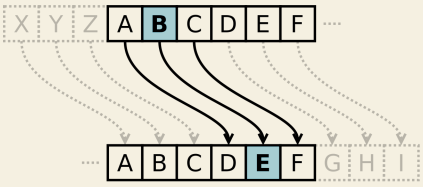
Chiffrement symétrique

Un peu d'histoire

La cryptographie symétrique, également dite à clé secrète, est la plus ancienne forme de chiffrement. Elle permet à la fois de chiffrer et de déchiffrer des messages à l'aide d'un même mot clé. On a des traces de son utilisation par les Égyptiens vers 2000 av. J.-C. Plus proche de nous, on peut citer le chiffre de Jules César, dont le ROT13 est une variante.

Cours

Le chiffre de César ou le code de César, est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes. Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet. Pour les dernières lettres (dans le cas d'un décalage à droite), on reprend au début.



Ce chiffrement est symétrique car il faut la même clé secrète pour chiffrer et déchiffrer un texte.

Exercice 1 :

1. Dans cette question, on considère un décalage de 7.
   - **a.** Chiffrer le message « **NUMERIQUE ET SCIENCES INFORMATIQUES** ».
   - **b.** Quel caractère possède l'occurrence maximale dans le message original ? dans le message chiffré ?
2. On considère le message « **WLUZLG HB NYHUK VYHS** ».
   - **a.** Quel est le caractère le plus présent dans ce message ? À quel caractère correspond-il ?
   - **b.** Déchiffrer le message.

Cours

Le chiffre de César est un mauvais chiffrement car :

- ❑ il n'existe que ____ clés différentes ;
- ❑ il ne résiste pas à la technique de ____ .

Cette dernière consiste à associer la lettre chiffrée la plus fréquente à une lettre très utilisée en français.

D.W.

Exercice 2 :

On donne ci-dessous le code unicode pour les caractères de `'A'` à `'Z'`.

Caractère	A	B	C	D	E	F	G	H	I	J	K	L	M
Code Unicode	65	66	67	68	69	70	71	72	73	74	75	76	77
Caractère	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Code Unicode	78	79	80	81	82	83	84	85	86	87	88	89	90

1. a. On propose le code suivant :

```
def chiffrement_cesar(c, k):
    code = ord(c) - ord('A') + k
    if code > 26:
        code -= 26
    if code < 0:
        code += 26
    return chr(ord('A') + code)
```

Que renvoie l'instruction `chiffrement_cesar(c, k)` dans les cas :

❑ `c, k = 'D', 4` ❑ `c, k = 'T', 13` ❑ `c, k = 'F', -10`

b. Proposer une version de `chiffrement_cesar` qui n'utilise qu'une seule instruction.

2. a. Proposer une fonction `chiffre_cesar(c:str, k:int) -> str` qui effectue un chiffrement de César de clé `k` sur chaque caractère de la chaîne `c`.

Par exemple, l'instruction `chiffre_cesar('NSI', 1)` devrait renvoyer `'OTJ'`.

b. Quelle instruction doit-on taper pour déchiffrer `'UZRADYMFUCGQ'` qui a subi un décalage de 12 ?

Exercice 3 :

On s'intéresse dans cet exercice à l'analyse fréquentielle d'un texte chiffré.

1. On considère le texte chiffré **YR YLPRR PBYOREG FR GEBHIR N GBHEPBVAT**.

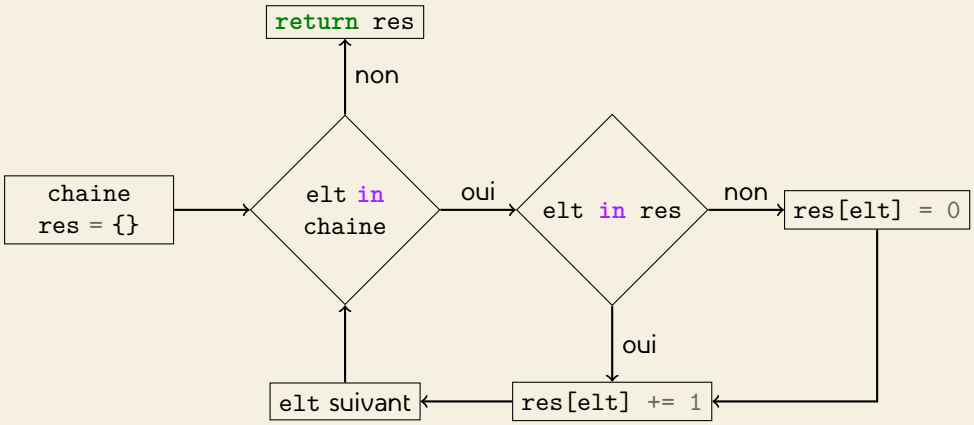
a. Compléter le tableau suivant :

Caractère	Y	R	L	P	B	O	E	G	F	H	I	N	V	A	T
Occurrence															

b. Quel est le caractère le plus fréquent dans ce message chiffré ?

c. Déchiffrer le message précédent.

Pour automatiser le calcul des occurrences, on peut utiliser un dictionnaire que l'on met à jour à chaque nouveau caractère en suivant l'algorigramme ci-dessous :



2. Que renvoie l'algorigramme précédent pour la chaîne `"LA RACLETTE EST PARTICULIEREMENT FONDANTE"` ?

On peut aussi automatiser le calcul des occurrences grâce à la méthode `count` des objets de type `str`. Par exemple, l'instruction `"INFORMATIQUE".count('I')` renvoie 2.

3. a. Compléter le code de la fonction `occurrences(chaine:str) -> list` qui renvoie la liste des occurrences d'une chaîne passé en paramètre. Par exemple, l'instruction `occurrences("DAVID")` renvoie `[('D', 2), ('A', 1), ('V', 1), ('I', 1)]`.

```
def occurrences(chaine:str) -> list:
    """ Occurrences d'une chaine. """
    res = []
    for elt in chaine:
        res.append((elt, ...))
    return res
```

b. Que renvoie `occurrences("DAVID")` ?

c. Proposer une correction du code précédent.

Pour déterminer la lettre la plus fréquente, il nous reste à déterminer le maximum d'une liste constituée de tuple de la forme `(str, int)`. On a besoin pour cela d'une fonction de comparaison.

4. a. Proposer une fonction `cmp_occ(el1:tuple, el2:tuple) -> int` qui renvoie :

- ❏ 1 si `el1` correspond à une occurrence moins élevée que `el2` ;
- ❏ -1 si `el1` correspond à une occurrence plus élevée que `el2` ;
- ❏ 0 sinon.

Par exemple, `cmp_occ(('D', 2), ('E', 7))` renvoie `1`.

b. Écrire alors une fonction `occ_max(L:list) -> str` qui renvoie le caractère le plus fréquent dans une liste d'occurrences. Par exemple, `occ_max([('D', 2), ('A', 1), ('V', 1), ('I', 1)])` renvoie le caractère `'D'`.

Cours

Le chiffre de Vigenère est une méthode de chiffrement symétrique dans lequel une même lettre du message clair peut, suivant sa position dans celui-ci, être remplacée par des lettres différentes. La clé utilisée est une suite de caractères qui peut, ou pas, avoir de sens.

Par exemple pour chiffrer le mot « **INFORMATIQUE** » avec la clé « **NSI** », on procède de la sorte :

I	N	F	O	R	M	A	T	I	Q	U	E
N	S	I	N	S	I	N	S	I	N	S	I

Le premier « **I** » devient ☐ par la transformation qui envoie ☐.

Le premier « **N** » devient ☐ par la transformation qui envoie ☐.

Le premier « **F** » devient ☐ par la transformation qui envoie ☐.

Exercice 4 :

Dans cet exercice, on pourra utiliser la table de Vigenère fournie.

1. Chiffrer la phrase « **THE FIGHT CLUB** » avec la clé « **CHAOS** ».
2. Déchiffrer le texte « **NO JNQUVB QG ZR EIESSGVS MN KQTARPQVE** » chiffré avec la clé « **CORNICHON** ».

Exercice 5 :

On possède le texte suivant :

HGYCN VIGYQ ZVJDJ CZVJP TWXTJ XJPIK TPXCI GXHJO RGXTN

EMGXN FIWCN UXGVW NRWQA GSCNV IGQGZ TXHJU XGX

qui a été chiffré par le chiffre de Vigenère avec une clé de longueur 2.

1. a. Donner les deux sous-messages qui ont été chiffrés avec le même caractère de la clé.

b. Quelle instruction python permet d'obtenir ces sous-messages simplement ?

2. a. Compléter les tableaux d'occurrences pour chacun de ces sous-messages.

Caractère	A	F	H	I	J	M	N	Q	R	S	T	V	W	X	Y	Z
Occurrence																

Caractère	C	D	E	G	H	I	K	N	O	P	Q	R	T	U	V	W	X
Occurrence																	

b. En déduire les clés probables.

3. Déchiffrer le message.

Cours

Pour casser le chiffre de Vigenère, il est possible d'utiliser **l'analyse fréquentielle** dans le cas où :

On découpe alors en sous-message et on étudie les occurrences sur chacun de ces sous-messages.

Dans le cas contraire, on utilise [ ] défini par :

$$IC = \frac{n_A \cdot (n_A - 1)}{n \cdot (n-1)} + \frac{n_B \cdot (n_B - 1)}{n \cdot (n-1)} + \dots + \frac{n_Z \cdot (n_Z - 1)}{n \cdot (n-1)}$$

où n est [ ] et n_A représente [ ], etc.

Dans un texte écrit en français, cet indice avoisine **0,0746**.
Dans un texte aléatoire, il se rapproche de **0,0385**.

Exercice 6 :

On possède un texte écrit en français dont le tableau d'occurrences est donné par :

Caractère	A	C	D	E	G	I	J	L	O	P	Q	R	S	T	U	V	Z
Occurrence	3	1	1	12	1	2	1	5	3	1	1	6	3	1	3	5	1

1. Calculer l'indice de coïncidence de ce texte.

2. On réalise un chiffrement de César de ce texte avec un décalage de 6.
Calculer l'indice de coïncidence du texte chiffré.

Exercice 7 :

Dans une console python, on teste l'instruction :

```
"".join([chr(randrange(65, 90+1)) for k in range(10000)])
```

1. Déterminer le rôle de cette instruction.

2. Grâce à une fonction d'occurrences sur ce message, on a obtenu :

```
[('A', 359), ('B', 400), ('C', 383), ('D', 362), ('E', 363),
 ('F', 392), ('G', 411), ('H', 400), ('I', 378), ('J', 395),
 ('K', 409), ('L', 372), ('M', 382), ('N', 383), ('O', 374),
 ('P', 366), ('Q', 429), ('R', 392), ('S', 375), ('T', 376),
 ('U', 391), ('V', 387), ('W', 362), ('X', 383), ('Y', 396), ('Z', 380)]
```

Calculer l'indice de coïncidence de ce message.

3. Proposer une fonction `coincidence(L:list) -> float` qui à une liste d'occurrences associe l'indice de coïncidence du texte associé à cette liste.

4. Proposer alors une méthode pour déterminer la longueur de la clé dans un chiffrement de Vigenère.

Cours

Pour chiffrer des données, on peut aussi travailler directement sur les bits à l'aide d'opérateurs binaires.
On rappelle ici, les tables de vérité de ces opérateurs :

ET	0	1
0		
1		

OU	0	1
0		
1		

XOR	0	1
0		
1		

On rappelle qu'en Python, on utilise les opérateurs `&`, `|` et `^` pour respectivement le **ET**, le **OU** et le **XOR**.
De façon générale, on utilise les opérateurs $\cdot$, $+$ et $\oplus$ pour respectivement le **ET**, le **OU** et le **XOR**.

Il existe également l'opérateur **NOT** qui inverse les bits. On le note $\overline{\cdots}$.
Par exemple, on souhaite calculer la valeur (en décimal) de l'expression

$$\Big((B3)_{16} + (11010001)_2\Big) \cdot \Big(143 \oplus (172)_8\Big)$$

❑ On commence par tout convertir en binaire :

$(B3)_{16} =$ $\qquad$ $143 =$ $\qquad$ $(172)_8 =$

❑ On réalise les opérations prioritaires :

$(B3)_{16} + (11010001)_2 =$ $\qquad$ $143 \oplus (172)_8 =$

❑ On en déduit le résultat :

$\qquad \cdot \qquad = \qquad =$

Attention : Il n'est pas toujours nécessaire de convertir en binaire (voir les prochaines interrogations).

Exercice 8 :

Calculer chacune des valeurs suivantes :

❑ $(11010010)_2 + (1101)_2$

❑ $134 \oplus 42$

❑ $\Big((167)_8 \cdot (431)_5\Big) \oplus (1100101)_2$

❑ $(4A2D)_{16} \cdot (10F2)_{16}$

❑ $\Big(155 \oplus 97\Big) \oplus 97$

❑ $\Big((61)_8 + (1101110)_2\Big) \cdot \Big((1F)_{16} \oplus 394\Big)$

Exercice 9 :

On s'intéresse à quelques propriétés de l'opérateur **XOR**.

1. Démontrer que **XOR** est associatif. C'est à dire que $(A \oplus B) \oplus C = A \oplus (B \oplus C)$
2. Démontrer que **XOR** est commutatif. C'est à dire que $A \oplus B = B \oplus A$
3. Démontrer que **XOR** est nilpotent d'ordre 2. C'est à dire que $A \oplus A = 0$
4. Démontrer que 0 est un élément neutre pour **XOR**. C'est à dire que $A \oplus 0 = A$

Exercice 10 :

On souhaite démontrer que $A \oplus B = (\overline{A} \cdot B) + (A \cdot \overline{B})$.

1. a. Compléter la table de vérité suivante :

A	B	$\overline{A}.B$	$A.\overline{B}$	$\overline{A}.B + A.\overline{B}$
0	0			
0	1			
1	0			
1	1			

b. En déduire que $A \oplus B = (\overline{A} \cdot B) + (A \cdot \overline{B})$.

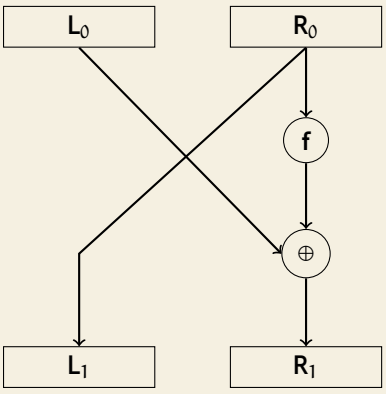
2. Démontrer de la même façon que $A \oplus B = (A + B) \cdot \overline{(A \cdot B)}$

Exercice 11 :

On considère un système de chiffrement simple qui repose sur **le schéma de Feistel** illustré par la figure ci-contre.

Ce système opère sur des blocs de 8 bits et se sert de la fonction **f** définie sur des mots de quatre bits par :

$$f(b_1b_2b_3b_4) = b_2b_3b_4b_1.$$



1. a. Exprimer L_1 et R_1 en fonction de L_0 et R_0.

b. Chiffrer le message « NSI » en utilisant la table ASCII.

2. a. En sortie de chiffrement, on a obtenu un mot de la forme « **L-R** ».
Démontrer que :

$$L_0 = R \oplus f(L) \text{ et que } R_0 = L$$

b. Déchiffrer le message **0x97E968FB**

Exercice 12 :

On considère un réseau de Feistel à deux tours sur des messages de 8 bits avec deux fonctions f_1 et f_2 définies sur des mots de 4 bits par les relations :

$$f_1(A) = A \oplus 1011 \text{ et } f_2(A) = A \cdot 0101$$

1. Réaliser un schéma de ce réseau de Feistel.

2. Donner l'équation de L_2 et R_2 en fonction de L_0 et R_0.

3. Chiffrer le message **« BAC »** en utilisant la table ASCII.

Exercice 13 :

On considère un réseau de Feistel à deux tours sur des messages de 24 bits avec deux fonctions **f** et **g** définies sur des mots de 12 bits par les relations :

$$\mathbf{f}(A) = A \oplus 0xC5A \qquad \text{et} \qquad \mathbf{g}(A) = \overline{A} \,.\, 0xB18$$

1. Réaliser un schéma de ce réseau de Feistel.
On notera « L_0-R_0 » pour le mot en entrée et « L_2-R_2 » pour le mot en sortie de réseau.

2. Donner l'équation de L_2 et R_2 en fonction de L_0 et R_0.

3. Chiffrer le message « **ΠSI** » en utilisant la table ASCII.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*