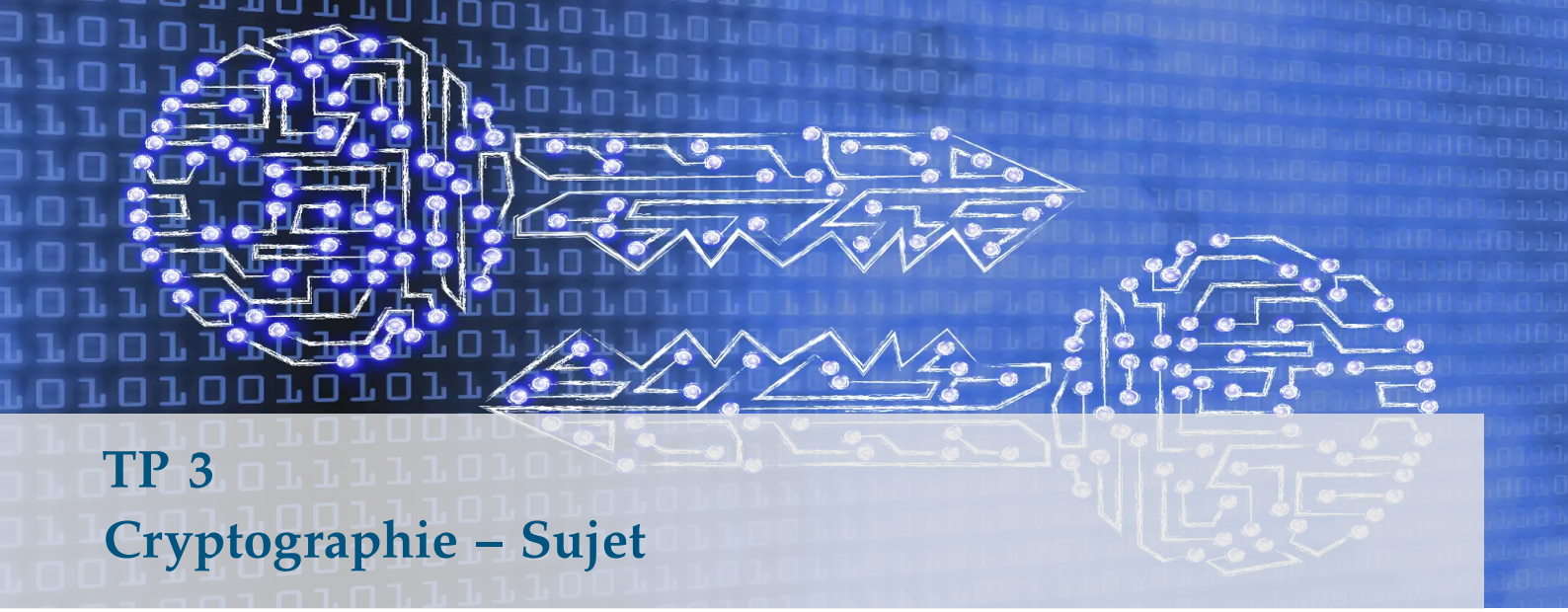

# TP 3
# Cryptographie – Sujet

La cryptographie a pour but de cacher le contenu d'un message. Afin de le rendre incompréhensible, on brouille le message suivant un protocole mis au point préalablement par l'expéditeur et le destinataire. Ce dernier n'aura qu'à inverser le procédé pour rendre le message lisible, alors que l'ennemi, s'il ne connaît pas le protocole de brouillage, trouvera difficile, voire impossible, de rétablir le texte original [1].

1: Ceux que le sujet intéresse pourront lire *Histoire des codes secrets* de Simon Singh.

Au premier siècle de notre ère est apparu un chiffrement par substitution, connu sous le nom de *code de César*, car l'empereur en a été l'un des plus assidus utilisateurs. Le chiffre de César consiste à assigner à chaque lettre de l'alphabet une autre lettre, résultant du décalage de l'alphabet d'un certain nombre de lettres. Par exemple, avec le décalage suivant :

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

le texte `'vous suivez le cours de python'` devient `'atzx xznaje qj htzwx ij udymts'`.
Pour connaître le code utilisé, il suffit de connaître la clé, c'est-à-dire la lettre qui correspond à la lettre `'a'` (dans l'exemple, la clé est donc `'f'`).

Le but de ce TP est de coder un message par cette méthode, et également de déchiffrer un message codé.

- Créez un dossier `TP03` puis, à l'intérieur, un fichier `TP03.py`. Récupérez le fichier `cesar.py` sur le site `https://ptsilamartin.github.io/info/TP.html` et enregistrez-le dans votre dossier `TP03`.
- Après avoir ouvert et parcouru `cesar.py`, copiez-collez son contenu au début de votre fichier `TP03.py`.

*Remarque* : On prendra bien soin, dans tout le TP, de documenter les fonctions écrites et de les tester.

## Faire correspondre les lettres et les entiers en python

On rappelle que c'est le type `string` (chaîne de caractères) qui permet, en python, de manipuler les caractères et les textes : par exemple `'a'` et `'bonjour, ça va ?'` sont de type `string`.

Nous allons utiliser deux fonctions prédéfinies en python : `ord` et `chr`. Elle permettent d'associer à chaque caractère un entier entre 0 et 255, et réciproquement. Voici un exemple :

```
>>> ord('a')
97
>>> chr(97)
'a'
```

**Question 1** Afficher les lettres dont les entiers associés sont compris entre 97 et 122.

Il serait plus pratique que l'entier associé à `'a'` soit 0 et celui associé à `'z'` soit 25.

**Question 2** Nous allons donc définir notre propre fonction `ordre(c:str) -> int`, qui prend pour argument une lettre `c`, et qui renvoie l'entier entre 0 et 25 lui correspondant. Vérifier que `ordre('a')` donne `0` et que `ordre('z')` donne `25`.

**Question 3** Écrire la fonction réciproque, c'est-à-dire la fonction `lettre(nb:int) -> str`, qui prend pour argument un entier `nb` entre 0 et 25, et qui renvoie la lettre correspondante. Par exemple, `lettre(2)` donne `'c'`.

## Codage et décodage d'un caractère connaissant la clé

On reprend le tableau de l'introduction, en indiquant les entiers associés aux caractères (leurs ordres) :

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| 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 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 0 | 1 | 2 | 3 | 4 |

On note $N$ l'ordre d'une lettre écrite en clair, $P$ l'ordre de cette même lettre après codage, et $K$ l'ordre de la clé. $N$, $P$ et $K$ sont donc des entiers entre 0 et 25.
Dans l'exemple ci-dessus, la clé est `'f'` donc $K$ vaut 5.

**Question 4** Exprimer $P$ en fonction de $N$ et $K$, puis $N$ en fonction de $P$ et $K$ (question sans ordinateur).

**Question 5** En déduire une fonction `crypte(cle:str, c:str) -> str` qui prend pour arguments deux chaînes de caractères : la clé `cle` et le caractère en clair `c`. Cette fonction devra renvoyer le caractère codé. Vérifier, par exemple, qu'en prenant comme clé le caractère `'f'`, le caractère en clair `'x'` donne le caractère codé `'c'`.

**Question 6** Ecrire la fonction réciproque c'est-à-dire la fonction `clair(cle:str, c:str) -> str` qui a pour arguments la clé `cle` et le caractère codé `c` et qui renvoie le caractère en clair.

## Codage et décodage d'un texte connaissant la clé

**Question 7** Écrire une fonction `codage(cle:str, texte:str) -> str`, qui a pour arguments la clé `cle` et un message en clair `texte`, et qui renvoie le message codé suivant la clé `cle`.

On supposera que, dans ce texte, il n'y a pas de caractère écrit en majuscule et qu'il n'y a pas de lettres accentuées. On laissera les éléments de ponctuation inchangés.

**Question 8** Tester la fonction sur un message de votre choix.

**Question 9** Écrire une fonction `decodage(cle:str, texte:str) -> str`, qui a pour arguments la clé `cle` et un message codé `texte`, et qui renvoie le message en clair.

**Question 10** Tester la fonction sur le message codé de la question précédente et vérifier que l'on récupère le message initial.

## Déterminer la clé : méthode itérative

On intercepte un message codé par le chiffrement de César (mais dont on ignore la clé). On veut déterminer la clé et le message original. Le codage de Cesar étant très basique (il n'y a que 26 clés possibles) il est très facile de décoder le message sans connaitre la clé.

**Question 11** A l'aide d'une boucle sur les 26 clés possibles déterminer la clé utilisée pour les `message1`, `message2`, `message3` présents au début de votre script.

## Pour aller + loin : Déterminer la clé : méthode automatique

Même si la méthode itérative est très rapide car le code est très simple, nous pouvons aussi proposer une méthode automatique. Nous allons exploiter l'idée que, dans une langue donnée, la fréquence d'apparition de chacune des lettres de l'alphabet n'est pas la même.

Après la récupération de `cesar.py`, au début de votre fichier, il y a en particulier une liste `fr` des fréquences d'apparition des lettres de l'alphabet en français.

*Remarque* : On notera bien que chaque fréquence est donnée en %, et qu'elle concerne les caractères alphabétiques uniquement. Par exemple, la lettre `'a'` apparaît, dans un texte comportant 100 caractères alphabétiques, en moyenne $8,4$ fois.

Nous allons « comparer » cette liste `fr` à la liste `fc` des fréquences obtenues à partir d'un texte donné.

**Question 12** Écrire une fonction `frequence`, qui a pour argument un message `texte`, et qui renvoie une liste de 26 éléments contenant la fréquence d'apparition de chacune des lettres de l'alphabet dans le message `texte`.

*Remarque* : `[0] * 26` est une liste de 26 éléments tous égaux à 0.

Pour trouver la clé, on va faire "tourner" le tableau `fc` des fréquences d'apparition des lettres du texte chiffré pour le faire coïncider le mieux possible avec le tableau `fr` des fréquences d'apparition de référence.

**Question 13** Écrire une fonction `distance` qui a pour argument un message `texte` et qui renvoie la liste `d` telle que :

$$\forall i \in \{0,\dots,25\}, d_i = \sum_{j=0}^{25} |fr_j - fc_{ijmod26}|$$

.

*Remarque* : Pour un entier $i$ donné, la quantité $|fr_j - fc_{ijmod26}|$ est l'écart entre la fréquence de la lettre d'ordre $j$ en français et la fréquence de cette même lettre dans le texte chiffré, en faisant l'hypothèse que la clé est la lettre d'ordre $i$. La quantité $d_i$ est la somme de tous ces écarts pour la clé d'ordre $i$ fixée. Donc, plus $d_i$ est petit, plus le tableau des fréquences avec la clé d'ordre $i$ est proche du tableau des fréquences de référence en français.

**Question 14** Écrire une fonction `minimum_distance` qui a pour argument un message `texte` et qui renvoie l'entier `i0` tel que $d_{i0} = \min\{d_i, i \in \{0,\dots,25\}\}$.

**Question 15** Écrire une fonction `decodage\_auto` qui a pour argument un message `texte` et qui renvoie le message en clair.

**Question 16** Vous testerez la fonction précédente sur les messages codés `message1`, `message2`, `message3` présents au début de votre script.

## Amélioration

Nous allons faire en sorte que la méthode de codage fonctionne sur des textes écrits avec des majuscules et des accents.

**Remarque**

- On supposera, dans la suite, que les majuscules ne sont pas accentuées.
- On n'utilisera pas de fonction prédéfinie en python pour répondre aux question suivantes.

**Question 17** Écrire une fonction `enleve_majuscules` d'argument une chaîne `texte` qui renvoie la même chaîne en transformant les majuscules en minuscules.

**Question 18** Écrire une fonction `enleve_accents` d'arguments une chaîne `texte` qui renvoie la même chaîne en enlevant les accents et en remplaçant les ç par des c.

**Question 19** Avec les fonctions suivantes, transformez `message4` pour ne plus avoir ni majuscules, ni accents. Proposez un codage du message obtenu avec la clé de votre choix puis effectuer un décodage automatique.