TP N° 01_Cryptographie Artisanale

November 27, 2021

1 Élève : IRGUI Ilyas

Exercice 1: Chiffre de César

1)

[1]:
```
def chiffr_cesar(ch, k):
    ch = [ord(o)-65 for o in ch]
    l1 = [(i+k)%26 for i in ch]
    l2 = [chr(j+65) for j in l1]
    return "".join(l2)

chiffr_cesar("ATTAQUELECHATEAU", 12)
```

[1]:
```
'MFFMCGQXQOTMFQMG'
```

2)

[2]:
```
String = "ATTAQUELECHATEAU"
chiffre = chiffr_cesar(String, 12)
print("Chaîne chiffrée : "+ chiffre)
dechiffre = chiffr_cesar(chiffre, -12)
print("Chaîne déchiffrée : "+ dechiffre)
```

```
Chaîne chiffrée : MFFMCGQXQOTMFQMG
Chaîne déchiffrée : ATTAQUELECHATEAU
```

3)

[3]:
```
def cesar_cryptanalyse(String):
    for k in range(26):
        print(chiffr_cesar(String, -k))

cesar_cryptanalyse(chiffre)
```

```
MFFMCGQXQOTMFQMG
LEELBFPWPNSLEPLF
KDDKAEOVOMRKDOKE
JCCJZDNUNLQJCNJD
IBBIYCMTMKPIBMIC
HAAHXBLSLJOHALHB
```

```
GZZGWAKRKINGZKGA
FYYFVZJQJHMFYJFZ
EXXEUYIPIGLEXIEY
DWWDTXHOHFKDWHDX
CVVCSWGNGEJCVGCW
BUUBRVFMFDIBUFBV
ATTAQUELECHATEAU
ZSSZPTDKDBGZSDZT
YRRYOSCJCAFYRCYS
XQQXNRBIBZEXQBXR
WPPWMQAHAYDWPAWQ
VOOVLPZGZXCVOZVP
UNNUKOYFYWBUNYUO
TMMTJNXEXVATMXTN
SLLSIMWDWUZSLWSM
RKKRHLVCVTYRKVRL
QJJQGKUBUSXQJUQK
PIIPFJTATRWPITPJ
OHHOEISZSQVOHSOI
NGGNDHRYRPUNGRNH
```

Exercice 2: Chiffrement par Substitution Mono-Alphabétique

1)

[4]:
```
import string
from random import shuffle
def Tab():
    alpha = string.ascii_uppercase
    l = list(alpha)
    while True:
        shuffle(l)
        for a, b in zip(list(alpha), l):
            if a == b :
                break
        else:
            return("".join(l))

print(string.ascii_uppercase+"\t")

cle = Tab()
cle
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

[4]:
```
'DEPCBHFYMQOSNWVITLJZAGURXK'
```

2)

[5]:
```
def chiffr_monoalpha(msg, cle):
    alpha = string.ascii_uppercase
    chiffre = ""
    for l in msg:
        i = alpha.index(l)
        chiffre += cle[i]
    return chiffre

f0 = open('Msg.txt','r')
msg = f0.read()
chiffre = chiffr_monoalpha(msg, cle)
f1 = open('Chiffre_msg.txt','w')
f1.write(chiffre)
f1.close()
```

3)

[6]:
```
def frequence(msg,l):
    c = 0
    msg = msg.replace(" ","")
    for i in msg:
        if i == l:
            c += 1
    return (c/len(msg) * 100)
```

4)

[7]:
```
f1 = open('Chiffre_msg.txt','r')
chiffre = f1.read()

stats = ['E','S','A','I', 'N', 'T', 'R', 'U', 'L', 'O', 'D']

tab = {}
for i in chiffre :
    tab[i] = frequence(chiffre,i)
app = dict(sorted(tab.items(), key=lambda item: item[1], reverse=True))

l = list(app.keys())
print("{}-->{}".format(l,stats))

l1=[]
for a,b in zip(l, stats):
    a = b
    l1.append(a)

dech = []
for i in list(chiffre):
    for j in range(len(l)):
```

```
        if i == l[j]:
            dech.append(l1[j])

print("Le_vrai_message : {}, le_Chiffre : {}, Tentative_de_cryptanalyse : {}".
 ↪format(msg,chiffre,"".join(dech)))
```

```
['D', 'Z', 'B', 'A', 'T', 'S', 'P', 'Y']-->['E', 'S', 'A', 'I', 'N', 'T', 'R',
'U', 'L', 'O', 'D']
Le_vrai_message : ATTAQUELECHATEAU, le_Chiffre : DZZDTABSBPYDZBDA,
Tentative_de_cryptanalyse : ESSENIATARUESAEI
```

Exercice 3 : Chiffrement de Vigenère

[8]:
```
def vigenere(msg, cle):
    chiffre=""
    i=0
    for l in msg:
        a = ord(l) - 65
        b = (a + cle[i])%26
        c = chr(b + 65)
        i = (i + 1)%len(cle)
        chiffre += c
    f = open("chiffre_vigenere.txt", "w")
    f.write(chiffre)
    f.close
    return chiffre

vig = vigenere("ATTAQUELECHATEAU", [2,9,8,12])
```

[9]:
```
def dechiffr_vigenere(msg, cle):
    dechiffre=""
    i=0
    for l in msg:
        a = ord(l) - 65
        b = (a - cle[i]) % 26
        c = chr(b + 65)
        i = (i + 1) % len(cle)
        dechiffre += c
    return dechiffre

dechiffr_vigenere(vig,[2,9,8,12])
```

[9]:
```
'ATTAQUELECHATEAU'
```

Cette fonction rassemble le chiffrement et le déchiffrement de vigenere en indiquant le paramètre booléen “decoder”

[10]:
```
def vigenere ( msg, cle, decoder = False) :
    output=""
    i=0
    for l in msg:
        a = ord(l) - 65
        d = cle[i]
        if decoder : d = - cle[i]
        b = (a + d) % 26
        c = chr(b + 65)
        i = (i + 1) % len(cle)
        output += c
    return output

vigenere(vig, [2,9,8,12], True)
```

[10]:
```
'ATTAQUELECHATEAU'
```

Exercice 4: Chiffre Homophone

1)

[11]:
```
import random
def homo_tab():
    alpha = list(string.ascii_uppercase)
    st = [8,1,3,3,17,1,1,1,7,1,1,5,2,7,5,2,1,6,8,7,6,1,1,1,1,1]
    tab = {}
    l = [o for o in range(1, 100)]
    for i,j in zip(st,alpha):
        l1 = random.sample(l, k = i)
        tab[j]=l1
        for k in l1:
            if k in l:
                l.remove(k)
    return tab

tab = homo_tab()
tab
```

[11]:
```
{'A': [5, 32, 27, 74, 6, 94, 67, 33],
 'B': [36],
 'C': [23, 4, 49],
 'D': [14, 28, 66],
 'E': [85, 93, 31, 68, 82, 37, 92, 54, 29, 15, 84, 46, 57, 44, 16, 75, 88],
 'F': [47],
 'G': [21],
 'H': [41],
 'I': [48, 38, 43, 8, 7, 26, 96],
 'J': [65],
```

```
 'K': [99],
 'L': [17, 58, 73, 79, 62],
 'M': [90, 52],
 'N': [30, 35, 39, 64, 95, 60, 12],
 'O': [9, 11, 34, 91, 89],
 'P': [18, 1],
 'Q': [71],
 'R': [51, 70, 87, 25, 61, 97],
 'S': [78, 20, 42, 77, 2, 69, 53, 10],
 'T': [19, 55, 24, 83, 81, 22, 86],
 'U': [72, 45, 63, 3, 59, 56],
 'V': [98],
 'W': [76],
 'X': [40],
 'Y': [50],
 'Z': [80]}
```

2)

[12]:
```
def chiffr_homo(msg,tab):
    c = []
    i = 0
    for l in msg:
        nbr = len(tab[l])
        c.append(tab[l][i%nbr])
        i += 1
    chiffre = " ".join([str(j) for j in c])
    return chiffre

chiffr_homo("ATTAQUELECHATEAU", tab)
```

[12]:
```
'5 55 24 74 71 56 92 73 29 23 41 74 22 44 67 3'
```

3.a)

[13]:
```
import numpy as np
def carre_polybe():
    carre = np.empty((5,5), str)
    alpha = list(string.ascii_uppercase.replace('W', ''))
    for i in range(5):
        carre[i] = list(random.sample(alpha, k = 5))
        for k in carre[i]:
            if k in alpha:
                alpha.remove(k)
    return carre

carre = carre_polybe()
carre
```

[13]:
```
array([['D', 'V', 'J', 'Y', 'M'],
       ['A', 'E', 'H', 'X', 'N'],
       ['K', 'R', 'I', 'F', 'Z'],
       ['T', 'S', 'U', 'O', 'P'],
       ['C', 'Q', 'B', 'L', 'G']], dtype='<U1')
```

[14]:
```
def chiffr_polybe(msg, carre):
    tab = {}
    chiffr = []
    a = 0
    for t in msg:
        i,j = list(zip(np.where(carre==t)[0],np.where(carre==t)[1]))[0]
        rs = []
        rl = []

        for i1 in range(5):
            for j1 in range(5):
                if ((i1!=i) and (j1!=j)):
                    rs.append(((i,j1),(i1,j)))

        for result in rs:
            rl.append(str(carre[result[0]]+carre[result[1]]))

        tab[t] = rl

        chiffr.append(tab[t][a%16])
        a += 1

    return chiffr

chiffr_polybe("ATTAQUELECHATEAU", carre)
```

[14]:
```
['ED',
 'UD',
 'OD',
 'ND',
 'CE',
 'SH',
 'XR',
 'GX',
 'AS',
 'BK',
 'XU',
 'NT',
 'SC',
 'HQ',
 'XC',
```

```
 'PB']
```

3.b)

[15]:
```
carre_2 = np.empty((6,6), dtype='O')
carre_2[0] = ['','12345', '6', '7', '8', '9']
carre_2[1] = ['12345','E','A','S','T','R']
carre_2[2] = ['6','I','B','C','D','F']
carre_2[3] = ['7','L','G','H','K','M']
carre_2[4] = ['8','N','O','P','Q','V']
carre_2[5] = ['9','U','W','X','Y','Z']
carre_2
```

[15]:
```
array([['', '12345', '6', '7', '8', '9'],
       ['12345', 'E', 'A', 'S', 'T', 'R'],
       ['6', 'I', 'B', 'C', 'D', 'F'],
       ['7', 'L', 'G', 'H', 'K', 'M'],
       ['8', 'N', 'O', 'P', 'Q', 'V'],
       ['9', 'U', 'W', 'X', 'Y', 'Z']], dtype=object)
```

[16]:
```
def Alternative_2(msg, carre):
    chiffr = ''
    for t in msg:
        i,j = list(zip(np.where(carre==t)[0],np.where(carre==t)[1]))[0]


        chiffr += str(random.choice(carre[i][0])+random.choice(carre[0][j]))

    return chiffr

Alternative_2("ATTAQUELECHATEAU", carre_2)
```

[16]:
```
'36581816889521725367773618313695'
```

Exercice 5: Chiffre de Hill (m = 2)

1)

[17]:
```
def pgcd(a,b):
    return a if b==0 else pgcd(b,a%b)

def inverse_nbr(n):
    if pgcd(n,26) == 1:
        return 1/n
    else:
        return 0
```

2)

[18]:
```
def generate_matrix():
    while True:
        matrice = np.random.randint(5, size = (2,2))
        det = np.linalg.det(matrice)
        if inverse_nbr(det) != 0:
            break
    return matrice

q = generate_matrix()
q
```

[18]:
```
array([[2, 3],
       [1, 2]])
```

3)

[19]:
```
def chiffr_hill(msg, q):
    y = []
    blocs = []
    for i in range(0, len(msg), 2):
        blocs.append([(ord(msg[i])-65), (ord(msg[i+1])-65)])

    for bloc in blocs:
        rs = (q.dot(bloc)%26).tolist()
        rs = [chr(r+65) for r in rs]

        y.append("".join(rs))

    return "".join(y)

hill = chiffr_hill("ATTAQUELECHATEAU",q)
hill
```

[19]:
```
'FMMTOEPAOIOHYBIO'
```

4)

[20]:
```
def dechiffr_hill(msg, q):
    y = []
    blocs = []
    for i in range(0, len(msg), 2):
        blocs.append([(ord(msg[i])-65), (ord(msg[i+1])-65)])

    for bloc in blocs:
        rs = ((np.linalg.inv(q).dot(bloc))%26).tolist()
        rs = [chr(int(r)+65) for r in rs]

        y.append("".join(rs))
```

```
    return "".join(y)

dechiffr_hill(hill,q)
```

[20]: 'ATTAQUELECHATEAU'

5)

[21]:
```
msg = "ILYASIRGUI"
print("Message en clair :", msg)
message_code = chiffr_hill(msg, q)
print ("message chiffré :", message_code)
message_decode = dechiffr_hill(message_code, q)
print ("message déchiffré :", message_decode)
```

```
Message en clair : ILYASIRGUI
message chiffré : XEWYIIADMK
message déchiffré : ILYASIRGUI
```