Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2023 – 2024
DEV1 – DEV1L

DEV1 – DEV1L – Laboratoires Python

TD 15 – Exercices récapitulatifs
Tout le cours

Ce TD est un TD récapitulatif sur la matière que vous avez vue lors des précédents TDs. Il ne se focalise pas sur un concept en particulier, mais vous demandera de mettre en œuvre tous les concepts que vous avez vu pour le résoudre. Comme il est plus long que d'habitude, il est prévu d'y allouer deux séances.

Table des matières

Rappel

Vous devez absolument avoir fait et compris les TDs précédents *avant* d'aborder celui-ci.

D'un point de vue contextuel, ce TD vous demande d'implémenter le chiffrement de Vigenère [1], un algorithme de chiffrement inventé au XV^{e} s., et qui a résisté aux cryptanalyses jusqu'au XVIII^{e} s. Intuitivement, cet algorithme de chiffrement est une généralisation du chiffrement de César [2].

Notez qu'il est « moins dirigé » que les précédents TDs : on s'attend à ce que vous réfléchissiez par vous-même, et effectuiez des recherches sur le web.

1 Le chiffrement de César

Le chiffrement de César est un algorithme permettant d'encoder l'information afin de la rendre illisible pour les personnes non autorisées. L'information lisible est qualifiée de *texte clair*, et celle illisible de *texte chiffré*.

Pour cela, on procède à l'aide d'une clé k (un entier), et on « décale » chaque lettre du texte en entrée de k caractères.

1. https://en.wikipedia.org/wiki/Vigenère_cipher - Dernier accès le 27 novembre 2023.
2. https://en.wikipedia.org/wiki/Caesar_cipher - Dernier accès le 27 novembre 2023.

Version du document
27 novembre 2023

Par exemple, avec une clé $k = 5$, on chiffre le texte « laboratoire de programmation » en « qfgtwfytnwj ij uwtlwfrrfynts » (chaque lettre est décalée de 5). La correspondance de chaque lettre est illustrée à la figure 1.

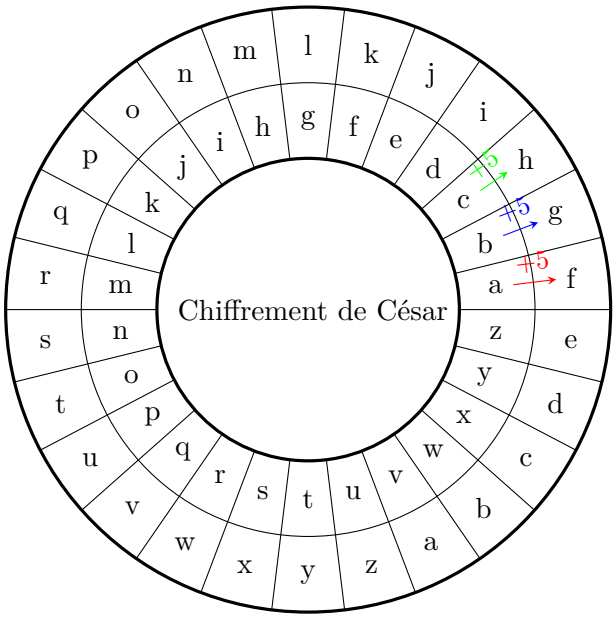

FIGURE 1 – Illustration du chiffrement de César avec une clé de 5

Si le texte chiffré est « glmjjviqirx hi giwev » et a été produit avec une clé $k = 4$, en décalant chaque lettre du texte chiffré de -4, on obtient le texte clair « chiffrement de cesar ».

Notez que dans les deux exemples ci-dessus, on a choisi d'ignorer les caractères qui ne sont pas des lettres.

1.1 Exercices

Dans les exercices suivants, vous écrirez vos *implémentations* dans un module `caesar.py`, et vous les *testerez* dans un module `caesar_main.py`.

Exercice 1 Chiffrement d'un caractère

Écrivez une fonction `encode_letter` qui prend deux paramètres `letter` (lettre minuscule) et `key` (entier), et qui retourne `letter` encodée en la décalant de `key`.

Par exemple, appeler `encode_letter('c', 3)` retourne `'f'`, car quand on décale `'c'` de 3 dans l'alphabet, on obtient `'f'`.

Pour écrire cette fonction, vous aurez besoin de l'opérateur `%` (modulo), ainsi que des fonctions

- ▷ `ord`, pour connaître la valeur entière associée à un caractère,
- ▷ `chr`, pour connaître le caractère associé à un entier.

Exercice 2 Test de `encode_letter`

Testez votre fonction `encode_letter` avec quelques caractères et quelques clés. Vous pouvez vérifier qu'elle est exacte grâce à des outils en ligne [3].

Exercice 3 Chiffrement d'une chaîne de caractères

3. https://cryptii.com/pipes/caesar-cipher - Dernier accès le 27 novembre 2023.

Écrivez une fonction `encode_str` qui prend deux paramètres `string` (chaîne de caractères) et `key` (entier) et qui retourne `string` encodée en décalant chacun de ses caractères de `key`. Vous devez laisser à l'identique les caractères qui ne sont pas des lettres.

Notez que sur les exemples illustrés en section 1, les caractères qui n'étaient pas des lettres ont été ignorés. De la même manière, on peut ne considérer que les lettres minuscules dans l'entrée. Ainsi, votre fonction `encode_str` doit se comporter comme tel :

- ▷ les caractères qui sont des lettres minuscules sont encodés « normalement »,
- ▷ les caractères qui sont des lettres majuscules sont d'abord convertis en minuscule et ensuite encodés,
- ▷ les caractères qui ne sont pas des lettres sont laissés en l'état.

Pour écrire cette fonction, vous devrez appeler la fonction `encode_letter` que vous avez écrite précédemment. Vous devrez également vous servir des fonctions `lower` et `isalpha` du module `string`.

Exercice 4 — Test de `encode_str`

Testez votre fonction `encode_str` avec quelques chaînes de caractères et quelques clés. Vous pouvez vérifier qu'elle est exacte grâce à des outils en ligne [4].

Pour l'exercice suivant, vous aurez besoin d'être capable

- ▷ d'ouvrir un fichier,
- ▷ de lire un fichier ligne par ligne,
- ▷ d'écrire des lignes dans un fichier.

Cette procédure est détaillée en annexe 3, page 6. Allez la lire !

Exercice 5 — Chiffrement d'un fichier

Écrivez une fonction `encode_file` qui prend trois paramètres :

- ▷ `infile`, un fichier d'entrée à ouvrir en lecture,
- ▷ `outfile`, un fichier de sortie à ouvrir en écriture,
- ▷ `key`, un entier.

Cette fonction lit `infile` ligne par ligne, et les écrit dans `outfile` en les chiffrant comme des chaînes de caractères avec `key`. Vous pouvez supposer qu'aucune erreur d'entrée / sortie ne se produira.

Pour écrire cette fonction, vous devrez appeler la fonction `encode_str` que vous avez écrite précédemment.

Exercice 6 — Test de `encode_file`

Testez votre fonction `encode_file` avec quelques fichiers et quelques clés. Vous pouvez vérifier qu'elle est exacte grâce à des outils en ligne [5].

4. https://cryptii.com/pipes/caesar-cipher - Dernier accès le 27 novembre 2023.
5. https://cryptii.com/pipes/caesar-cipher - Dernier accès le 27 novembre 2023.

2 Le chiffrement de Vigenère

Dans le chiffrement de César, on remarque que chaque caractère du texte en entrée est décalé du même nombre de caractères : la clé. L'idée principale du chiffrement de Vigenère est d'introduire un décalage variable (afin d'augmenter sa résistance à la cryptanalyse).

Pour cela, on procède à l'aide d'une clé k qui est un tuple d'entiers, par exemple $k = (3, 1, 9, 12)$. Les éléments de la clé sont des entiers allant de 1 à 25. S'il y a l éléments dans la clé, on dit que la clé est *de longueur* l. On note k_i le i$^{\text{e}}$ élément de la clé.

Pour chiffrer un texte en entrée, on décale un par un ses caractères en fonction des éléments de la clé. Par exemple, avec une clé $k = (3, 1, 9, 12)$, on décale

1. le 1$^{\text{er}}$ caractère du texte en entrée de 3,
2. le 2$^{\text{e}}$ caractère du texte en entrée de 1,
3. le 3$^{\text{e}}$ caractère du texte en entrée de 9,
4. le 4$^{\text{e}}$ caractère du texte en entrée de 12,
5. le 5$^{\text{e}}$ caractère du texte en entrée de 3,
6. le 6$^{\text{e}}$ caractère du texte en entrée de 1,
7. le 7$^{\text{e}}$ caractère du texte en entrée de 9,
8. le 8$^{\text{e}}$ caractère du texte en entrée de 12,
9. le 9$^{\text{e}}$ caractère du texte en entrée de 3, etc.

Ainsi, si l'on veut chiffrer le texte « Cours de programmation ! » avec la clé $(3, 1, 9, 12)$, on obtient « fpddv en buppddnvmwjxz ! ». La figure 2 illustre la correspondance de chaque caractère pour cette clé.

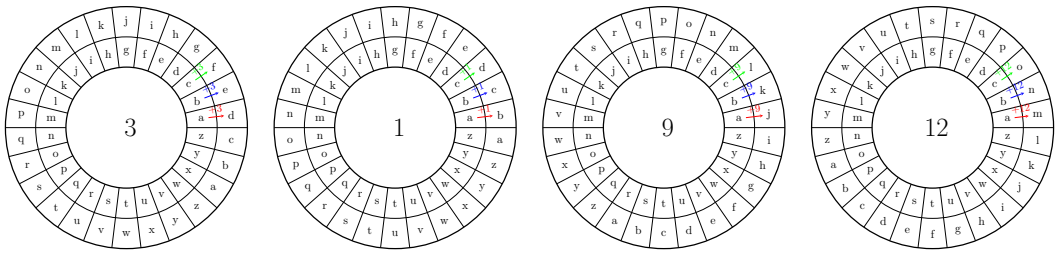

FIGURE 2 – Le chiffrement de Vigenère avec la clé $(3, 1, 9, 12)$

Notez que, comme dans le cas du chiffrement de César, les lettres majuscules ont d'abord été converties en minuscules avant d'être décalées (le « C »), et on a laissé à l'identique les caractères qui n'étaient pas des lettres (les espaces et le « ! »).

Par ailleurs, souvent, plutôt que de travailler avec des clés qui sont des tuples d'entiers, on travaille plutôt avec des clés qui sont des chaînes de caractères, plus faciles à retenir. Ainsi, la clé « CAIL » correspond à la clé $(3, 1, 9, 12)$, car

1. « C » est la 3$^{\text{e}}$ lettre de l'alphabet,
2. « A » est la 1$^{\text{ère}}$ lettre de l'alphabet,
3. « I » est la 9$^{\text{e}}$ lettre de l'alphabet,
4. « L » est la 12$^{\text{e}}$ lettre de l'alphabet.

2.1 Exercices

Dans les exercices suivants, vous écrirez vos *implémentations* dans un module `vigenere.py`, et vous les *testerez* dans un module `vigenere_main.py`.

Exercice 7 Validité de clés

Écrivez une fonction

- `valid_str_key` qui prend en paramètre une clé sous la forme d'une chaîne de caractères, et retourne vrai si elle représente une clé valide, c'est-à-dire si elle est de longueur au moins 2, et si les caractères de cette clé sont tous des lettres majuscules entre 'A' et 'Z',
- `valid_int_key` qui prend en paramètre une clé sous la forme d'un tuple d'entiers, et retourne vrai si elle représente une clé valide, c'est-à-dire si le tuple est de longueur au moins 2 et si les entiers de ce tuple sont dans l'intervalle $[-25, 25]$ et non tous nuls.

Exercice 8 Test de validité de clés

Testez vos deux fonctions de validité de clés avec quelques tuples d'entiers et quelques chaînes de caractères.

Par exemple, $(1, 2, 3, 4)$ et `HELLO` sont des clés valides, mais pas (1), $(1,)$, $(1, 2, 27)$, $(0, 0, 0)$, `HELLO!`, `hELLO`, `H`.

Exercice 9 Conversion de clés

Écrivez une fonction

- `str_to_int_key`, qui prend en paramètre une clé sous la forme d'une chaîne de caractères majuscules, et la retourne sous la forme d'un tuple d'entiers,
- `int_to_str_key`, qui prend en paramètre une clé sous la forme d'un tuple d'entiers, et la retourne sous la forme d'un mot.

Ces fonctions retournent `None` si on leur fournit des clés invalides.

Exercice 10 Test de conversion de clés

Testez vos deux fonctions de conversion de clés avec quelques tuples d'entiers et quelques chaînes de caractères.

Exercice 11 Chiffrement d'une chaîne de caractères

Écrivez une fonction `encode_str` qui prend deux paramètres `string` (chaîne de caractères) et `key` (une clé sous la forme d'un tuple d'entiers) et qui retourne `string` encodée. Vous devez laisser à l'identique les caractères qui ne sont pas des lettres. Cette fonction retourne `None` si on lui fournit une clé invalide.

Notez que sur les exemples illustrés en section 2, les caractères qui n'étaient pas des lettres ont été ignorés. De la même manière, on peut ne considérer que les lettres minuscules dans l'entrée. Ainsi, votre fonction `encode_str` doit se comporter comme tel :

- les caractères qui sont des lettres minuscules sont encodés « normalement »,
- les caractères qui sont des lettres majuscules sont d'abord convertis en minuscules et ensuite encodés,
- les caractères qui ne sont pas des lettres sont laissés en l'état.

Pour écrire cette fonction, vous devrez appeler la fonction `encode_letter` que vous avez écrite précédemment dans le module `caesar.py`.

Exercice 12 Test de `encode_str`

Testez votre fonction `encode_str` avec quelques chaînes de caractères et quelques clés. Vous pouvez vérifier qu'elle est exacte grâce à des outils en ligne [6].

Exercice 13 Chiffrement d'un fichier

Écrivez une fonction `encode_file` qui prend trois paramètres :

- ▷ `infile`, un fichier d'entrée à ouvrir en lecture,
- ▷ `outfile`, un fichier de sortie à ouvrir en écriture,
- ▷ `key`, une clé sous la forme d'un tuple d'entiers.

Cette fonction lit `infile` ligne par ligne, et les écrit dans `outfile` en les chiffrant comme des chaînes de caractères avec `key`. Cette fonction retourne `None` si on lui fournit une clé invalide. Vous pouvez supposer qu'aucune erreur d'entrée / sortie ne se produira.

Pour écrire cette fonction, vous devrez appeler la fonction `encode_str` que vous avez écrite précédemment.

Faites attention : quand vous avez fini d'encoder une ligne, vous n'avez peut-être pas « épuisé » toute la clé, il faut donc vous souvenir où vous vous étiez arrêtés. Par exemple, si vous terminez d'encoder une ligne avec l'entier 4 de la clé $(3, -1, 4, 8, 0)$, l'encodage de la ligne suivante commence avec le 8.

Exercice 14 Test de `encode_file`

Testez votre fonction `encode_file` avec quelques fichiers et quelques clés. Vous pouvez vérifier qu'elle est exacte grâce à des outils en ligne [7].

3 Annexe - Lecture de fichiers en Python

Pour lire ou écrire dans un fichier, il est nécessaire de l'*ouvrir* préalablement. Tout fichier qui est ouvert doit ultimement être *fermé*. Si l'on souhaite lire dans un fichier, on dit qu'on l'*ouvre en lecture*, et si l'on souhaite y écrire, on dit qu'on l'*ouvre en écriture*. Quand on ouvre, que l'on lit ou que l'on écrit dans un fichier, des erreurs peuvent se produire (comme en cas d'un disque défectueux, d'un câble USB arraché, etc.). On parle d'*erreur d'entrée / sortie*.

En Python, on peut ouvrir un fichier texte en lecture [8] grâce à l'appel `open`. On peut ainsi simplement en imprimer le nom comme illustré au code 1. On ferme un fichier grâce à `close`.

Si on souhaite ouvrir un fichier en écriture, on ajoute simplement un paramètre `"w"`, et cet appel devient ainsi `f = open("test.txt", "w")`. Notez que l'on peut ouvrir plusieurs fichiers au sein d'un même `with`, par exemple en faisant

6. https://cryptii.com/pipes/vigenere-cipher - Dernier accès le 27 novembre 2023. Attention : ce lien considère que la lettre « A » d'une clé correspond à un décalage de zéro ! Ce TD considère au contraire que c'est un décalage de 1.
7. https://cryptii.com/pipes/vigenere-cipher - Dernier accès le 27 novembre 2023. Attention : ce lien considère que la lettre « A » d'une clé correspond à un décalage de zéro ! Ce TD considère au contraire que c'est un décalage de 1.
8. Pour que cela fonctionne, il faut que le fichier existe.

```
f = open("test.txt")
print(f.name)
f.close()
```

Code 1: Ouvrir un fichier, imprimer son nom, et le fermer

`with open(inputfile) as inf, open(outputfile, "w") as outf:`

On préférera toujours ouvrir un fichier grâce au mot-clé `with`, qui va se comporter comme un « bloc » et permettra d'éviter de mettre des objets dans des états incohérents en cas d'erreur d'entrée / sortie. Un fichier ouvert de cette manière sera également toujours fermé quand on sortira du bloc. On procède ainsi comme illustré au code 2.

```
with open("test.txt") as f:
    print(f.name)
```

Code 2: Ouvrir un fichier et imprimer son nom

Une fois un fichier ouvert en lecture, on peut

- ▷ lire son contenu caractère par caractère avec `read`,
- ▷ lire son contenu ligne par ligne avec une boucle `for`,

tel qu'illustré au code 3. Souvent, lire un fichier ligne par ligne est suffisant.

```
# read a file char by char
with open("test.txt") as f:
    while c := f.read(1):
        print(c)

# read a file line by line
with open("test.txt") as f:
    nbr = 1
    for line in f:
        print(nbr, ":", line)
        nbr += 1
```

Code 3: Lire un fichier caractère par caractère, et ligne par ligne

De la même manière, une fois que l'on a ouvert un fichier en écriture, on peut y écrire avec `write`, tel qu'illustré au code 4. Ce code

1. ouvre le fichier « test.txt » en lecture,
2. ouvre le fichier « brol.txt » en écriture,
3. lit le contenu de « test.txt » ligne par ligne,
4. écrit chacune des lignes lues dans « brol.txt ».

En d'autres termes, il copie le contenu du fichier « test.txt » dans « brol.txt » (et écrase ce qui s'y trouvait).

```
with open("test.txt") as inf, open("brol.txt", "w") as outf:
    for line in inf:
        outf.write(line)
```

Code 4: Ouvrir un fichier et imprimer son nom