DS02

Algorithmique et programmation

Sources : exercice 1 : Clément Roux -UPSTI exercice 2 : Clément Roux -UPSTI

Exercice 1 : Décryptage de texte

1.1 Indication sur les méthodes associées aux chaînes de caractères

Les attributs suivants s'appliquent à des variables de type de chaîne de caractère :

- `.isalpha` renvoie `True` si c'est une des 26 lettres de l'alphabet et `False` sinon.

```
>>> 'c'.isalpha()
True
>>> '1'.isalpha()
False
```

- `.index(x)` renvoie l'indice de la première occurrence de `x` :

```
>>> 'hello'.index('l')
2
```

- `.count(x)` renvoie le nombre d'occurrences de `x` :

```
>>> 'hello'.count('l')
2
```

1.2 Le chiffre de César

Le **chiffrement de César** est un des tout premier code de chiffrement qui ait existé.

La méthode est simple : il suffit de décaler toutes les lettres de l'alphabet du même nombre de lettres.

Par exemple en choisissant un décalage de 3, le A devient le D, le B devient le E, le C devient le F et ainsi de suite. Pour la fin de l'alphabet, il suffit de revenir au début : le W devient Z, le X devient A, le Y devient B et le Z devient C. Ainsi un message comme « la metamorphose » devient par décalage d'une lettre « mb nfubnpsqiptf », ce qui est incompréhensible pour le non initié.

Q 1 : Avec des instructions python, définir trois chaînes de caractères nommées *alphabet*, *mess* et *code* contenant respectivement les caractères de l'alphabet, un message à chiffrer (exemple « la metamorphose ») et le message crypté (vide initialement). On se limitera à des lettres minuscules non accentuées. Les autres caractères (espaces, chiffres...) seront gardés tels quels (non chiffrés).

Q 2 : Comment est codé le message « franz » avec une valeur de décalage $n = 3$?

Q 3 : Écrire une fonction `def decalage(c:str, n:int) -> str:` permettant de renvoyer un caractère chiffrer avec le chiffrement de César.

Q 4 : Etablir un algorithme permettant de chiffrer un message par le code de César, pour un décalage n donné. La fonction associée à cet algorithme aura la signature suivante :

```
def chiffrement_cesar(mess:str, n:int) -> str:.
```

Q 5 : Proposer ensuite l'algorithme de déchiffrement qui affiche le message chiffré en clair, en supposant que n est inconnu. L'utilisateur choisira parmi les déchiffrements proposés celui qui a du sens! La fonction aura la signature suivante : `def decryptage_cesar(code:str) -> None:`.

Q 6 : Quelle est la faiblesse de ce type de code? Proposer en deux lignes une méthode permettant de déterminer (casser) la clé.

1.3 Le chiffre de Vigenère

On étudie maintenant la méthode de chiffrement de Vigenère. Pour comprendre le système de chiffrement, on commence par créer la table Vigenère : on recopie 26 fois l'alphabet en ligne et à chaque saut de ligne on décale d'une lettre vers la droite comme sur la figure 1.
Cet algorithme fonctionne à l'aide d'une clé de chiffrement : une chaîne de caractère (typiquement un mot).
Prenons un exemple, mettons que l'on veuille coder « anticonstitutionnellement » avec la clé « roue » . Pour ce faire, on crée le tableau présenté table 1 dont la première ligne contient le message à coder, et la deuxième la clé répétée autant de fois que nécessaire pour atteindre la longueur du texte à coder. On fait alors une lecture colonne par colonne de ce tableau et dans chaque colonne :

1. la première ligne donne l'abscisse dans la table de la lettre à coder;
2. la deuxième donne l'ordonnée dans la table de la lettre à coder;

Ainsi le chiffrement du mot « anticonstitutionnellement » avec la clé « roue » est obtenue dans la table 1.

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

FIGURE 1 – Table de Vigenère

mot	a	n	t	i	c	o	n	s	t	i	t	u	t	i	o	n	n	e	l	l	e	m	e	n	t
clé	r	o	u	e	r	o	u	e	r	o	u	e	r	o	u	e	r	o	u	e	r	o	u	e	r
code	r	b	n	m	t	c	h	w	k	w	n	y	k	w	i	r	e	s	f	p	v	a	y	r	k

TABLE 1 – Exemple de chiffrement de Vigenère

La méthode de chiffrement par lecture de la table est une méthode adaptée « aux humains », ainsi on réfléchira à l'implémentation d'un algorithme plus efficace en s'aidant de l'exemple.

On dispose de la variable `alphabet= "abcdefghijklmnopqrstuvwxyz"`.

On donne le bloc d'instruction suivant :

```
alphabet= "abcdefghijklmnopqrstuvwxyz"
for i in range(26):
    alphabet = alphabet[1:]+alphabet[:1]
    print (alphabet)
```

Q 7 : Combien de lignes seront affichées? Quelles seront les deux premières lignes affichées?

Q 8 : Écrire la fonction `def generer_table()->list` : qui retourne la table de Vigenère sous la forme d'une liste de listes. Le résultat sera de la forme suivante :

```
[['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'],
 ['b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','a'],
 ... ]
```

On donne deux fonctions permettant de constuire la ligne « clé » de la table 1 à partir d'un mot et d'une clé.

```
1. def generer_cle_1(mot,cle):
2.     nb = len(mot)//len(cle)+1
3.     ch_cle=nb*cle
4.     ch_cle = ch_cle[0:len(mot)]
5.     tab_cle = [car for car in ch_cle]
6.     return tab_cle
```

```
1. def generer_cle_2(mot,cle):
2.     tab_cle = []
3.     for i in range(len(mot)):
4.         id = i%len(cle)
5.         tab_cle.append(cle[id])
6.     return tab_cle
```

Q 9 : En 2 à 3 lignes, expliquer les différences entre les 2 fonctions. Commentez les lignes 2 à 5 des deux fonctions.

Q 10 : Écrire une fonction étant spécifier ainsi : `code_vigenere(ch:str, cle:str) -> str` **où la chaîne renvoyée correspond à la chaîne codée avec la clé** `cle`**. Chaque ligne sera commentée. Vous pourrez utiliser les fonctions définies précédemment.**

Q 11 : Quel est selon vous l'intérêt de ce codage par rapport à l'algorithme de César?

Exercice 2 : Les carrés magiques

Tout d'abord, qu'est ce qu'un carré magique? Selon Wikipédia en voici la définition : *« En mathématiques, un carré magique d'ordre n est composé de n^2 entiers strictement positifs, écrits sous la forme d'un tableau carré. Ces nombres sont disposés de sorte que leurs sommes sur chaque rangée, sur chaque colonne et sur chaque diagonale principale soient égales. On nomme alors constante magique (et parfois densité) la valeur de ces sommes. Un carré magique normal est un cas particulier de carré magique, constitué de tous les nombres entiers de 1 à n^2, où n est l'ordre du carré, sa densité est de $n \cdot \frac{n^2+1}{2}$. »*

					↗	65
17	24	1	8	15	→	65
23	5	7	14	16	→	65
4	6	13	20	22	→	65
10	12	19	21	3	→	65
11	18	25	2	9	→	65
					↘	
						65

TABLE 2 – Exemple de carré magique avec $n = 5$, la densité est de 65.

Dans le cas d'un carré magique normal et d'une valeur de n impaire, il existe une méthode simple de construction.

1. Nous notons x et y les numéros de colonne et de ligne $(x, y) \in [0, 1, \cdots, n-1]^2$.
2. Dans tous les carrés impairs, il y a une case centrale située de coordonnée $((n-1)/2, (n-1)/2)$, on commence par remplir avec le chiffre 1, la cellule juste à gauche de cette cellule centrale.
3. On continue ensuite à remplir les autres cases avec la suite des entiers jusqu'à n^2, en suivant les règles suivantes, à partir des coordonnées (x, y) :
   - si le chiffre que l'on vient de placer était un multiple de n on place le nouveau chiffre en $(x-2, y)$ **modulo** n ;
   - sinon on place le nouveau chiffre à la case de coordonnées $(x-1, y-1)$ **modulo** n.

On prendra soin de représenter ce carré magique, qui est une matrice, à l'aide d'une liste de listes. Par exemple, pour le carré magique de taille 3 suivant on utilisera le code suivant :

Code : `A=[[2,7,6],[9,5,1],[4,3,8]]` et **Résultat :** $A = \begin{pmatrix} 2 & 7 & 6 \\ 9 & 5 & 1 \\ 4 & 3 & 8 \end{pmatrix}$

				3
2				
	1			
6		5		
			4	

TABLE 3 – Exemple de carré magique incomplet de taille 5 illustrant la méthode proposée.

Q 12 : Continuez de compléter la carré magique de la table 3 en utilisant la méthode proposée. Testez les 3 propriétés du carré magique sur cet exemple.

Q 13 : Proposer une fonction `def Carre_vide(n:int) -> list` **: qui crée et renvoie un carré magique vide (rempli de 0), et qui renvoie une liste vide si n est pair.**

Q 14 : Proposer une fonction `def Remplir_carre(CarreVide : list) -> None` **: qui complète et renvoie un carré magique à partir d'un carré vide.**

Q 15 : Proposer une fonction `def Verif_carre(Carre : list) -> bool` **: qui vérifie les trois propriétés d'un carré magique et renvoie un booléen** (`True` **ou** `False`)