cu text cifrat este mai dificil, dar unui calculator îi va lua doar câteva secunde pentru a-l sparge.

***#COD***

Dictionarul

aN = [

27, 44, 43, 48, 38, 22, 59, #a

34, 92, #b

20, 37, 94, #c

31, 99, 50, 60, #d

91, 68, 21, 45, 90, 58, 81, 64, 65, 36, 96, #e

12, 74, #f

41, 76, #g

40, 55, 78, 83, #h

62, 61, 79, 80, 15, #i

24, #j

88, #k

95, 49, 28, 70, #l

89, 25, #m

53, 33, 66, 98, 13, #n

46, 86, 11, 42, 63, 69, #o

54, 82, #p

85, #q

77, 87, 97, 29, 26, #r

35, 17, 84, 47, 52, #s

71, 14, 51, 30, 32, 73, 16, 19, #t

93, 72, 39, #u

10, #v

67, 75, #w

23, #x

57, 56, #y

18 #z

]

Criptare

def encrypt():

print('Introduceti mesajul dvs.')

msg = input(">")

msg = msg.upper()

encrMsg = ""

maxLen = 0

i = 0

while(len(msg) > i):

dMsg = ord(msg[i]) - 65 #este cheia UNICODE pentru numerele cu text simplu

maxLen = len(x[dMsg]) #Lungimea maximă a generatorului aleator

encrMsg += str(x[dMsg][random.randint(0, maxLen - 1)])

i += 1

print("\nMesajul dvs.: " + encrMsg + "\n\n")

askScreen()

Decriptare

def decrypt():

print('Introduceti mesajul dvs. pentru decriptare')

msg = input(">")

msg = wrap(msg, 2)

msgcount = 0

w = 0

i = 0

decrMsg = "Mesajul dvs.:"

print("")

while(len(msg) -1 >= msgcount):

while(24 > i):

while(len(x[i]) > w):

if(str(msg[msgcount]) == str(x[i][w])):

decrMsg = str(decrMsg) + (chr((i)+65))

w += 1

i += 1

w = 0

w = 0

i = 0

msgcount += 1

print(str(decrMsg) + "\n\n")

askScreen()

Corpul

def askScreen():

print("|Pavel Dordea|\n |Homophonic System|")

print("\ne pentru criptare, d pentru decriptare")

print(" fara spatii!")

choose = input(">")

if(choose == "e"):

encrypt()

elif(choose == "d"):

decrypt()

else:

print("Incercati din nou \n\n")

askScreen()

askScreen()

Tot programul

# -\*- coding: utf-8 -\*-

"""

Created on Tue Dec 28 15:11:14 2021

@author: Pavel

"""

import os

import random

from textwrap import wrap

aN = [

27, 44, 43, 48, 38, 22, 59, #a

34, 92, #b

20, 37, 94, #c

31, 99, 50, 60, #d

91, 68, 21, 45, 90, 58, 81, 64, 65, 36, 96, #e

12, 74, #f

41, 76, #g

40, 55, 78, 83, #h

62, 61, 79, 80, 15, #i

24, #j

88, #k

95, 49, 28, 70, #l

89, 25, #m

53, 33, 66, 98, 13, #n

46, 86, 11, 42, 63, 69, #o

54, 82, #p

85, #q

77, 87, 97, 29, 26, #r

35, 17, 84, 47, 52, #s

71, 14, 51, 30, 32, 73, 16, 19, #t

93, 72, 39, #u

10, #v

67, 75, #w

23, #x

57, 56, #y

18 #z

]

x = [

[aN[0], aN[1], aN[2], aN[3], aN[4], aN[5], aN[6]], #a 0

[aN[7], aN[8]], #b 1

[aN[9], aN[10], aN[11]], #c 2

[aN[12], aN[13], aN[14], aN[15]], #d 3

[aN[16], aN[17], aN[18], aN[19], aN[20], aN[21], aN[22], aN[23], aN[24], aN[25], aN[26]], #e 4

[aN[27], aN[28]], #f 5

[aN[29], aN[30]], #g 6

[aN[31], aN[32], aN[33], aN[34]], #h 7

[aN[35], aN[36], aN[37], aN[38], aN[39]], #i 8

[aN[40]], #j 9

[aN[41]], #k 10

[aN[42], aN[43], aN[44], aN[45]], #l 11

[aN[46], aN[47]], #m 12

[aN[48], aN[49], aN[50], aN[51], aN[52]], #n 13

[aN[53], aN[54], aN[55], aN[56], aN[57], aN[58]], #o 14

[aN[59], aN[60]], #p 15

[aN[61]], #q 16

[aN[62], aN[63], aN[64], aN[65], aN[66]], #r 17

[aN[67], aN[68], aN[69], aN[70], aN[71]], #s 18

[aN[72], aN[73], aN[74], aN[75], aN[76], aN[77], aN[78], aN[79]], #t 19

[aN[80], aN[81], aN[82]], #u 20

[aN[83]], #v 21

[aN[84], aN[85]], #w 22

[aN[86]], #x 23

[aN[87], aN[88]], #y 24

[aN[89]] #z 25

]

def encrypt():

print('Introduceti mesajul dvs.')

msg = input(">")

msg = msg.upper()

encrMsg = ""

maxLen = 0

i = 0

while(len(msg) > i):

dMsg = ord(msg[i]) - 65 #este cheia UNICODE pentru numerele cu text simplu

maxLen = len(x[dMsg]) #Lungimea maximă a generatorului aleator

encrMsg += str(x[dMsg][random.randint(0, maxLen - 1)])

i += 1

print("\nMesajul dvs.: " + encrMsg + "\n\n")

askScreen()

def decrypt():

print('Introduceti mesajul dvs. pentru decriptare')

msg = input(">")

msg = wrap(msg, 2)

msgcount = 0

w = 0

i = 0

decrMsg = "Mesajul dvs.:"

print("")

while(len(msg) -1 >= msgcount):

while(24 > i):

while(len(x[i]) > w):

if(str(msg[msgcount]) == str(x[i][w])):

decrMsg = str(decrMsg) + (chr((i)+65))

w += 1

i += 1

w = 0

w = 0

i = 0

msgcount += 1

print(str(decrMsg) + "\n\n")

askScreen()

def askScreen():

print("|Pavel Dordea|\n |Homophonic System|")

print("\ne pentru criptare, d pentru decriptare")

print(" fara spatii!")

choose = input(">")

if(choose == "e"):

encrypt()

elif(choose == "d"):

decrypt()

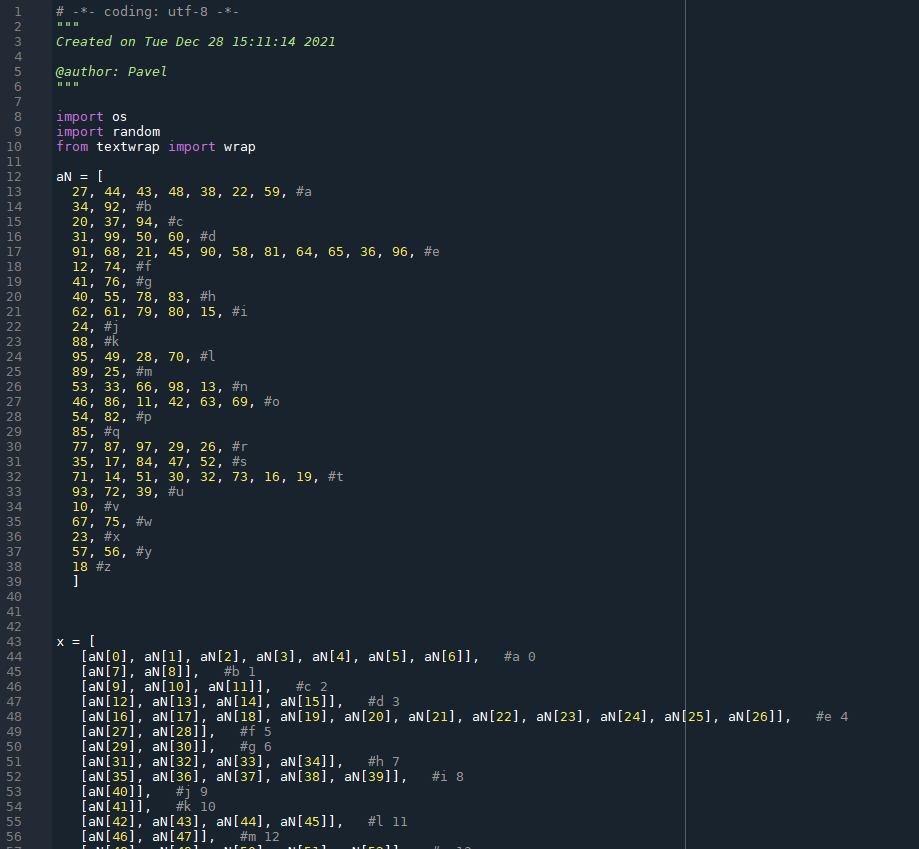
else:

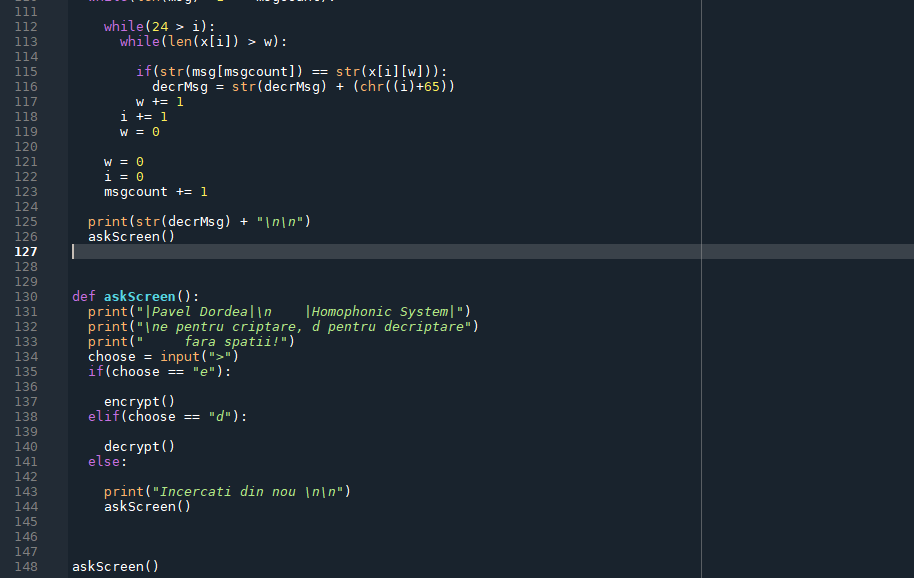
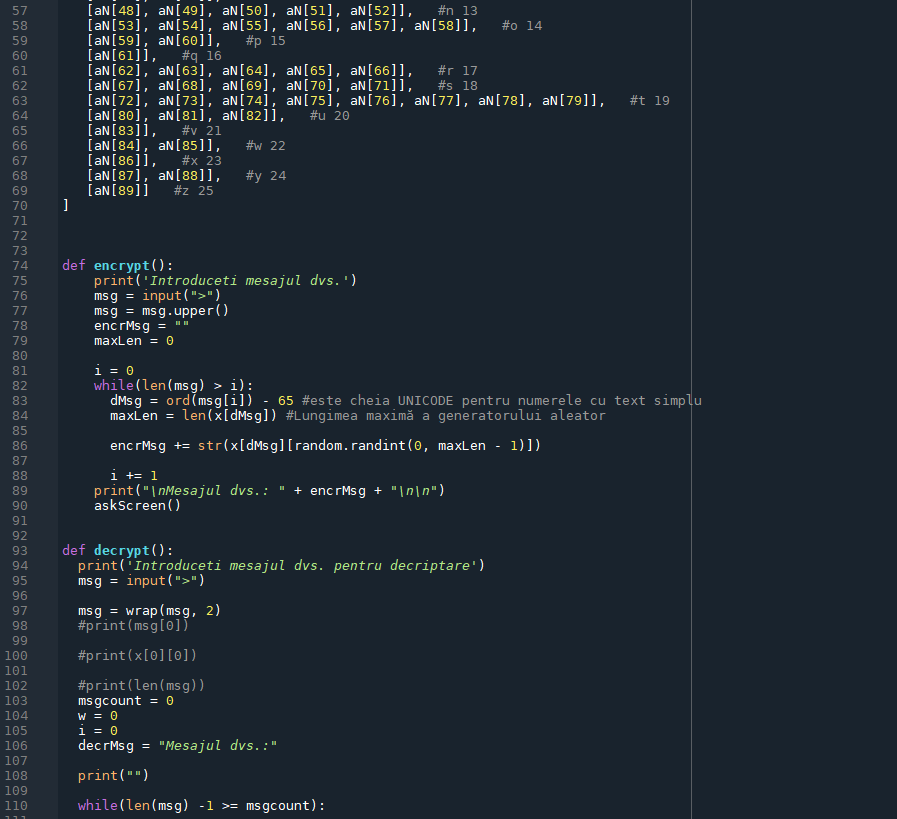
print("Incercati din nou \n\n")

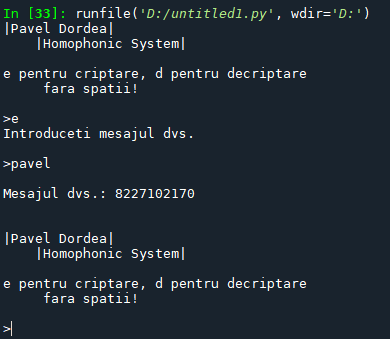
askScreen()

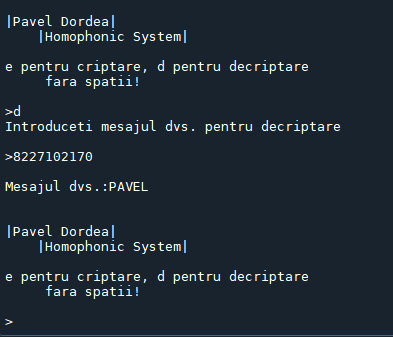
askScreen()

***#Screenshots***









**Împărțirea secretelor. Schema lui Shamir**

Fie că trebuie expediat un pin cod: YURII

Y –24

U –20

R –17

I –08

I –08

YURII - 2420170808

Schema lui Shamir se bazează pe interpolarea polinomială

1. Fie *S* – secretul; *S*=2420170808;
2. Alegem aleator *k* coieficienți: a1, a2, ..., ak;

*k=*6; a1=12; a2=3; a3=15; a4=7; a5=20; a6=27;

1. Construim polinomul: *f*(*x*)= ak·*xk* +ak-1·*xk*-1 + ... + a1·*x*+a0

a0=S;

*f*(*x*)= 26∙x6+20·*x*5+7·*x*4 + 15·*x*3+ 3·*x 2*+12·*x+*2420170808

1. Se alege un număr *n*=18; n - numărul de umbre (curieri)

Definesc umbrele, adică formez perechile (*i*, *f*(*i*))

|  |  |
| --- | --- |
| X | F(X) |
| 1 | 2420170891 |
| 2 | 2420173380 |
| 3 | 2420195657 |
| 4 | 2420300632 |
| 5 | 2420645943 |
| 6 | 2421551876 |
| 7 | 2423588005 |
| 8 | 2427678552 |
| 9 | 2435226467 |
| 10 | 2448256228 |
| 11 | 2469575361 |
| 12 | 2502954680 |
| 13 | 2553327247 |
| 14 | 2627006052 |
| 15 | 2731920413 |
| 16 | 2877871096 |
| 17 | 3076804155 |
| 18 | 3343103492 |

Ubrele se repartizează ca perechi:

**(1,** 2420170891**), (2,** 2420173380**),…, (18,** 3343103492**)**

Pentru restabilirea secretului avem nevoie de *k*+1 umbre

Deoarece *k*=6, atunci avem nevoie de 7 umbre pentru restabilirea secretului.

Pentru restabilirea polinomului vom aplica polinomul de interpolare Lagrange:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | X0 | X1 | X2 | X3 | X4 | X5 | X6 |
| i,x | 2 | 5 | 8 | 10 | 12 | 15 | 18 |
| f(i),y | 2420173380 | 2420645943 | 2427678552 | 2448256228 | 2502954680 | 2731920413 | 3343103492 |
|  | y0 | y1 | y2 | y3 | Y4 | Y5 | Y6 |

=...+...+ 2420170808

Secretul restabilit este: 2420170808

Pentru a obține secretul inițial se împarte numărul în secvențe de 2 cifre de la dreapta spre stânga: 24 20 17 08 08. Dacă prima secvență ete formată dintr-o cifră, atunci se adaugă 0 în fața sa: 24 20 17 08 08– YURII.

Sarcină.

1. Secretul lui Shamir, pentru k=5
2. Program
3. De verificat rezultatele de la pl1 și de afișat screenshot.

2.cod:

# -\*- coding: utf-8 -\*-

"""

Created on Wed Nov 10 13:21:17 2021

@author: Pavel

"""

import tkinter as tk

import random

root= tk.Tk()

root.resizable(0,0)

root.title('Impartirea secretelor')

root.configure(bg='#164A41')

L1=tk.Label(root,text="Introdu cheia secreta s :",bg='#164A41',fg='#FFFFFF',font='Helvetica 14 bold').grid(row=0,column=0,sticky='w')

sec = tk.Entry(root,bg='#F1B24A',fg='#FFFFFF',font='Helvetica 12 bold')

sec.grid(row=0,columnspan=3,sticky='ne')

L2=tk.Label(root,text="Introdu n (in cite parti va fi divizat secretul):",bg='#164A41',fg='#FFFFFF',font='Helvetica 14 bold').grid(row=1,column=0,sticky='w')

nec = tk.Entry(root,bg='#F1B24A',fg='#FFFFFF',font='Helvetica 12 bold')

nec.grid(row=1,columnspan=3,sticky='ne')

L3=tk.Label(root,text="Introdu k (cite bucati sunt necesare pentru restabilirea cheii):",bg='#164A41',fg='#FFFFFF',font='Helvetica 14 bold').grid(row=2,column=0,sticky='w')

kec = tk.Entry(root,bg='#F1B24A',fg='#FFFFFF',font='Helvetica 12 bold')

kec.grid(row=2,column=1)

b=[]

L7=tk.Label(root,text="",bg='#164A41',fg='#FFFFFF',font='Helvetica 10 bold')

L7.grid(row=6,column=1,sticky='w')

def afla():

s=sec.get()

s = int(sec.get())

k = int(kec.get())

a = random.sample(range(1, 100), k)

n = int(nec.get())

def F(t):

w = s

for i in range(len(a)):

w += a[len(a)-i-1]\*pow(t,len(a)-i)

return w

#a=[12,3,15,7,20,26]

L5["text"]=str(a)

for i in range (1,n+1):

b.append(F(i))

for i in range (0,n):

L7["text"]+=("["+str(i+1)+"] "+str(b[i])+" \n")

L10['text']+="("+str(k)+' pozitii din lista de mai sus):'

x=[]

def calc():

y=[]

final=0

k = int(kec.get())

for i in range(k+1):

value = (nume.get())

value = value.split(' ')

x = [int(x) for x in value]

y.append(b[x[i]-1])

for g in range(k+1):

p2=p1=1

for r in range(k+1):

if (g != r):

p2 \*= (x[g]-x[r])

p1 \*= x[r]

final += ((p1\*y[g])/p2)

L12["text"]="CHEIA RESTABILITA ESTE:"

L13["text"]=round(final)

button\_afla = tk.Button(root,text="Calculeaza",command=afla,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold').grid(row=4,columnspan=3,sticky='we')

L4=tk.Label(root,text="Numerele aleatoare generate:",bg='#164A41',fg='#FFFFFF',font='Helvetica 12 bold').grid(row=5,column=0,sticky='w')

L5=tk.Label(root,text="",bg='#164A41',fg='#FFFFFF',font='Helvetica 10 bold')

L5.grid(row=5,column=1,sticky='w')

L6=tk.Label(root,text="Perechile de numere in care a fost impartita cheia:",bg='#164A41',fg='#FFFFFF',font='Helvetica 12 bold').grid(row=6,column=0,sticky='w')

L8=tk.Label(root,text="RESTABILIREA CHEII:",bg='#164A41',fg='#FFFFFF',font='Helvetica 12 bold underline').grid(row=7,column=0,columnspan=3,sticky='n')

L9=tk.Label(root,text="Introduceti perechile de numere care participa la restabilirea cheii:",bg='#164A41',fg='#FFFFFF',font='Helvetica 12 bold').grid(row=8,column=0,sticky='w')

L10=tk.Label(root,text="Numarul de ordine a perechii ",bg='#164A41',fg='#FFFFFF',font='Helvetica 12 bold')

L10.grid(row=9,column=0,sticky='w')

nume=tk.Entry(root,bg='#F1B24A',fg='#FFFFFF',font='Helvetica 12 bold')

nume.grid(row=9,column=1,sticky='w')

L12=tk.Label(root,text="",bg='#164A41',fg='red',font='Helvetica 12 bold')

L12.grid(row=11,column=0,columnspan=3,sticky='w')

L13=tk.Label(root,text="",bg='#164A41',fg='red',font='Helvetica 12 bold')

L13.grid(row=11,column=1,sticky='ne')

#button\_ceck = tk.Button(root,text="Adauga",command=check).grid(row=11,column=0)

button\_calc = tk.Button(root,text="Decrypteaza",command=calc,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold').grid(row=10,columnspan=3,sticky='we')

root.mainloop()

3.



**Sistemul de criptare Playfair**

Cheia: CONTRACT DE DONATIE BILATERALA A SOCIETATII PE ACTIUNI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **C** | O | N | T | R |
| A | D | **E** | I | B |
| L | S | P | F | G |
| H | J | K | M | U |
| V | W | X | Y | Z |

Text clar: **SPECIALITATEA**

Criptarea:

1. Se imparte textul clar in secvente de 2 caractere:

**SP EC IA LI TA TE AW**

1. Daca literele se afla pe aceeasi linie/coloanal ne deplasam, circular, cu o pozitie in directia secventei de litere;

**SP** – **PF**

**IA** - **EB**

1. Analizam dreptunghiurile cu colturile in literele specificate:

**EC** - **AN**

**LI** - **FA**

**TA** – **CI**

**TE** - **NI**

**AW** - **DV**

Text criptat: **PFANEBFACINIDV**

Decriptarea:

**PF AN EB FA CI NI DV**

**PF – SP**

**AN – EC**

**EB -**

***#COD***

# -\*- coding: utf-8 -\*-

"""

Created on Tue Dec 28 17:18:33 2021

@author: Pavel

"""

def generateKeyMatrix (key):

# Create 5X5 matrix with all values as 0

# [

# [0, 0, 0, 0, 0],

# [0, 0, 0, 0, 0],

# [0, 0, 0, 0, 0],

# [0, 0, 0, 0, 0],

# [0, 0, 0, 0, 0]

# ]

matrix\_5x5 = [[0 for i in range (5)] for j in range(5)]

simpleKeyArr = []

"""

Generate SimpleKeyArray with key from user Input

with following below condition:

1. Character Should not be repeated again

2. Replacing J as I (as per rule of playfair cypher)

"""

for c in key:

if c not in simpleKeyArr:

if c == 'J':

simpleKeyArr.append('I')

else:

simpleKeyArr.append(c)

"""

Fill the remaining SimpleKeyArray with rest of unused letters from english alphabets

"""

is\_I\_exist = "I" in simpleKeyArr

# A-Z's ASCII Value lies between 65 to 90 but as range's second parameter excludes that value we will use 65 to 91

for i in range(65,91):

if chr(i) not in simpleKeyArr:

# I = 73

# J = 74

# We want I in simpleKeyArr not J

if i==73 and not is\_I\_exist:

simpleKeyArr.append("I")

is\_I\_exist = True

elif i==73 or i==74 and is\_I\_exist:

pass

else:

simpleKeyArr.append(chr(i))

"""

Now map simpleKeyArr to matrix\_5x5

"""

index = 0

for i in range(0,5):

for j in range(0,5):

matrix\_5x5[i][j] = simpleKeyArr[index]

index+=1

return matrix\_5x5

def indexLocator (char,cipherKeyMatrix):

indexOfChar = []

# convert the character value from J to I

if char=="J":

char = "I"

for i,j in enumerate(cipherKeyMatrix):

# [

# (0, ['K', 'A', 'R', 'E', 'N']),

# (1, ['D', 'B', 'C', 'F', 'G']),

# (2, ['H', 'I', 'L', 'M', 'O']),

# (3, ['P', 'Q', 'S', 'T', 'U']),

# (4, ['V', 'W', 'X', 'Y', 'Z'])

# ]

# j refers to inside matrix => ['K', 'A', 'R', 'E', 'N'],

for k,l in enumerate(j):

# [(0,'K'),(1,'A'),(2,'R'),(3,'E'),(4,'N')]

if char == l:

indexOfChar.append(i)

indexOfChar.append(k)

return indexOfChar

def encryption (plainText,key):

cipherText = []

# 1. Generate Key Matrix

keyMatrix = generateKeyMatrix(key)

# 2. Encrypt According to Rules of playfair cipher

i = 0

while i < len(plainText):

# 2.1 calculate two grouped characters indexes from keyMatrix

n1 = indexLocator(plainText[i],keyMatrix)

n2 = indexLocator(plainText[i+1],keyMatrix)

# 2.2 if same column then look in below row so

# format is [row,col]

# now to see below row => increase the row in both item

# (n1[0]+1,n1[1]) => (3+1,1) => (4,1)

# (n2[0]+1,n2[1]) => (4+1,1) => (5,1)

# but in our matrix we have 0 to 4 indexes only

# so to make value bound under 0 to 4 we will do %5

# i.e.,

# (n1[0]+1 % 5,n1[1])

# (n2[0]+1 % 5,n2[1])

if n1[1] == n2[1]:

i1 = (n1[0] + 1) % 5

j1 = n1[1]

i2 = (n2[0] + 1) % 5

j2 = n2[1]

cipherText.append(keyMatrix[i1][j1])

cipherText.append(keyMatrix[i2][j2])

cipherText.append(", ")

# same row

elif n1[0]==n2[0]:

i1= n1[0]

j1= (n1[1] + 1) % 5

i2= n2[0]

j2= (n2[1] + 1) % 5

cipherText.append(keyMatrix[i1][j1])

cipherText.append(keyMatrix[i2][j2])

cipherText.append(", ")

# if making rectangle then

# [4,3] [1,2] => [4,2] [3,1]

# exchange columns of both value

else:

i1 = n1[0]

j1 = n1[1]

i2 = n2[0]

j2 = n2[1]

cipherText.append(keyMatrix[i1][j2])

cipherText.append(keyMatrix[i2][j1])

cipherText.append(", ")

i += 2

return cipherText

def convertPlainTextToDiagraphs (plainText):

# add X if Two letters are being repeated

for s in range(0,len(plainText)+1,2):

if s<len(plainText)-1:

if plainText[s]==plainText[s+1]:

plainText=plainText[:s+1]+'X'+plainText[s+1:]

# append X if the total letters are odd, to make plaintext even

if len(plainText)%2 != 0:

plainText = plainText[:]+'X'

return plainText

def main():

key = input("Introduceti key-ul: ").replace(" ","").upper()

plainText =input("Textul clar: ").replace(" ","").upper()

convertedPlainText = convertPlainTextToDiagraphs(plainText)

print('Textul clar introdus : '+convertedPlainText)

keyMatrix = generateKeyMatrix(key)

cipherText = " ".join(encryption(convertedPlainText,key))

print('Textul criptat : '+cipherText)

# print(len(keyMatrix)) => 5

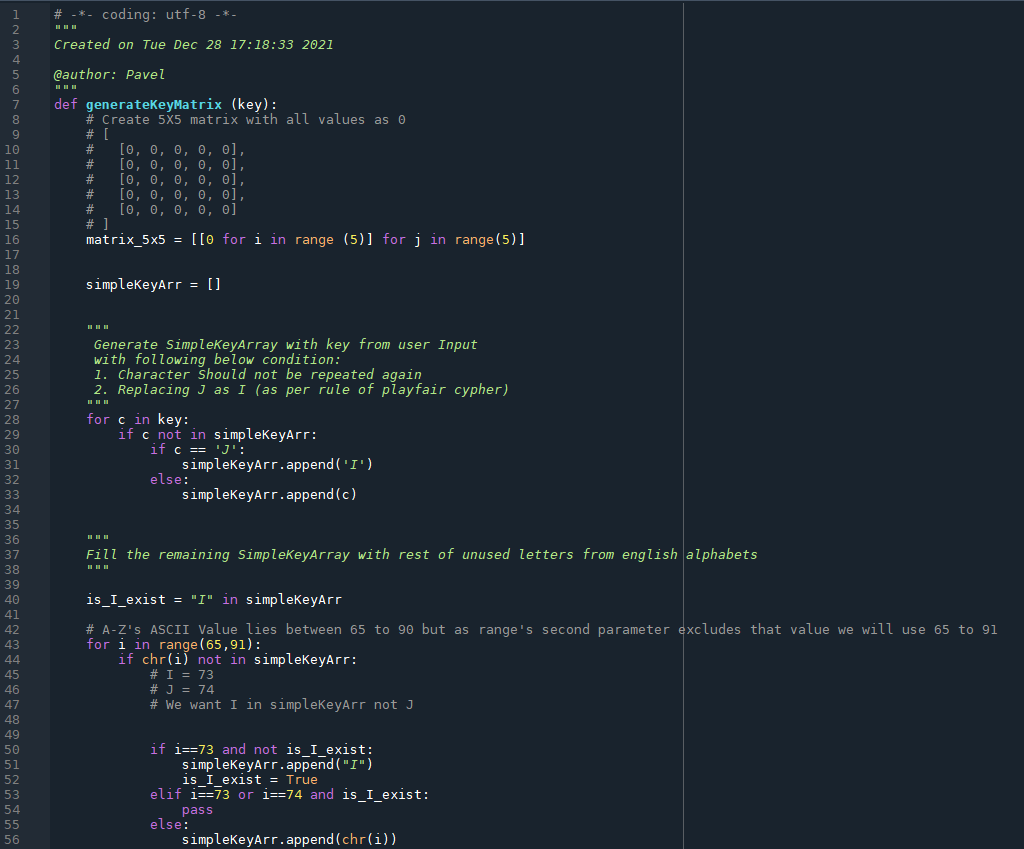
print('\nMatricea : \n')

print(keyMatrix)

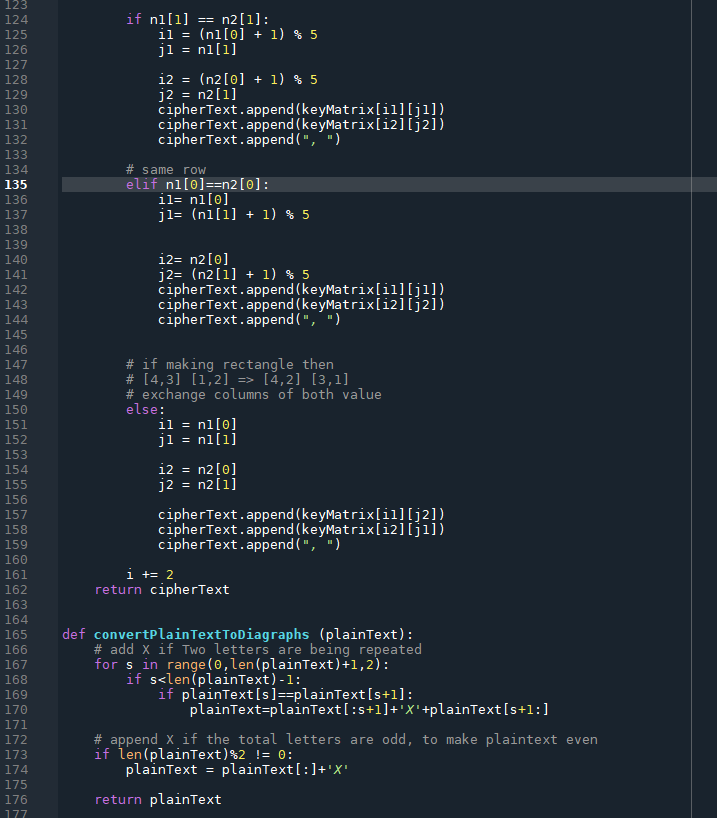
if \_\_name\_\_ == "\_\_main\_\_":

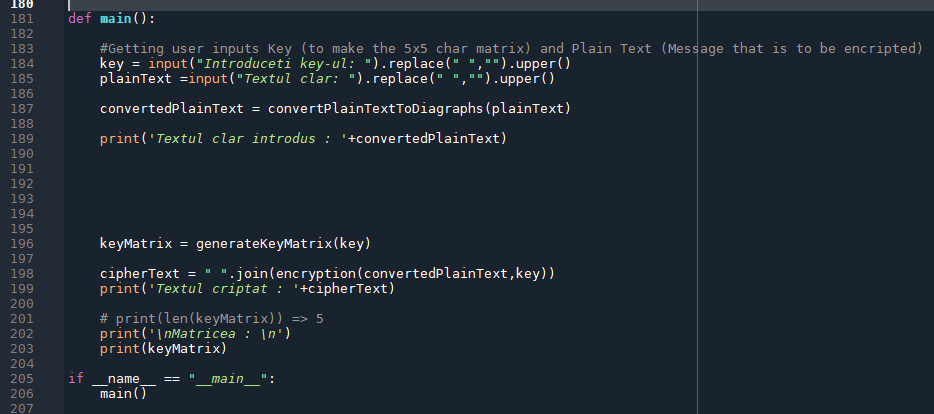
main()

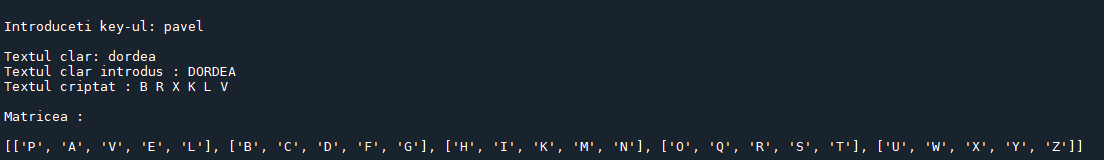
***#Screenshots***











**Sistemul de criptare Rishelieu**

Text clar: **securitatea informatiei este un obiect…**

**Criptarea:**

1. Scriem textul intr-un tabel patrat de dimensiunea n, unde n\*n se divide cu 4 (n=2,4,6,8,10,….);
2. Fie n=4;
3. Cream cheia conform regulii:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

begin

a[i,j]:=0;

a[j,n-i+1]:=1;

a[n-j+1,i]:=1;

a[n-i+1,n-j+1]:=1; k:=k+1;

end;

**daca n=4, atunci avem nevoie de n2/4 spatii.**

**Deci, vor fi 4 spații.**

Inițial, alegem primul spațiu în mod aleator.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Fie, a[2,3]=0 (spatiu),i=2,j=3,

a[j,n-i+1]=a[3,4-2+1]= a[3,3]=1;

a[n-j+1,i]= a[4-3+1,2]:=a[2,2]=1;

a[n-i+1,n-j+1]:= a[4-2+1,4-3+1]:=a[3,2]=1

Din nou, aleatory, alegem un spațiu din cele nemarcate.

Fie **a[1,3]=0**, i=1,j=3

a[j,n-i+1]=a[3,4-1+1]= a[3,4]=1;

a[n-j+1,i]= a[4-3+1,1]:=a[2,1]=1;

a[n-i+1,n-j+1]:= a[4-1+1,4-3+1]:=a[4,2]=1

Alegem aleatory umătorul spațiu.

Fie, **a[3,1]=0**, i=3,j=1

a[j,n-i+1]=a[1,4-3+1]= a[1,2]=1;

a[n-j+1,i]= a[4-1+1,3]:=a[4,3]=1;

a[n-i+1,n-j+1]:= a[4-3+1,4-1+1]:=a[2,4]=1

1. Pentru criptare, cheia se roteste de trei ori la 90 de grade;
2. Scriem literele care se vad in celulele transparente;

Text criptat:

euiaftiub t\_oa\_t\_\_e ctnresnoi sraeimiee

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s | e | c | u | r | i |
| t | a | t | e | a |  |
| i | n | f | o | r | m |
| a | t | i | e | i |  |
| e | s | t | e |  | u |
| n |  | o | b | i | e |

Pentru decrptare:

1. Se ia o retea goala si se suprapune pe ea cheia
2. Se scrie in cellele transparente literele din textul ctiptat in ordine dupa linii si Coloane
3. Se roteste la 90 de grade cheia de trei ori si se repeta acelas algoritm.

Decriptarea: euiaftiub t\_oa\_t\_\_e ctnresnoi sraeimiee

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s | e | c | u | r | i |
| t | a | t | e | a | \_ |
| i | n | f | o | r | m |
| a | t | i | e | i | \_ |
| e | s | t | e | \_ | u |
| n | \_ | o | b | i | e |

Sistemul POLYBIOS

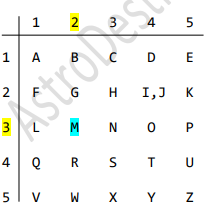
***#Introducere***

Cifrul Polybios este în esență identic cu cel mai simplu cifru de substituție, cu excepția faptului că fiecare caracter text simplu este cifrat ca 2 caractere criptate. De obicei, poate fi detectat numai dacă există 5 sau 6 caractere diferite în textul cifrat.

Acest algoritm oferă foarte puțină securitate în comunicare,și poate fi rupt ușor chiar și cu mâna, mai ales ca mesajele devin mai lungi (mai mult de câteva sute de caractere criptate).

***#Exemplu***

Iată un exemplu rapid al pașilor de criptare și decriptare implicați în cifrul Polybios. Textul pe care îl vom cripta este „WELCOMEEXPLORERS”.



Un exemplu de criptare folosind cheia de mai sus:

WELCOMEEXPLORERS

52153113343215155335313442154243

Este ușor de văzut cum este fiecare caracter din textul simplu înlocuit cu 2 caractere în alfabetul cifrat. Decriptarea este la fel de ușor, utilizând 2 caractere cifrate ca rând și coloană în pătratul cheie pentru a recupera caracterul original de text simplu.

***#COD***

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Criptarea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def polybiosEncrypt(s):

encpt=""

for char in s:

row = int((ord(char) -ord('a')) / 5) + 1

col = ((ord(char) -ord('a')) % 5) + 1

if char == 'k':

row = row -1

col = 5 -col + 1

elif ord(char) >= ord('j'):

if col == 1 :

col = 6

row = row -1

col = col -1

r=str(row)

c=str(col)

encpt = encpt+r+c

return encpt

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Decriptarea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def polybiosDecrypt(s):

s1 = list(s)

decpt = ""

for i in range(0,len(s),2):

r = int(s1[i])

c = int(s1[i+1])

ch = chr(((r-1)\*5+c+96))

if(ord(ch)-96>=10):

ch = chr(((r-1)\*5+c+96+1))

ch1 = str(ch)

decpt = decpt + ch1

return decpt

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Corpul\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

pt = input("Introduceti textul clar: ")

encpt = polybiosEncrypt(pt)

print("Textul criptat: ",encpt)

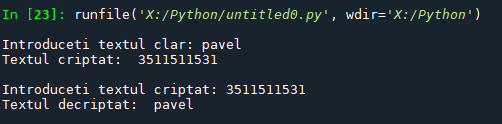
pt2 = input("Introduceti textul criptat: ")

decpt = polybiosDecrypt(pt2)

print("Textul decriptat: ",decpt)

***#Screenshots***





**Sistemul Vigenere**

***#Despre***

**Cifrul Vigenère** este o metodă de criptare care folosește o serie de cifruri Cezar diferite bazate pe literele unui cuvânt-cheie. Este o formă simplă de substituție polialfabetică.

Cifrul Vigenère a fost reinventat de multe ori. Metoda a fost descrisă inițial de către Giovan Batista Belaso în 1553, în cartea sa *La cifra del. Sig. Giovan Batista Belaso*; totuși, schema a fost atribuită greșit mai târziu lui Blaise de Vigenère în secolul al XIX-lea, și este cunoscută acuma ca "cifrul Vigenère".

Acest cifru este cunoscut deoarece deși este ușor de înțeles și implementat, pare pentru începători imposibil de spart; acestui fapt i se datorează descrierea **le chiffre indéchiffrable** (franceză, "cifrul indescifrabil"). În consecință, mulți oameni au încercat să implementeze scheme de criptare care sunt, esențialmente, cifruri Vigenère, doar ca să fie sparte.

***#COD***

Criptare

def encrypt(message, key): #functia de criptare

crypted.delete(0,tk.END)

message = clear\_message.get().lower()

key = cheie.get().lower()

encrypted = ""

split\_message = [

message[i : i + len(key)] for i in range(0, len(message), len(key))

] # despartim mesajul in bucati de lungimea key-ului

for each\_split in split\_message:

i = 0

for letter in each\_split:

number = (letter\_to\_index[letter] + letter\_to\_index[key[i]]) % len(alphabet)

#convertim fiecare litera in numarul de oridne adunam key-a (mod26)

encrypted += index\_to\_letter[number]

#convertim din numerele de ordine in literele corespunzatoare

i += 1

L3["text"]="MESAJUL CRIPTAT ESTE :"

crypted.insert(0,encrypted)

crypted.grid(row=5,column=2,sticky='ne') #returnam mesajul criptat

Decriptare

def decrypt(cipher, key):

cipher = crypted.get().lower()

key =reverse\_key.get().lower()

decrypted = ""

split\_encrypted = [

cipher[i : i + len(key)] for i in range(0, len(cipher), len(key))

] # despartim mesajul in bucati de lungimea key-ului

for each\_split in split\_encrypted:

i = 0

for letter in each\_split:

number = (letter\_to\_index[letter] - letter\_to\_index[key[i]]) % len(alphabet)

#convertim fiecare litera in numarul de oridne si scadem key-a (mod26)

decrypted += index\_to\_letter[number]

#convertim din numerele de ordine in literele corespunzatoare

i += 1

L8.grid(row=12,column=0,sticky='w')

clr\_message.insert(0,decrypted)

clr\_message.grid(row=12,column=2,sticky='ne') #returnam mesajul criptat

**Corpul**

# -\*- coding: utf-8 -\*-

"""

Created on Tue Nov 16 21:10:52 2021

@author: Pavel

"""

import tkinter as tk

root= tk.Tk()

root.resizable(0,0)

root.title('Sistemul "Vigenere" by Pavel Dordea')

root.configure(bg='#164A41')

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_pentru user input\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

L1=tk.Label(root,text="Introdu mesajul clar :",bg='#164A41',fg='#FFFFFF',font='Helvetica 14 bold')

clear\_message = tk.Entry(root,bg='white',fg='black',font='Helvetica 13 bold')

L2=tk.Label(root,text="Introdu cheia secreta :",bg='#164A41',fg='#FFFFFF',font='Helvetica 14 bold')

cheie = tk.Entry(root,bg='white',fg='black',font='Helvetica 13 bold')

alphabet = "abcdefghijklmnopqrstuvwxyz "

letter\_to\_index = dict(zip(alphabet, range(len(alphabet))))#transformam din litere in numarul de ordine din alfabet

index\_to\_letter = dict(zip(range(len(alphabet)), alphabet))#transformam din numarul de ordine din alfabet in litere

message =''

key=''

def show\_user():

L1.grid(row=2,column=0,sticky="w")

L2.grid(row=3,column=0,sticky="w")

clear\_message.grid(row=2,column=2,sticky='ne')

cheie.grid(row=3,column=2,sticky='ne')

L3.grid(row=5,column=0,sticky='w')

button\_crypt.grid(row=4,columnspan=3,sticky='we')

L4.grid(row=6,columnspan=3,sticky='w')

button\_show.grid(row=7,column=2,sticky='ne')

b1.grid\_forget()

l3.grid\_forget()

b2.grid\_forget()

def show\_file():

L1.grid\_forget()

L2.grid\_forget()

clear\_message.grid\_forget()

cheie.grid\_forget()

L3.grid\_forget()

button\_crypt.grid\_forget()

L4.grid\_forget()

button\_show.grid\_forget()

b1.grid(row=2,column=0,sticky='w')

l3.grid(row=2,column=1)

b2.grid(row=2,column=2,sticky='ne')

crypted.grid\_forget()

L5.grid\_forget()

button\_hide.grid\_forget()

L6.grid\_forget()

crypted\_message.grid\_forget()

L7.grid\_forget()

reverse\_key.grid\_forget()

button\_decrypt.grid\_forget()

L8.grid\_forget()

clr\_message.grid\_forget()

def encrypt(message, key): #functia de criptare

crypted.delete(0,tk.END)

message = clear\_message.get().lower()

key = cheie.get().lower()

encrypted = ""

split\_message = [

message[i : i + len(key)] for i in range(0, len(message), len(key))

] # despartim mesajul in bucati de lungimea key-ului

for each\_split in split\_message:

i = 0

for letter in each\_split:

number = (letter\_to\_index[letter] + letter\_to\_index[key[i]]) % len(alphabet)

#convertim fiecare litera in numarul de oridne adunam key-a (mod26)

encrypted += index\_to\_letter[number]

#convertim din numerele de ordine in literele corespunzatoare

i += 1

L3["text"]="MESAJUL CRIPTAT ESTE :"

crypted.insert(0,encrypted)

crypted.grid(row=5,column=2,sticky='ne') #returnam mesajul criptat

def decrypt(cipher, key):

cipher = crypted.get().lower()

key =reverse\_key.get().lower()

decrypted = ""

split\_encrypted = [

cipher[i : i + len(key)] for i in range(0, len(cipher), len(key))

] # despartim mesajul in bucati de lungimea key-ului

for each\_split in split\_encrypted:

i = 0

for letter in each\_split:

number = (letter\_to\_index[letter] - letter\_to\_index[key[i]]) % len(alphabet)

#convertim fiecare litera in numarul de oridne si scadem key-a (mod26)

decrypted += index\_to\_letter[number]

#convertim din numerele de ordine in literele corespunzatoare

i += 1

L8.grid(row=12,column=0,sticky='w')

clr\_message.insert(0,decrypted)

clr\_message.grid(row=12,column=2,sticky='ne') #returnam mesajul criptat

def show():

button\_show.grid\_forget()

L5.grid(row=8,columnspan=3,sticky='w')

button\_hide.grid(row=7,column=0,sticky='w')

L6.grid(row=9,column=0,sticky='w')

crypted\_message.grid(row=9,column=2,sticky='ne')

L7.grid(row=10,column=0,sticky='w')

reverse\_key.grid(row=10,column=2,sticky='ne')

button\_decrypt.grid(row=11,columnspan=3,sticky='we')

def hide():

L5.grid\_forget()

button\_show.grid(row=7,column=2,sticky='ne')

button\_hide.grid\_forget()

L6.grid\_forget()

crypted\_message.grid\_forget()

L7.grid\_forget()

reverse\_key.grid\_forget()

button\_decrypt.grid\_forget()

L8.grid\_forget()

clr\_message.grid\_forget()

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def show\_crypt():

key\_crypt.delete(0,tk.END)

L1['text']= 'Introdu cheia de criptare:'

L1.grid(row=3,column=0,sticky="w")

key\_crypt.grid(row=3,column=2,sticky='ne')

butt\_crypt.grid(row=4,columnspan=3,sticky='we')

l3["text"]=" "

def show\_decrypt():

key\_crypt.delete(0,tk.END)

L1['text']= 'Introdu cheia de decriptare:'

L1.grid(row=3,column=0,sticky="w")

key\_crypt.grid(row=3,column=2,sticky='ne')

butt\_decrypt.grid(row=4,columnspan=3,sticky='we')

l3["text"]=" "

def getmsg():

message = open('Message.txt', 'r+').read().lower().replace('\n',' ')

return message

def getkey():

cheie = key\_crypt.get()

return cheie

def fencrypt(): #functia de criptare

message=getmsg()

key =getkey()

encrypted = ""

split\_message = [

message[i : i + len(key)] for i in range(0, len(message), len(key))

] # despartim mesajul in bucati de lungimea key-ului

for each\_split in split\_message:

i = 0

for letter in each\_split:

number = (letter\_to\_index[letter] + letter\_to\_index[key[i]]) % len(alphabet)

#convertim fiecare litera in numarul de oridne adunam key-a (mod26)

encrypted += index\_to\_letter[number]

#convertim din numerele de ordine in literele corespunzatoare

i += 1

f = open("Encrypted.txt", "w+")

f.write(encrypted)#returnam mesajul criptat

l3["text"]="Success ! "

def getenc():

encrypted = open('Encrypted.txt', 'r+').read().lower().replace('\n',' ')

return encrypted

def fdecrypt():

cipher = getenc()

key =getkey()

decrypted = ""

split\_encrypted = [

cipher[i : i + len(key)] for i in range(0, len(cipher), len(key))

] # despartim mesajul in bucati de lungimea key-ului

for each\_split in split\_encrypted:

i = 0

for letter in each\_split:

number = (letter\_to\_index[letter] - letter\_to\_index[key[i]]) % len(alphabet)

#convertim fiecare litera in numarul de oridne si scadem key-a (mod26)

decrypted += index\_to\_letter[number]

#convertim din numerele de ordine in literele corespunzatoare

i += 1

f = open("Decrypted.txt", "w+")

f.write(decrypted)

l3["text"]="Success ! "

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_pentru user input\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

user = tk.Button(root,text="De la user",command=show\_user ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold')

user.grid(row=1,columnspan=3,padx=200,pady=10,sticky='we')

button\_crypt = tk.Button(root,text="Cripteaza",command=lambda: encrypt(message, key) ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold')

L3=tk.Label(root,text="",bg='#164A41',fg='red',font='Helvetica 12 bold')

crypted=tk.Entry(root,bg='white',fg='black',font='Helvetica 13 bold')

L4=tk.Label(root,text="\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",bg='#164A41',fg='#ffffff',font='Helvetica 12 bold')

button\_show = tk.Button(root,text="Mai mult",command=show ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 7 bold')

button\_hide = tk.Button(root,text="Mai putin",command=hide ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 7 bold')

L5=tk.Label(root,text="\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",bg='#164A41',fg='#ffffff',font='Helvetica 12 bold')

L6=tk.Label(root,text="Introdu mesajul criptat :",bg='#164A41',fg='#ffffff',font='Helvetica 12 bold')

crypted\_message=tk.Entry(root,bg='white',fg='black',font='Helvetica 13 bold')

L7=tk.Label(root,text="Introdu cheia secreta :",bg='#164A41',fg='#ffffff',font='Helvetica 12 bold')

reverse\_key=tk.Entry(root,bg='white',fg='black',font='Helvetica 13 bold')

button\_decrypt=tk.Button(root,text="Decripteaza",command=lambda: decrypt(message, key) ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold')

L8=tk.Label(root,text="MESAJUL CLAR ESTE :",bg='#164A41',fg='red',font='Helvetica 12 bold')

clr\_message = tk.Entry(root,bg='white',fg='black',font='Helvetica 13 bold')

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_pentru fisiere\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fisiere = tk.Button(root,text="Din fisier",command=show\_file ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold')

fisiere.grid(row=0,columnspan=3,padx=200,pady=10,sticky='we')

Lf1=tk.Label(root,text="",bg='#164A41',fg='#FFFFFF',font='Helvetica 14 bold')

key\_crypt = tk.Entry(root,bg='white',fg='black',font='Helvetica 13 bold')

b1 = tk.Button(root,text="Codificare",command= show\_crypt ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold')

l3=tk.Label(root,text=" ",bg='#164A41',fg='darkred',font='Helvetica 12 bold')

b2 = tk.Button(root,text="Decodificare",command= show\_decrypt ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold')

butt\_crypt = tk.Button(root,text="Cripteaza",command= fencrypt ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold')

butt\_decrypt = tk.Button(root,text="Decripteaza",command= fdecrypt ,bg='#4D774E',fg='#FFFFFF',font='Helvetica 12 bold')

root.mainloop()

***#Screenshots***

