

CHIFFREMENT ET CRYPTOGRAPHIE

Exercice 1 : Cryptage affine

Chacune des 26 lettres est associée à l'un des entiers de 0 à 25, selon le tableau de correspondance suivant.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Le cryptage affine se fait à l'aide d'une clé, qui est un nombre entier k fixé, compris entre 1 et 25. Pour crypter une lettre donnée on suit le processus ρ suivant :

- on repère le nombre x associé à la lettre dans le tableau de correspondance précédent
- on multiplie ce nombre x par la clé k
- on calcule le reste r de la division euclidienne du nombre obtenu par 26
- on repère la lettre associée au nombre r dans le tableau de correspondance, qui devient la lettre cryptée

Par exemple, pour crypter la lettre P avec la clé $k = 11$:

- le nombre x associé à la lettre P est le nombre 15
- on multiplie 15 par la clé k , ce qui donne $11 \times 15 = 165$
- on calcule le reste de la division euclidienne par 26 : on obtient $165 \% 26 = 9$
- on repère finalement la lettre associée à 9 dans le tableau, c'est-à-dire J

Ainsi avec la clé $k = 11$, la lettre P est cryptée en la lettre J.

On crypte un mot en cryptant chacune des lettres de ce mot. En Python, on crée une liste L qui contient les 26 lettres de l'alphabet rangées dans l'ordre alphabétique à l'aide de la commande ci-dessous :

```
L = ['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']
```

1. Que penser du cryptage obtenu lorsque la clé k est égale à 1 ?
2. En quoi la lettre A constitue-t-elle un cas particulier dans le processus de cryptage ?
3. Dans le cas où la clé est égale à 11, crypter le mot MIRO.
4. Ecrire une fonction **indice** qui prend en paramètre une lettre de l'alphabet et qui renvoie son indice dans la liste L (L étant supposée définie comme variable globale).
5. Ecrire une fonction **crypterLettre** qui prend en paramètre une chaîne de caractères constituées d'une lettre majuscule de l'alphabet et une clé et qui renvoie la lettre cryptée. Cette fonction utilisera la fonction

indice précédente. En supposant qu'un appel à la fonction **indice** compte pour une opération élémentaire, quel est le nombre d'opérations élémentaires effectuées à chaque appel de la fonction **crypterLettre** ?

6.

(a) Ecrire une fonction **crypterTexte** qui prend en paramètre une chaîne de caractères dont les éléments sont soit des lettres majuscules soit des espaces (qui ne seront pas modifiés), et une clé. La fonction renvoie la chaîne de caractères cryptés.

(b) Donner l'appel à effectuer pour répondre à la question 3

(c) Soit N la longueur de la chaîne de caractères en paramètre de la fonction **crypterTexte** et M le nombre d'espaces qu'elle contient. Déterminer en fonction de N et M la complexité de la fonction **crypterTexte**.

7. On dit qu'une clé est une bonne clé de cryptage si elle possède une clé associée k' , qui est un nombre entier compris entre 1 et 25 tel qu'en appliquant le processus \wp avec la clé k' à une lettre cryptée (avec la clé k) on obtient la lettre initiale. k' est alors appelée clé de décryptage associée à k .

On admet qu'une clé k est une bonne clé de cryptage si et seulement si $k \neq 1$ et $k \wedge 26 = 1$, c'est-à-dire si k est différent de 1 et si le seul diviseur commun dans \mathbb{N} à k et à 26 est 1.

(a) On suppose que 19 est une clé de décryptage associée à la clé $k = 11$. Avec la clé $k = 11$, un mot a été crypté. On a obtenu le mot HARK. Retrouver le mot initial à l'aide de la clé de décryptage « à la main », puis donner l'appel à la fonction **crypterTexte** donnant le même résultat.

(b) Soit F la liste qui contient les termes de la suite de Fibonacci strictement inférieurs à 26 rangés par ordre croissant. On rappelle que la suite de Fibonacci est définie par ses deux premiers termes 0 et 1 et par le fait qu'à partir de son troisième terme, chaque terme est égal à la somme des deux précédents. Expliciter F .

(c) Déterminer la liste G des éléments de F qui sont de bonnes clés de cryptage.

(d) On admet que la clé de décryptage k' associée à une bonne clé de cryptage k est unique et que c'est le nombre entier strictement compris entre 0 et 26 qui est tel que le reste de la division euclidienne par 26 de $k \times k'$ est 1. Vérifier que 19 est la clé de décryptage associée à la clé $k = 11$.

8. Ecrire une fonction **cleDecryptage** qui prend en paramètre un entier k premier avec 26 et qui renvoie la clé de décryptage associée k' . On dispose de bonnes clés de cryptage avec les éléments de G , la fonction **cleDecryptage** nous permet ainsi de calculer aussi les clés de décryptage associées.

9. Un petit futé cherche à décrypter un long message crypté écrit en français sans connaître ni la clé de cryptage, ni la clé de décryptage. Pour cela, il repère dans la chaîne de caractères que constitue le message la lettre la plus fréquente et en déduit que c'est la lettre cryptée qui correspond à E (la lettre la plus utilisée en français).

(a) Que renvoie la fonction suivante ? On ne justifiera pas la réponse.

```
import numpy as np
def mystere (C) :
    """ C est une chaîne de caractères dont les éléments sont soit des lettres
    majuscules, soit des espaces """
    V = np.zeros(26)
```

```

max, indexMax = 0, 0
for j in range(26):
    for i in range(len(C)):
        if C[i] == L[j]:
            V[j] += 1
    if V[j] > max:
        max, indexMax = V[j], j
return L[indexMax]

```

(b) Donner une preuve rapide de la terminaison de cette fonction.

(c) *Question bonus.* En utilisant les fonctions **count**, **max** et **index** de Python, réécrire la fonction précédente avec le moins de lignes possible, et donner lui aussi un nom plus explicite.

10. Connaissant E et la lettre cryptée correspondante, le petit futé peut en déduire la clé de cryptage et ainsi « cracker » le code. Ecrire une fonction **cracker** qui prend en paramètre la lettre cryptée correspondant à E et qui renvoie la liste des clés de cryptage possibles (liste pouvant être vide) permettant de crypter E en cette lettre.

11. Que se passe-t-il s'il essaie de « cracker » un message crypté où la lettre E n'est pas la lettre la plus fréquente du message ?

Exercice 2 : Codage de César

On cherche à crypter un texte t de longueur n composé de caractères en minuscules (soit 26 lettres différentes) représentés par des entiers compris entre 0 et 25 (0 pour a, 1 pour b, etc.). Nous ne tenons pas compte des éventuels espaces. Ainsi, le texte **capemathoptioninfo** est représenté par le tableau suivant où la première ligne représente le texte, la seconde les entiers correspondants, et la troisième les indices dans le tableau t .

c	a	p	e	s	m	a	t	h	o	p	t	i	o	n	i	n	f	o
2	0	15	4	18	12	0	19	7	14	15	19	8	14	13	8	13	5	14
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Dans cette partie, on supposera que les textes sont déjà représentés par des listes de nombres et que donc toutes les fonctions de codage et décodage prendront en arguments des listes de nombres et renverront des listes de nombres.

Le codage de César est le plus rudimentaire que l'on puisse imaginer. Il a été utilisé par Jules César (et même auparavant) pour certaines de ses correspondances. Le principe est de décaler les lettres de l'alphabet vers la droite d'une ou plusieurs positions. Par exemple, en décalant les lettres d'une position, le caractère a se transforme en b, le b en c, etc. et le z en a. Le texte **avecésar** devient donc **bwfdftbs**.

1. Que donne le codage du texte **info** en utilisant un décalage de 5 ?

2. Ecrire une fonction **codageCesar** qui prend en paramètres une liste t d'entiers (représentant un texte) et un entier d et qui retourne une liste de même taille que t représentant le texte décalé de d positions.

3. Ecrire de même une fonction **decodageCesar** qui prend les mêmes paramètres mais qui réalise le décalage dans l'autre sens (vers la gauche).

Pour réaliser ce décodage, il faut connaître la valeur de décalage. Une manière de la déterminer automatiquement est d'essayer de deviner cette valeur (afin de « cracker » le code). L'approche la plus couramment employée consiste à regarder la fréquence d'apparition de chaque lettre dans le texte crypté. On utilise ensuite encore le fait que la lettre e est la plus fréquente en français.

4. Ecrire une fonction **frequenceLettres** qui prend en paramètre une liste t représentant le texte crypté et qui retourne une liste de taille 26 dont la case d'indice i contient le nombre d'apparitions du nombre i dans t divisé par la longueur de t.

5. Ecrire une fonction **decodageAuto** qui prend en argument une liste t représentant le texte crypté, et qui retourne le texte d'origine (en calculant le décalage pour que la lettre e soit la plus fréquente dans le message décrypté).

Exercice 3 : Codage ROT13

Le ROT13 (rotate by 13 places) est un cas particulier du codage de César. Comme son nom l'indique, il s'agit d'un décalage de 13 caractères de chaque lettre du texte à chiffrer.

1. Modifier le code du codage de César pour écrire la fonction **codageROT13** qui prend en paramètre une liste t d'entiers (représentant un texte) et qui retourne une liste de même taille que t représentant le texte codé par le codage ROT13.

2. Donner le code de la fonction **decodageROT13** qui prend le même paramètre mais qui réalise le décodage. Que remarquez-vous ? Quel est le principal avantage et le principal défaut de ce codage ?

ROT47 est une variante de ROT13 qui permet de chiffrer les lettres, les chiffres, et les autres caractères spéciaux. ROT47 se base sur le code ASCII, qui assigne à chaque symbole un nombre. Il utilise les nombres dans l'intervalle 33 à 126 (caractère '!' au caractère '~'). Ensuite, selon le même principe que le codage ROT13, il effectue une rotation de 47 lettres. Le fait d'utiliser un panel de symboles plus important que le ROT13 permet un brouillage plus efficace, même si le code reste trivial à déchiffrer.

Exercice 4 : Codage de Vigenère

Au XVI^e siècle, Blaise de Vigenère a modernisé le codage de César, très peu résistant, de la manière suivante. Au lieu de décaler toutes les lettres du texte de la même manière, on utilise un texte clé qui donne une suite de décalages. Prenons par exemple la clé **concours**. Pour crypter un texte, on code la première lettre en utilisant le décalage qui renvoie a sur c (la première lettre de la clé). Pour la deuxième lettre, on prend le décalage qui renvoie le a sur le o (deuxième lettre de la clé) et ainsi de suite. Pour la neuvième lettre du texte (la clé en contient que huit) on reprend à partir de la première lettre de la clé. Sur l'exemple **capemathoptioninfo** avec la clé **concours**, on obtient :

c	a	p	e	s	m	a	t	h	o	p	t	i	o	n	i	n	f	o
2	0	15	4	18	12	0	19	7	14	15	19	8	14	13	8	13	5	14
c	o	n	c	o	u	r	s	c	o	n	c	o	u	r	s	c	o	n
2	14	13	2	14	20	17	18	2	14	13	2	14	20	17	18	2	14	13
4	14	2	6	6	6	17	1	9	2	2	21	22	8	4	0	15	19	1
e	o	c	g	g	g	r	b	j	c	c	v	w	i	e	a	p	t	b

La quatrième ligne donne le code de la clé (décalage à appliquer), la cinquième ligne donne la somme de la deuxième et la quatrième (modulo 26), la sixième ligne donne le texte crypté.

1. Ecrire une fonction **codageVigenere** qui prend en paramètres une liste représentant le texte à crypter et une liste d'entiers donnant la clé servant au codage, et qui retourne une liste contenant le texte crypté.

Afin d'essayer de deviner le texte original sans la clé (de « cracker » le code), on procède en deux temps. D'abord on détermine la longueur k de la clé c , ensuite on détermine les lettres composant c .

La première étape est la plus difficile. On remarque que deux lettres identiques dans t , le texte crypté, espacées de $l \times k$ caractères (l entier) sont codées par la même lettre dans t . Cette condition n'est pas suffisante pour déterminer la longueur k de la clé c car des répétitions peuvent apparaître dans t sans qu'elles existent dans le texte original (les trois g consécutifs dans l'exemple ont pour origine trois lettres différentes). De plus, la même lettre dans le texte original peut être cryptée différemment dans t .

Nous allons rechercher des répétitions non pas d'une lettre mais de séquences de lettres dans t puisque deux séquences de lettres répétées dans le texte original, dont les premières lettres sont espacées par $l \times k$ caractères sont aussi cryptées par deux mêmes séquences dans t .

Dans la suite de l'énoncé, on ne considère que des séquences de taille 3 en supposant que toute répétition d'une séquence de trois lettres dans t provient exclusivement d'une séquence de trois lettres répétées dans le texte original. Ainsi, la distance séparant ces répétitions donne des multiples de k (i.e. les entiers l). La valeur de k est obtenue en prenant le PGCD de tous ces multiples. Si le nombre de répétitions est suffisant (i.e. si le texte est assez long), on a de bonnes chances d'obtenir la valeur de k . On suppose donc que cette assertion est vraie.

2. Ecrire une fonction **pgcd** qui calcule le PGCD de deux entiers positifs ou nuls passés en paramètres, et ce par soustractions successives.

3. Ecrire une fonction **pgcdDistancesEntreRepetitions** qui prend en paramètres le texte crypté t de longueur n et un entier $i \in [0, n - 3]$ qui est l'indice d'une lettre dans t . La fonction retourne le PGCD de toutes les distances entre les répétitions de la séquence de 3 lettres $t[i]$, $t[i+1]$, $t[i+2]$ dans le texte $t[i+3]$ $t[i+4]$... $t[n-1]$. Cette fonction retourne 0 s'il n'y a pas de répétition.

4. Ecrire une fonction **longueurCle** qui prend en paramètres le texte crypté t et qui retourne la longueur k probable de la clé de codage.

5. Donner le nombre maximal d'opérations réalisées par la fonction **longueurCle** en fonction de la longueur de t . On ne comptera que le nombre d'appels à la fonction **pgcd**.

6. Une fois la longueur de la clé connue, donnez une idée d'algorithme permettant de retrouver chacune des lettres de la clé. Il s'agit de décrire rapidement l'algorithme et non d'écrire le programme.

7. *Question bonus.* Ecrire une fonction **decodageVigenereAuto** qui prend en paramètres le texte crypté `t` et qui retourne le texte original probable.

Exercice 5 : Codage XOR

Le cryptage XOR est un système de cryptage basique mais pas trop limité. Ainsi, il a beaucoup été utilisé dans les débuts de l'informatique et continue à l'être encore aujourd'hui car il est facile à implémenter, dans toutes sortes de programmes.

Le XOR est un opérateur logique qui correspond à un "OU exclusif" : c'est le (A ou B) qu'on utilise en logique mais qui exclue le cas où A et B sont simultanément vrais. Voici sa table de vérité :

A	B	A XOR B
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	FAUX

Chaque caractère du message à coder est représenté par un entier, le code ASCII. Ce nombre est lui-même représenté en mémoire comme un nombre binaire à 8 chiffres (les bits). On choisit une clé que l'on place en dessous du message à coder, en la répétant autant de fois que nécessaire, comme dans le codage de Vigenère. Le message et la clé étant converti en binaire, on effectue un XOR, bit par bit, le 1 représentant VRAI et le 0 FAUX. Le résultat en binaire peut être reconverti en caractères ASCII et donne alors le message codé.

Exemple avec le texte « MESSAGE » et la clé « CLE » :

Lettre	M	E	S	S	A	G	E
Code ASCII	77	69	83	83	65	71	69
Code binaire	01001101	01000101	01010011	01010011	01000001	01000111	01000101
Clé en binaire	01000011	01001100	01000101	01000011	01001100	01000101	01000011
Lettre codée bin.	00001110	00001001	00010110	00010000	00001101	00000010	00000110
Lettre codée ascii	SO Shift out	HT Horizontal tab	SYN Sync. Idle	DLE Data Link Esc.	CR Carriage Return	STX Start of Text	ACK Acknowl..

1. Donner l'expression permettant de calculer le caractère codé d'une lettre **l** en utilisant le principe de codage XOR avec une lettre **c** d'une clé.

2. Ecrire une fonction **codageXOR** qui prend en paramètres le texte à crypter et la clé servant au codage, et qui retourne le texte crypté selon le codage XOR.

3. Expliquer quelle serait la méthode de décodage connaissant la clé.

Exercice 6 : Codage RSA

Le but de cette partie est de présenter le codage RSA (du nom de ses inventeurs Rivest, Shamir et Adleman). Ce système découvert dans les années 1970 est l'un des plus utilisés actuellement. C'est un codage asymétrique : la clé de codage et la clé de décodage sont différentes. La clé de codage est publique, tandis que la clé de décodage est privée et très difficile en pratique à obtenir à partir de la clé publique.

Le chiffrement RSA est le plus souvent utilisé pour communiquer des clés symétriques afin de poursuivre l'échange de donnée de façon confidentielle. Une personne A communique la clé publique à une personne B et garde la clé privée pour elle. La personne B envoie une clé symétrique cryptée à A avec le chiffrement RSA en utilisant la clé publique. La personne A décrypte la clé symétrique grâce à sa clé privée. Les deux personnes continuent leurs échanges avec les clés symétriques.

L'étape de création des clés est à la charge de A. Le renouvellement des clés n'intervient que si la clé privée est compromise, ou par précaution au bout d'un certain temps (qui peut se compter en années).

1. Choisir p et q , deux nombres premiers distincts
2. Calculer leur produit $n = p \times q$, appelé module de chiffrement
3. Calculer $\varphi(n) = (p - 1)(q - 1)$ (c'est la valeur de l'indicatrice d'Euler en n)
4. Choisir un entier naturel e premier avec $\varphi(n)$ et strictement inférieur à $\varphi(n)$, appelé exposant de chiffrement
5. Calculer l'entier naturel d , inverse de e modulo $\varphi(n)$, et strictement inférieur à $\varphi(n)$, appelé exposant de déchiffrement ; d peut se calculer efficacement par l'algorithme d'Euclide

Comme e est premier avec $\varphi(n)$, d'après le théorème de Bachet-Bézout il existe deux entiers d et k tels que $ed + k\varphi(n) = 1$, c'est-à-dire que $ed \equiv 1 \pmod{\varphi(n)}$: e est bien inversible modulo $\varphi(n)$.

Le couple (n, e) est la clé publique de chiffrement, alors que (n, d) est la clé privée.

Une valeur originale V est ainsi chiffrée en la valeur C par la formule : $C = V^e \pmod{n}$, et déchiffrée en la valeur V' par la formule : $V' = C^d \pmod{n}$.

Un exemple avec de petits nombres premiers (en pratique il faut de très grands nombres premiers afin que la factorisation de n soit trop longue à exécuter, et donc le code difficile à « cracker ») :

1. On choisit deux nombres premiers $p = 3$, $q = 11$
2. Leur produit $n = 3 \times 11 = 33$ est le module de chiffrement
3. $\varphi(n) = (3 - 1) \times (11 - 1) = 2 \times 10 = 20$
4. On choisit $e = 3$ (premier avec 20) comme exposant de chiffrement
5. L'exposant de déchiffrement est $d = 7$, l'inverse de 3 modulo 20 (en effet $ed = 3 \times 7 \equiv 1 \pmod{20}$)

La clé publique est $(n, e) = (33, 3)$, et la clé privée est $(n, d) = (33, 7)$.

Lorsque la personne B veut transmettre un message à A (par exemple une clé symétrique pour simplifier les échanges futurs), on a par exemple l'envoi du message représenté par la valeur 4 (ex. caractère e ou le décalage d'un code de César) :

- Chiffrement de la valeur 4 par B avec la clé publique : $4^3 \equiv 31 \pmod{33}$, la valeur envoyée à A par B est 31
- Déchiffrement de la valeur 31 par A avec la clé privée : $31^7 \equiv 4 \pmod{33}$, A retrouve bien la valeur originale 4

1. Fonctions préliminaires

Vous supposerez connu les deux fonctions suivantes.

```

pgcd (a,b)
"""retourne le pgcd de a et b"""

euclide (a,b)
"""retourne le triplet r, u, v tels que au+bv=r par l'algorithme d'Euclide"""

```

1.1. Ecrire une fonction **inverseModulaire** prenant en paramètres deux entiers e et m et retournant l'entier d compris entre 1 et m-1 tel que $ed \equiv 1 \pmod{m}$ lorsque e et m sont premiers entre eux et rien sinon.

1.2. Ecrire une fonction booléenne **estPremier** permettant de tester si un entier en paramètre est un nombre premier.

1.3. Ecrire une fonction **premierAleatoire** prenant en paramètres deux entiers inf et lg et choisissant aléatoirement des nombres dans $[inf, inf+lg]$ jusqu'à ce qu'elle trouve un nombre premier, qu'elle doit retourner. En pratique, il faut l'appliquer avec lg suffisamment grand pour qu'il existe un nombre premier dans l'intervalle.

1.4. Ecrire une fonction **premierAleatoireAvec** prenant en paramètre un entier n et tirant aléatoirement des nombres entiers compris entre 2 et n-1 jusqu'à ce qu'elle en trouve un premier avec n, qu'elle doit retourner. Rappel : vous avez la fonction **pgcd** à votre disposition.

1.5. Ecrire une fonction **expoModulaire** qui calcule a^n modulo m lorsque a, n et m sont passés en paramètres. Vous écrirez cette fonction de manière récursive et en vous assurant que le nombre de multiplications effectuées est de l'ordre de $\log_2 n$ (et non de n).

2. Codage et décodage RSA

2.1 Ecrire une fonction **choixCle** de paramètres deux nombres inf et lg, permettant de créer un triplet (p,q,e) tel que p et q sont deux nombres premiers distincts où $p \in [inf, inf + lg]$ et $q \in [p + 1, p + lg + 1]$, et e compris entre 2 et (p-1)(q-1) et e est premier avec (p-1)(q-1).

2.2 Ecrire une fonction **clePublique** permettant de calculer la clé publique (n,e) à partir de (p,q,e) et une fonction **clePrivee** permettant de calculer la clé privée (n,d) à partir de (p,q,e).

2.3 Ecrire une fonction **codageRSA** qui prend en paramètres un nombre $M < n$ et la clé publique (n,e) et qui permet de coder M.

2.4 Ecrire une fonction **decodageRSA** qui prend en paramètres un nombre $M < n$ et la clé privée (n,d) et qui permet de décoder M.