Devoir Maison - Musique Décalée

Consignes

▷ *Rendu*

Le devoir est à faire de manière électronique. Si vous souhaitez le rendre sur papier pour une raison quelconque, merci de m'adresser un mail à itc-eiffel@sjaziri.fr.

Le rendu consistera en un fichier Python *.py* à envoyer par mail à l'adresse : itc-eiffel@sjaziri.fr, avant le **dimanche 20 novembre 2022 à 21 :12**.

Vous devez télécharger l'archive `.zip` disponible à l'adresse : `https://itc-eiffel.sjaziri.fr/sup/devoirs/dm1/dm1.zip` et l'extraire dans le dossier de votre choix. Vous y trouverez :

- `dm.py` : le fichier à remplir avec vos fonctions. Il contient déjà deux lignes.
  ```
  import numpy as np
  from scipy.io import wavfile
  ```
  Il s'agit de *l'importation de librairies*. Ces librairies mettent à disposition des fonctions utiles, dans notre cas, pour la génération de fichier WAV.
- `songs` : un dossier qui contient des partitions pour vos tests
- `messages` : un dossier qui contient des messages à chiffrer ou déchiffrer pour vos tests
- `dm1_sujet.pdf` : ce fichier.

Pour extraire le fichier des tutoriels sont mis à disposition dans la section *Devoirs* de la page d'accueil du site web `https://itc-eiffel.sjaziri.fr`. Si vous avez des problèmes les fichiers sont rendus accessibles ici : `https://itc-eiffel.sjaziri.fr/sup/devoirs/dm1/dm1_fichiers.html`. **Attention** à créer les dossiers *songs* et *messages* dans ce cas.

Vous utiliserez pour faire le devoir et écrire dans `dm.py` le logiciel Spyder. Une documentation est disponible à l'adresse `https://itc-eiffel.sjaziri.fr/doc/spyder_tuto/tuto.html`. **Merci de la lire intégralement**.

Si vous avez un problème avec l'outil, tournez vous rapidement vers un camarade ou vers un professeur par mail. Exceptionnellement seront acceptés les fichiers txt, word, libreoffice, Google Doc et, au pire des cas, des captures d'écran de votre travail ou des copier-coller par mail. Il faudra toutefois prendre en main Spyder rapidement.

Pour toute question : itc-eiffel@sjaziri.fr.

▷ *Règles de travail*

J'attends de vous que vous consacriez à ce devoir quelques heures de vos vacances et au minimum 2 h la semaine où vous n'avez pas informatique à la rentrée.

Le travail en groupe et l'entre aide sont encouragés. Le transfert de code est interdit. Tous codes identiques à renommage et permutations d'instructions près seront sanctionnés par un 0 à la note finale. Pour éviter que cela arrive la règle à suivre est simple :

- Ne pas copier-coller le code d'un camarade.
- Si un camarade vous explique la solution à partir de son propre code, prendre le temps de digérer l'explication et réécrire plus tard le code de la réponse.

▷ *Code Python*

Les consignes suivantes **doivent** être respectées sous peine d'être pénalisé par des **points négatifs**.

Le code doit être clair. - *1 point par fonction.*
- Évitez les lignes à rallonge. S'il faut introduire une variable intermédiaire, faîte le.
- Nommez vos variables avec du sens. Évitez toutefois des noms trop long. Auquel cas décrivez le rôle de la variable en commentaire.
- Ne surchargez pas de commentaires. Les commentaires utiles sont :
  - Spécification de la fonction
  - Description du rôle d'une variable
  - Description du rôle d'une boucle complexe.

Respectez la syntaxe Python. - *0 à l'exercice si syntaxe inventée.*
- Gardez la fiche de syntaxe à côté de vous lorsque vous codez
- Vérifiez que ce que vous écrivez respecte la syntaxe.
- Vérifiez que votre code s'exécute avant de l'envoyer.
- Vous pouvez envoyer du code qui ne s'exécute pas :
  - Ajoutez un commentaire avant la fonction
  - Les petites erreurs n'implique pas 0 (oublie de :, = au lieu de ==,...)
  - Si vous avez une idée, mais que vous n'arrivez pas à l'écrire, n'inventez pas. Écrivez votre idée en commentaire.
  - Par exemple :
    ```
    # Cette fonction n'est pas correcte
    def maximum(l) :
        m = 0
        # Regarder chaque élément x de l
        if m < x :
            m = x
        # Quand c'est fini
        return m
    ```

Pour nommer une fonction utilisez le nom donné dans l'énoncé - *0 à l'exercice sinon*

Utilisation de `print` ou de `input` interdite pour le moment - *0 à l'exercice sinon*

1 Mélodie MIDI

L'objectif de ce problème est de générer un fichier de type WAV à partir d'une partition écrite.

1.1 Introduction

Le son est une vibration mécanique d'un fluide qui se propage sous forme d'ondes. On modélisera dans ce problème l'onde par une sinusoïde. Une note de musique correspond à une onde d'une certaine fréquence. Par exemple le `la`^{3}, note du diapason, correspond à la fréquence 440 Hz. On l'appelle d'ailleurs souvent le `la 440`.

La sinusoïde de fréquence f et d'amplitude A est la fonction

$$\begin{array}{rccl} h: & \mathbb{R} & \to & \mathbb{R} \\ & t & \mapsto & A\sin(2\pi f t) \end{array}$$

Un haut-parleur permet de produire du son à partir d'un signal électrique. Le son en lui-même produit grâce aux vibrations d'une membrane. Cette membrane est reliée à un aimant qui est lui-même attiré et repoussé par une bobine en fonction du courant électrique qui la traverse.

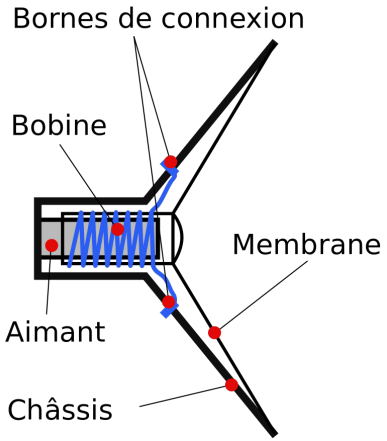


FIGURE 1 – Schéma d'un Haut-Parleur

Le son analogique (signal électrique) correspondant à une note de musique est produit par la carte son de l'ordinateur. Cette dernière est capable de traduire un son numérique (données binaires) en son analogique.

Le son numérique consiste en un *échantillonnage* du signal analogique. Le signal analogique est un signal continu, un échantillonnage consiste à extraire un nombre fini de valeur du signal.

Lorsque les valeurs sont prises à intervalles réguliers, le nombre de valeurs extraites en une seconde est appelée *fréquence d'échantillonnage*. On définit aussi *l'amplitude* de l'échantillonnage comme étant l'amplitude de la fonction sinusoïde.

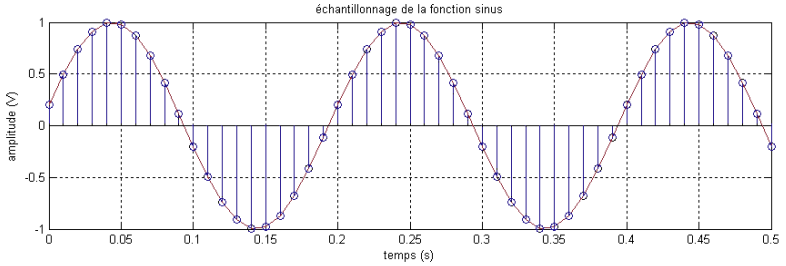


FIGURE 2 – Échantillonnage de la fonction sinus

Un fichier WAV est un fichier qui contient certaines informations sur la nature du son à produire par la carte son et qui contient un échantillonnage du signal sonore à produire.

Le but du problème est donc de produire un échantillonnage du signal sonore correspondant à une partition écrite sous format numérique textuel.

Dans un premier temps les notes seront représentée par le numéro de la touche qui la produit sur un piano à 88 touches. Par exemple le `la 440` est produit par la touche 49 du piano.

Si n est le numéro de la touche de piano qui produit une note, la fréquence de cette note s'écrit :

$$2^{\frac{n-49}{12}} \cdot 440$$

Question 1.1. *Fréquence d'une touche de piano*

Implémentez une fonction `freq_touche` qui prend en entrée un entier n et renvoie la fréquence en Hz de la touche n du piano si $0 < n < 89$, 0 sinon.

1.2 Outils numériques

Dans cette section, on commencera par implémenter des fonctions permettant de faire des calculs numériques utiles pour l'échantillonnage.

Question 1.2. *Valeur absolue*

Implémentez une fonction `abs` qui prend en entrée un nombre flottant et renvoie sa valeur absolue.

Question 1.3. *Approximation de pi*

On définit la suite réelle suivante qui tend vers π :

$$P_n = 12^{\frac{1}{2}} \sum_{i=0}^{n} \frac{(-3)^{-i}}{2i+1}$$

Implémentez une fonction `approx_pi` qui prend en entrée un entier n et qui renvoie P_n. Vérifiez que `approx_pi` renvoie la même valeur pour 40, 100, 1000 et 100.000. Comparez les décimales obtenues à celle de π.

Question 1.4. *Première approximation de* $\sin(x)$

Pour tout $x \in \mathbb{R}$, on définit la suite réelle suivante qui tend vers $\sin(x)$:

$$s_n^x = \begin{cases} x & \text{si } n = 0 \\ -s_{n-1}^x \cdot \frac{x^2}{2n(2n+1)} & \text{sinon} \end{cases}$$

$$S_n^x = \sum_{i=0}^{n} s_i^x$$

Implémentez une fonction `approx_sin` qui prend en entrée un flottant x et un entier n et qui renvoie S_n^x.

Question 1.5. *Etude de la convergence de* ***approx_sin***

On propose la fonction suivante :

```
def conv(n,erreur,nmax) :
    """
    Entrées : n en entier, erreur un flottant, nmax un entier positif
    Sortie : ???
    """
    pi = approx_pi(40)
    i = 0
    while abs(approx_sin(2 * n * pi,i)) > erreur and i < nmax:
        i = i + 1
    return i
```

On remarquera que la fonction `conv` utilise le résultat des fonctions `approx_pi` et `approx_sin`. Lorsqu'une fonction est appelée dans une instruction, la fonction est exécutée sur ses entrées puis sa sortie est utilisée dans l'instruction en lieu et place de l'appel. Par exemple :

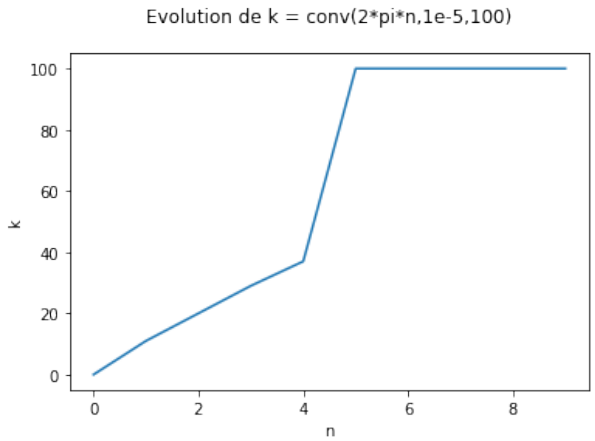
```
pi = approx_pi(40)
```

affecte à la variable `pi` la sortie de la fonction `approx_pi` exécutée sur l'entrée 40.

```
abs(approx_sin(2 * n * pi,i)) > erreur
```

récupère la sortie de la fonction `approx_sin` appelée sur les entrées `2 * n * pi` et `i`, puis cette sortie est utilisée comme entrée de la fonction `abs`, et enfin la sortie de `abs` est comparée à `erreur`. La condition est donc vrais si $|S_i^{2\pi n}| >$ `erreur`.

Donnez la spécification de la sortie de la fonction. Puis analysez le graphe ci-dessous et expliquez pourquoi, la fonction `approx_sin` ne peut pas être utilisée telle quelle sur de grandes valeurs en pratique.



Question 1.6. *Amélioration de l'approximation de* $\sin(x)$

Implémentez dans un premier temps une fonction `modulo_2pi` qui prend en entrée un flottant x et qui renvoie sa valeur modulo 2π. On rappelle que x modulo 2π est l'unique flottant $y \in]-\pi, \pi]$ tel que $\exists n \in \mathbb{N}, |x - y| = 2\pi n$.

On pourra définir dans la fonction la variable `pi` comme ceci :

- si la question 2 a été réussie et que `approx_pi` (40) est égal à `3.141592653589793` à 10^{-14} près :
  `pi = approx_pi(40)`
- sinon
  `pi = np.pi`

Vous pouvez aussi utiliser la fonction `abs` pour vous restreindre au cas d'un angle positif.

Vous modifierez ensuite la fonction `approx_sin` pour ramener l'angle donné en entrée entre $]-\pi, \pi[$ et ainsi assurer une convergence rapide de l'approximation. Vous utiliserez pour cela la fonction `modulo_2pi`.

1.3 Échantillonnage

Une partition est donnée dans un premier temps comme étant une liste de couple `(n,d)`, avec `n` le numéro de la touche de piano produisant la note et `d` la durée de la note en secondes.

L'échantillonnage d'une partition est obtenu en concaténant l'échantillonnage de chaque couple. L'échantillonnage d'un couple `(n,d)` correspond à l'échantillonnage de la sinusoïde modélisant l'onde sonore produite par l'appuie de la touche `n` pendant `d` secondes. Les valeurs de l'échantillonnage sont donc prises de manière régulière dans l'intervalle $[0, \texttt{d}]$.

Par exemple voici un échantillonnage du `la 440` sur une durée de `2 sec` avec comme fréquence d'échantillonnage `5 Hz` et comme amplitude 2.

```
[ 0.0 , -1.9696... , -0.6840... , 1.7320... , 1.2855... , -1.2855... ,
```

```
-1.7320... , 0.6840... , 1.9696... , -6.7165...e-12 ]
```

Question 1.7. *Discrétisation d'un intervalle*

Implémentez une fonction `discretisation` qui prend en entrée deux flottants $a < b$ et un entier $n \geq 2$ et qui renvoie une *liste* de n points (flottants) de l'intervalle $[a, b]$ à écarts réguliers.

Formellement la sortie est une liste $l = [i_1, i_2, \ldots, i_n]$ tels que :

1. $i_1 = a$
2. $i_n = b$
3. $\exists \alpha \in \mathbb{R}^{+*}, \forall 1 \leq k < n,\ i_{k+1} - i_k = \alpha$

Indication : Déterminez α et pour tout $1 \leq k < n$ la valeur de i_k.

Question 1.8. *Échantillonnage d'une note*

Implémentez une fonction `echantillonnage_note` qui prend en argument la fréquence de la note, sa durée, la fréquence d'échantillonnage et l'amplitude et qui renvoie l'échantillonnage de la note.

Pour se faire on commencera par définir la liste des points sur lesquels sera fait l'échantillonnage :

```
t = discretisation(a,b,n)
```

avec `a`, `b` et `n` des valeurs à définir vous-même.

On pourra définir dans la fonction la variable `pi` comme ceci :
- si la question 2 a été réussie et que `approx_pi` (40) est égal à `3.141592653589793` à 10^{-14} près :
  `pi = approx_pi(40)`
- sinon
  `pi = np.pi`

Pour calculer le sinus d'une valeur flottante `x` on pourra utiliser :
- si la question 5 a été réussi et que `approx_sin(2*pi,100)` est égal à 0 à 10^{-14} près.
  `approx_sin(x,100)`
- sinon
  `np.sin(x)`

Question 1.9. *Échantillonnage d'une partition*

Implémentez une fonction `echantillonnage_partition` qui prend en argument une partition, la fréquence d'échantillonnage et l'amplitude et qui renvoie l'échantillonnage de la partition.

Vous devrez utiliser la fonction `echantillonnage_note` implémentée à la question précédente pour récupérer l'échantillonnage d'une note. Vous pourrez utiliser cette fonction même si vous n'avez pas réussi à l'implémenter.

Question 1.10. *Génération du fichier WAV*

Cette question n'est pas notée.

On peut maintenant utiliser l'échantillonnage que vous avez produit pour écrire un fichier WAV. Dans la console de Spyder ou à la fin de votre fichier vous pouvez ajouter le code suivant pour générer un fichier WAV dans le même dossier que votre fichier Python.

```
freq_e = 44100 # Fréquence d'échantillonnage
amplitude = 2048 # Amplitude de l'échantillonnage
# Changez les notes et durées ci-dessous pour générer d'autres mélodies...
# Do Ré Mi Fa Sol La Si - 2sec chaque
partition = [(40,2),(42,2),(44,2),(45,2),(47,2),(49,2),(51,2)]
e = echantillonnage_partition(partition,freq_e,amplitude)
wavfile.write('octave4_2s.wav', rate=freq_e,
                        data=np.array(e).astype(np.int16) )
```

Vous pouvez m'envoyez par mail vos plus belles mélodie. Elle seront mise à l'écoute sur le site web, de manière anonyme ou non selon votre préférence.

Question 1.11. *Échantillonnage d'un accord - Bonus*

Cette question est notée en bonus.

Un accord de piano consiste à appuyer sur plusieurs touches en même temps. L'onde sonore produite est la superposition des ondes sonores de chaque touche. Elle est modélisée par la somme des sinusoïdes de chaque note. Par exemple pour l'accord de fa^3 on appuie sur les touches `fa` (45), `la` (49) et `do` (52). On note $f_{\texttt{fa}}$, $f_{\texttt{la}}$ et $f_{\texttt{do}}$ la fréquence de ces trois notes. L'onde sonore de l'accord est modélisée par la fonction :

$$\begin{array}{rccl} h: & \mathbb{R} & \to & \mathbb{R} \\ & t & \mapsto & A\left(\sin(2\pi f_{\texttt{fa}} t) + \sin(2\pi f_{\texttt{la}} t) + \sin(2\pi f_{\texttt{do}} t)\right) \end{array}$$

Implémentez une fonction `echantillonnage_accord` qui prend en entrée une *liste de fréquences*, une durée, une fréquence d'échantillonnage et une amplitude et renvoie l'échantillonnage de l'accord dont les fréquences sont celles de la liste.

Une partition d'accord est une liste de couples `(l,d)` ou `l` est une liste `[n_0, n_1, \dots, n_k]` de numéro de touche de piano représentant un accord de piano et `d` est la durée de l'accord.

Implémentez ensuite une fonction `echantillonnage_partition_accords` qui prend en entrée une partition, une fréquence d'échantillonnage et une amplitude et qui renvoie l'échantillonnage de la partition.

Vous pouvez générer un fichier WAV à l'aide du code ci-dessous

```
freq_e = 44100
amplitude = 2048
# Vous pouvez changer la partition ci-dessous
# Accord de Fa, La, Accord de Re
partition = [([49,52,45],2), ([49],1), ([42,46,49],2)]
e = echantillonnage_partition_accords(partition,freq_e,amplitude)
wavfile.write('accords.wav', rate=freq_e,
                        data=np.array(e).astype(np.int16) )
```

1.4 Conversion d'une partition

On veut maintenant écrire la partition, non plus sous forme de liste de couple, mais sous la forme d'un texte où les notes sont écrite avec la notation grégorienne. Les notes de la gamme tempérée sont notées dans l'ordre `C, C#, D, D#, E, F, F#, G, G#, A, A#, B` qui correspond donc à `Do, Do#, Ré, Ré#, Mi, Fa, Fa#, Sol, Sol#, La, La#, Si`. Pour simplifier la lecture de la partition, on modifie la notation des demi-tons : les dièses sont notés par la version minuscule de la note. Ainsi `Do#` qui se note en grégorien `C#` sera noté `c`.

On apposera derrière le caractère décrivant la note le numéro de l'octave dans laquelle elle se trouve. Par exemple le `La#` de l'octave `2` sera noté `a2`. Les octaves du piano à 88 touches vont de 0 à 7. La touche numéro 1 est la note `A0` et la touche 88 est la note `C8`. Les touches produisent ensuite les notes de chaque octave dans l'ordre de la gamme et par octaves croissante :

`A0, a0, B1, C1, c1, ..., A6, a6, B6, C7`

Pour plus de détails vous pouvez lire la page Wikipédia sur les touches de pianos.

Question 1.12. *Touche correspondant à une note dans l'octave*

En faisant une recherche linéaire dans la chaîne de caractère `"CcDdEFfGgAaB"` implémentez une fonction `touche_octave` qui prend en entrée un caractère et renvoie son numéro entre `0` et `11` dans l'octave, *i.e* dans la chaîne listant les notes de la gamme tempérée. Si le caractère n'est pas une note de la gamme tempérée, la fonction doit renvoyer `-1`.

Question 1.13. *Touche correspondant à une note dans l'octave*

Soit `s` un caractère et `k` un numéro d'octave. On note `t = touche_octave(s)`. La touche correspondant à la note `s` de l'octave `k` est

$$n = \begin{cases} -1 & \text{si } t = -1 \\ t - 8 + k * 12 & \text{sinon} \end{cases}$$

Vérifiez que cette formule est valide pour toute note correspondant à une touche de piano et renvoie un nombre plus petit strictement que 1 ou plus grand strictement que 88 pour toute note invalide.

Implémentez une fonction `touche_piano` qui prend en entrée un caractère `s` et un numéro d'octave `k` et renvoie le numéro de la note `s` dans l'octave `k` entre `1` et `88` si c'est une note valide, un nombre strictement plus petit que 1 ou strictement plus grand que 89 sinon.

Question 1.14. *Lecture d'une partition*

Pour simplifier la lecture, on considérera qu'une note a une durée multiple de $\frac{1}{8}$ de seconde, entre 0 et 8. Une note s'écrira alors toujours comme une chaîne de trois caractères, note, octave, durée. Par exemple : `"G62"` représente le `Sol` de l'octave `6` pour une durée de $\frac{1}{4}$ secondes, `"d41"` représente le `Do#` de l'octave `4` pour une durée de $\frac{1}{8}$ secondes.

Une partition est la concaténation des chaînes de caractère représentant chaque notes de la partition : `"C41D41E41F41G41A41B41C51B41A41G41F41E41D41C41"`.

Implémentez une fonction `partition` qui prend en entrée une chaîne de caractère représentant une partition et qui renvoie une partition dans le format utilisé par la fonction `echantillonnage_partition`, c'est-à-dire une liste de couple `(n,d)` avec `n` le numéro de la touche de piano correspondant à la note et `d` la durée de la note.

On rappelle qu'une chaîne de caractère `s` qui représente un nombre peut être convertie en ce nombre en utilisant la fonction `int` : `int("12")` donne l'entier =12.

Vous pouvez alors générer un fichier WAV à partir d'une partition sous forme de chaîne de caractère avec le code suivant :

```
freq_e = 44100
amplitude = 2048
p = partition("C41D41E41F41G41A41B41C51B41A41G41F41E41D41C41") # partition
e = echantillonnage_partition(p,freq_e,amplitude)
wavfile.write('octave4_asc_desc_1s.wav', rate=freq_e,
                        data=np.array(e).astype(np.int16) )
```

Question 1.15. *Conversion fichier vers WAV - Bonus*

Cette question est notée en bonus.

On veut maintenant générer le fichier WAV en lisant la partition dans un fichier de type TXT. La fonction `open` permet d'ouvrir un fichier :

```
f = open(chemin,mode)
```

`chemin` décrit l'emplacement de votre fichier sur l'ordinateur et `mode` permet de préciser le mode d'ouverture de votre fichier (lecture, écriture, ajout, ...).

Ainsi pour ouvrir un fichier `partition.txt` situé dans le dossier `songs`, en mode lecture, on utilisera le code :

```
f = open("songs/partitition.txt",'r')
```

Puis la fonction `read` permet de récupérer le contenu du fichier dans une chaîne de caractère :

```
p = f.read()
```

L'écriture d'un fichier WAV se fait à l'aide de l'instruction suivante :

```
wavfile.write(wave, rate=freq_e, data=np.array(e).astype(np.int16) )
```

où `wave` est le chemin du fichier à écrire, par exemple `"melodie.wav"`, `freq_e` est la fréquence d'échantillonnage et `e` est l'échantillonnage de la mélodie.

Implémentez une fonction `generer_partition` qui prend en entrée une chaîne de caractère, `fichier`, contenant le chemin du fichier à lire, une seconde chaîne de caractère, `wave`, contenant le chemin du fichier à écrire, la fréquence d'échantillonnage et l'amplitude et qui :

- lit la partition contenue dans `fichier`
- écrit la mélodie correspondante dans le fichier `wav`
- renvoie `None`

Par exemple `generer_partition("songs/weezer.txt","melodie_weezer.wav",44100,2046)` permet de lire la partition `weezer.txt` et de générer la mélodie correspondante `melodie_weezer.wav` avec une fréquence d'échantillonnage de `44100` et une amplitude de `2046`.

2 Chiffrement par Décalage

Le chiffrement par décalage consiste à remplacer une lettre par une autre toujours à distance fixe, toujours du même coté, dans l'ordre de l'alphabet. Si le décalage dépasse la lettre `z` on reprend au début, à la lettre `a`.

Le chiffrement de César est la version la plus simple adoptant ce principe. Il se nomme ainsi, car selon Suétone, Jules César l'utilisait avec l'alphabet grec.

La correspondance peut être obtenue en observant cette roue :

FIGURE 3 – Décalage de 19

Les caractères spéciaux et espaces sont préservés. Avec un décalage de 19 on obtient donc le chiffrement suivant pour `"nadine n'est pas un velociraptor."` :
`"gtwbgx g'xlm itl ng oxehvbktimhk."`.

On appelle alors le décalage utilisé *clé de chiffrement*. La chaîne obtenue après décalage est appelée *chiffrement de la chaîne.*. À partir du chiffrement d'une chaîne et de la clé de chiffrement, l'opération qui permet de récupérer la chaîne originale s'appelle *déchiffrement de la chaîne*.

On se restreint dans cette section à des textes dont les caractères possibles sont :
- les caractères ASCII minuscules : `"abcdefghijklmnopqrstuvwxyz"`
- `" "` , `"\n"`, `"'"`, `"."` et `","` ,

2.1 Chiffrement et Déchiffrement

Question 2.1. *Décalage de caractère*

Implémentez une fonction `decalage` qui prend en entrée un caractère et un décalage `n` et qui renvoie le caractère décalé de `n` s'il s'agit d'un caractère ASCII minuscule, sinon renvoie le caractère lui-même.

Question 2.2. *Chiffrement de César*

En utilisant la fonction `decalage`, implémentez une fonction `chiffrement_cesar` qui prend en entrée une chaîne de caractère et une clé de chiffrement `n` et qui renvoie le chiffrement de César de la chaîne. On considérera que la chaîne contient uniquement des caractères ASCII minuscule, des espaces, points et virgules

Question 2.3. *Déchiffrement de César*

En utilisant la fonction `chiffrement_cesar`, implémentez une fonction `dechiffrement_cesar` qui prend en entrée une chaîne de caractère et une clé de chiffrement `n` et qui déchiffre la chaîne de caractère à l'aide de la clé `n`. On considérera que la chaîne contient uniquement des caractères ASCII minuscule, des espaces, points et virgules.

Question 2.4. *Chiffrement de Fichiers*

Cette question n'est pas notée.

On a vu dans le premier problème comment ouvrir un fichier en lecture et lire le contenu. Voici du code qui permet de lire un fichier et d'écrire un nouveau fichier avec le contenu du premier crypté à l'aide de la clé `12`.

```
f = open('messages/texte_a_crypter.txt','r')
texte = f.read()
texte_crypte = chiffrement_cesar(texte,12)
f_crypte = open('texte_crypte.txt','w') # Ouverture du fichier en ecriture
f_crypte.write(texte_crypte) # Ecriture de texte_crypte dans le fichier
```

Vous pouvez essayer d'adapter le code pour décrypter le fichier `"message_secret_1.txt"` avec la clé 7.

Question 2.5. *Chiffrement de Vigenère - Bonus*

Cette question est notée en bonus.

Le chiffre de Vigenère s'appuie sur le chiffre de César, mais le décalage n'est cette fois pas le même pour toutes les lettres du texte. Le chiffrement est basé sur une clé qui prend la forme d'un mot. On répète ensuite le mot pour obtenir un texte `K` de la même longueur que le texte à crypter. Pour savoir le décalage à appliquer sur la lettre i du texte original, on regarde la lettre $K[i]$ qui défini le décalage à appliquer. Le décalage est le numéro de la lettre dans l'alphabet, ainsi la lettre `"n"` signifie un décalage de 13 lettres.

Le texte `"nadine n'est pas un velociraptor."` chiffré avec la clé "nadine" donne le texte suivant :

```
"nadine n'est pas un velociraptor."  # Texte à crypter
"nadine n adi nen ad inenadinenad "  # Clé répétée
"aagqai a'evb cef uq drpbclzntgou."  # Texte chiffré
```

Implémentez une fonction `chiffrement_vigenere` et `dechiffrement_vigenere` qui prennent en entrée une chaîne de caractère et une clé de chiffrement et qui renvoie respectivement la chaîne chiffrée et la chaîne déchiffrée avec la clé.

Vous pouvez essayer de décrypter le fichier `message_secret_2.txt` avec la clé `"eiffel"` avec vos fonctions.

2.2 Attaques du chiffrement - Bonus

Toute la section est notée en Bonus.

La *cryptanalyse* est la technique qui consiste à déchiffrer un texte chiffré sans posséder la clé de chiffrement. Une *attaque* est un processus visant à essayer de comprendre un message crypté particulier.

Le chiffre de César n'est pas utilisable en pratique, car il peut facilement être attaqué.

Question 2.6. *Brute Force*

Une attaque par force brute consiste à tester une à une toutes les clés de chiffrement possible sur un texte dans le but de trouver la bonne clé.

Dans le chiffrement de César combien de clé de chiffrement possible existe-t-il ? Implémentez une fonction `brute_force` qui prend en entrée une chaîne de caractère, `s`, et qui renvoie la liste des chaînes obtenues en essayant de déchiffrer `s` avec chaque clé de chiffrement possible.

Déchiffrez la chaîne suivante : `"Wz dofowh eis borwbs b'owas dog zs xoapcb."`

Question 2.7. *Analyse de fréquence*

Une autre attaque, si on connaît la langue d'origine du texte, consiste à analyser la fréquence d'apparition des caractères dans le texte et de les comparer à la fréquence moyenne d'apparition des lettres de l'alphabet dans cette langue.

La fréquence d'apparition d'une lettre `l` dans un texte est le nombre $\frac{n_l}{n}$ avec n_l le nombre de fois que la lettre `l` apparaît dans le texte et n le nombre de lettres (minuscules ASCII, hors symboles spéciaux) du texte.

Implémentez une fonction `freq_lettre` qui prend en entrée une chaîne de caractère et une lettre minuscule ASCII et qui renvoie la fréquence d'apparition de la lettre dans la chaîne.

Utilisez la fonction `freq_lettre` pour implémenter une fonction `freq_texte` qui prend en entrée une chaîne de caractère et renvoie une liste avec la fréquence de chaque lettre minuscule ASCII, dans l'ordre alphabétique.

Vous pouvez utiliser `freq_lettre` pour faire l'analyse de fréquence du texte du fichier `message_secret_3.txt` et la comparer aux fréquences théoriques d'apparition des lettres dans un texte en français :

E	17.26 %	P	3.01 %	A	8.40 %
G	1.27 %	S	8.08 %	V	1.32 %
I	7.34 %	B	1.06 %	N	7.13 %
F	1.12 %	T	7.07 %	Q	0.99 %
R	6.55 %	H	0.92 %	L	6.01 %
X	0.45 %	U	5.74 %	J	0.31 %
O	5.26 %	Y	0.30 %	D	4.18 %
K	0.05 %	C	3.03 %	W	0.04 %
M	2.96 %	Z	0.12 %		

```
f = open('messages/message_secret_3.txt','r')
texte = f.read()
freq = freq_texte(texte)
freq
```

Trouvez ainsi la clé utilisée sans faire d'attaque par force brute. Vous pouvez ensuite décrypter le message.

Question 2.8. *Analyse de fréquence avec renvoi de la clé*

On définit la distance entre deux listes de même taille par la formule suivante, pour $\mathcal{L} = [l_1, l_2, \ldots, l_n]$ et $\mathcal{K} = [k_1, k_2, \ldots, k_n]$:

$$\sum_{i=1}^{n} (l_i - k_i)^2$$

Implémentez une fonction `distance` qui prend en entrée deux listes de même taille et qui renvoie la distance entre ces deux listes.

Utilisez la fonction `distance` pour implémenter une fonction `clé` qui renvoie la clé la plus probable, c'est-à-dire celle pour laquelle la distance entre la fréquence du texte décryptée par cette clé est la plus proche que la fréquence théorique.

Vous pouvez essayer de retrouver le résultat de la question précédente par cette méthode.

Question 2.9. *Attaque du chiffre de Vigenère. Longueur de clé connue*

Lorsque l'on connaît la longueur de la clé utilisée pour chiffrer un texte par le chiffrement de Vigenère, une attaque possible consiste à se ramener au cas d'un chiffrement de César.

Supposons que la clé soit de longueur n, et soit $1 \leq i \leq n$, si on considère un caractère sur n du texte en commençant par le $i^{\text{ème}}$, on obtient un texte crypté par le même décalage, celui induit par le caractère i de la clé.

Une attaque par force brute reste possible, mais le nombre de textes généré est 26^n, ce qui peut rapidement devenir long à analyser manuellement. On préférera faire une analyse de fréquence indépendante pour chaque texte extrait afin d'en déduire le caractère i de la clé.

Implémentez une fonction `freq_vigenere` qui prend en entrée un texte et un entier `n` et qui renvoie la clé la plus probable de longueur `n` qui aurait servie à chiffrer le texte.

Vous pouvez essayer de décrypter le texte dans `message_secret_4.txt` en sachant que la clé a une longueur égale à `12`.

Question 2.10. *Calcul de la longueur de clé la plus probable.*

Pour trouver la longueur probable de la clé ayant servi à chiffrer un texte par le chiffre de Vigenère, on utilise l'indice de coïncidence. Cet indice calcul la probabilité, ayant pris deux lettres au hasard dans un texte, qu'elle soit égale. La formule du calcul de l'indice de coïncidence dans un texte est :

$$I_c = \sum_{i=1}^{26} \frac{n_i(n_i - 1)}{n(n-1)}$$

où n_i est le nombre d'apparitions de la $i^{\text{ème}}$ lettre de l'alphabet dans le texte, et n est le nombre de lettres minuscules ASCII (hors caractères spéciaux).

On note $I_f = 0.074$ l'indice de coïncidence théorique d'un texte en langue française et $I_a = 0.038$ l'indice de coïncidence théorique d'un texte dont les caractères sont tirés aléatoirement suivant une loi de probabilité uniforme.

On note une grande différence entre les deux indices. Si la clé est de taille n, le texte obtenu en prenant un caractère sur n devrait avoir un indice d'incidence proche de I_f. Si la clé n'est pas de taille n ,ce même texte devrait avoir un indice d'incidence plus proche de I_a.

Implémentez une fonction `meilleur_incidence` qui prend en entrée une chaîne de caractère et qui renvoie le premier entier n tel que l'indice d'incidence I_c du texte obtenu en prenant un caractère sur n est supérieur ou égal à 0.07.

Décryptez le texte du fichier `message_secret_5.txt`.

Question 2.11. *Question ouverte*

Nous venons de voir des techniques de cryptanalyse permettant d'attaquer des chiffrement de César ou de Vigenère. Ces techniques de chiffrement sont donc à proscrire, car trop facilement attaquable. En cryptographie, on peut souvent rendre la tâche d'un attaquant plus difficile en combinant plusieurs techniques de chiffrement.

Est-ce que ces combinaisons de chiffrements rendent le texte chiffré plus difficile à attaquer ? Dans les trois cas, les clés utilisées dans chaque chiffrement peuvent être différente, et de tailles différentes pour Vigenère.

1. Un chiffrement de César sur un texte déjà chiffré avec le chiffrement de César.
2. Un chiffrement de Vigenère sur un texte chiffré avec le chiffrement de César.
3. Un chiffrement de Vigenère sur un texte chiffré avec le chiffrement de Vigenère.