1 Le cahier des charges

1. Les groupes de 2 ou 3 sont imposés.
2. Le choix des sujets est contraint :
   - une liste de sujets est proposée, tout sujet en dehors de la liste doit être validée par les professeurs qui peuvent le refuser ;
   - deux groupes ne peuvent choisir le même sujet ;
   - les sujets faciles et guidés de la liste sont réservés aux élèves les moins à l'aise ;
   - le choix d'un sujet difficile sera valorisé.
3. Le rendu doit s'effectuer sous trois formes, au plus tard le jeudi 20/05 :
   - Deux **rendus collectifs** (un seul par groupe de projet) :
     - Sur **8 points :** le code du projet et tous les fichiers ressources dans une archive zip nommée `projet-final-eleve1-eleve2-eleve3.zip`.
     - Sur **6 points :** un dossier projet de 5 pages minimums au format `odt`, `pdf` ou `html` (bonus de 0 à 2 points) :
       * Une introduction avec les motivations du projet.
       * Le cahier des charges avec les objectifs poursuivis et la carte mentale du projet (liens entre les différentes fonctions).
       * Le cahier technique avec le manuel d'utilisation, la description des principaux algorithmes et une présentation des choix techniques.
       * La gestion des ressources humaines avec la répartition des tâches, le planning, les outils collaboratifs utilisés.
       * Une conclusion avec un bilan des réussites et des échecs et des pistes de prolongement.
   - Un **rendu individuel** :
     - Sur **6 points :** sous la forme d'une capsule audio de 5 minutes au format `mp3`. Vous pouvez utiliser le service `https://www.mon-oral.net/capsule`

2 Quelques règles à respecter

- **Règle n°1** **Avant de coder je réfléchis crayon en main à l'architecture de mon programme et j'essaie de bien distinguer les données des fonctionnalités.**.
- **Règle n°2** **J'applique le principe de programmation modulaire :« *Chaque fois qu'une tâche peut être clairement séparée dans un programme, je l'encapsule dans une fonction et chaque fonction peut être écrite par une personne distincte.* ».** .
- **Règle n°3** **J'écris des tests unitaires pour chaque fonction de mon programme.**
- **Règle n°4** **Je prête attention à la lisibilité de mon programme : noms de variables et de fonctions explicites, autodocumentation par une docstring pour chaque fonction et commentaires pertinents des points les moins lisibles.**

- **Règle n°5** **Je prête attention à la portée des variables et j'essaie d'éviter l'usage de variables globales dans les fonctions.**
- **Règle n°6** **J'inclus un petit code client dans mon programme afin que le professeur puisse facilement le tester.**
- **Règle n°7** **Je prête attention à la portabilité de mon programme : j'utilise un encodage en UTF-8 et j'insère la directive `# -*- coding: utf-8 -*-` au début de mon script.**
- **Règle n°8** **Je ne perds jamais de vue mes objectifs, je fais un point régulier avec les membres de mon trinôme et les professeurs , j'utilise des outils de travail collaboratif.**

3 Liste de projets

3.1 5 Sujets guidés

Une archive `ressources.zip` est fournie avec les ressources (images et fichiers textes) nécessaires pour traiter les sujets 1, 2 et 4) et un modèle de script `Python` contenant les imports du modules et les fonctions outils nécessaires pour convertir des matrices de pixels en images et réciproquement (pour les sujets 1 et 2). Ce sont les mêmes fonctions que celles utilisées dans le chapitre 7.

3.1.1 Sujet 1 : stéganographie (*moyen*)

La stéganographie consiste à dissimuler un message dans un autre. Par exemple, on veut cacher l'image A `femme.bmp` dans l'image B `cypres.bmp`. Ici les images sont de mêmes dimensions mais il suffit que l'image invisible puisse s'intégrer comme un bloc de pixels dans l'image visible. Le format de stockage est important, les formats de compression avec perte comme JPEG ne permettent pas la stéganographie d'images.

Image visible

Image cachée

Image dévoilée

Pour réaliser cette opération, nous allons utiliser les représentations binaires des valeurs des pixels, et en particulier des opérateurs bit à bit, que nous présentons brièvement ci-dessous :

Méthode *Opérateurs bit à bit*

Pour convertir un entier en une série de bits, il suffit de l'écrire en base 2, on parle d'*écriture binaire*.
L'*écriture décimale* de treize est 13 car $1 \times 10 + 3 = 13$.
Son *écriture binaire* est $(1101)_2$ car $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$.

- En Python la fonction `bin` retour l'écriture binaire d'un entier sous forme de chaîne de caractère préfixée de '0b' et sa réciproque est la fonction `int` avec le paramètre optionnel 2.

```
In [92]: bin(13)
Out[92]: '0b1101'

In [93]: int('0b1101', 2)
Out[93]: 13
```

- L'opérateur » permet de décaler les bits vers la droite, tandis que « les décale vers la gauche :

```
In [94]: 13 >> 1
Out[94]: 6

In [95]: bin(6)
Out[95]: '0b110'

In [96]: 13 << 1
Out[96]: 26

In [97]: bin(26)
Out[97]: '0b11010'
```

- L'opérateur & permet de réaliser une opération logique ET bit par bit entre deux entiers, tandis que l'opérateur | permet de réaliser une opération logique OU bit par bit entre deux entiers :

Table de vérité du ET

x	y	x ET y
0	0	0
0	1	0
1	0	0
1	1	1

Table de vérité du OU

x	y	x OU y
0	0	0
0	1	1
1	0	1
1	1	1

```
In [101]: bin(13), bin(10)
Out[101]: ('0b1101', '0b1010')

In [102]: 13 & 10, bin(13 & 10)
Out[102]: (8, '0b1000')

In [103]: 13 | 10, bin(13 | 10)
Out[103]: (15, '0b1111')
```

On utilise souvent le ET binaire avec un entier particulier appelé *masque* qu'on applique à un autre entier pour faire apparaître une information par « gommage » des bits qui ne sont pas égaux à 1 comme dans le *masque*. Ce peut être pour récupérer l'adresse IP du sous-réseau dont dépend une machine (voir https://fr.wikipedia.org/wiki/Sous-réseau) ou pour fixer des droits par défaut à tout fichier créé en appliquant le masque à la série 111111111 représentant les droits RWXRWXRWX en lecture (R), écriture (W), exécution (X) pour le propriétaire, le groupe principal ou les autres utilisateurs du fichier.

Méthode *Application à la stéganographie*

On va créer une image C contenant à la fois les images A et B, et l'apparence de C sera celle de A car les informations extraites de A seront placées au « premier plan » dans le codage numérique de chaque pixel :

☞ Soit a la valeur d'une des trois composantes (R,G,B) d'un pixel de l'image A et b et c les valeur de la même composante du même pixel respectivement dans les images B et C.

a et b sont des entiers compris entre entre 0 et 255 dont la représentation binaire compte 8 bits.

Lorsqu'on lit de gauche à droite une écriture binaire de 8 bits, les premiers bits lus sont dits de *poids forts* et les derniers de *poids faibles*.

☞ Pour construire les 8 bits de c, on prend par exemple les 5 bits de poids forts de a suivis des $8-5=3$ bits de poids forts de b. On perd de l'information sur a et b : plus on prend de bits de poids forts moins la perte est importante pour b mais plus l'apparence de l'image C s'éloigne de celle de A.

Par exemple avec $a = 166$ d'écriture binaire $a = (\overbrace{10100}^{\text{forts}}110)_2$ et $b = 234$ d'écriture binaire $b = (\underbrace{111}_{\text{forts}}01010)_2$, on construit $c = (\overbrace{10100}^{a}\underbrace{111}_{b})_2$

- Pour annuler les 3 bits de poids faible de 166 on fait d'abord un ET logique entre 166 et le nombre « masque » d'écriture binaire $(11111000)_2$ qui est 248, on obtient $(10100000)_2$.
- Ensuite on décale de 5 bits vers la droite les bits de l'écriture binaire de 234 avec 234 >> 5 ce qui permet de les obtenir comme bits de poids faibles du nombre d'écriture binaire $(111)_2$ (c'est 7).
- Enfin on effectue un OU logique entre $(\overbrace{10100}\underbrace{000})_2$ et $(\underbrace{111})_2$ et on obtient la réunion de tous ces bits : $c = (\overbrace{10100}\underbrace{111})_2$ avec les trois bits de poids forts de $b = 234$ à la place des trois bits de poids faibles de $a = 166$.
- On peut réaliser les mêmes opérations avec les opérateurs de division euclidienne // et % :

```
In [11]: a, b, nbits = 166, 234, 5

In [12]: masque = (255 >> (8 - 5)) << (8 - 5)

In [13]: bin(a), bin(b), bin(masque)
Out[13]: ('0b10100110', '0b11101010', '0b11111000')

In [14]: c = (a & masque) | (b >> 5)

In [15]: bin(c)
Out[15]: '0b10100111'

In [16]: c, bin(c)
Out[16]: (167, '0b10100111')

In [17]: (a & masque) == (a // 2**(8-5)) * 2 ** (8 - 5)
Out[17]: True
```

```
In [18]: (b >> 5) == (b // 2 ** 5)
Out[18]: True

In [19]: c == (a // 2**(8-5)) * 2 ** (8 - 5) + (b >> 5)
Out[19]: True

In [20]: (c << 5) & 255 == (c % 2 ** (8 - 5)) * 2 ** 5
Out[20]: True
```

1. Écrire les fonctions dont on donne les prototypes ci-dessous :

```
def cacher_image(pix_visible, pix_invisible, mode):
    """Prend en paramètres :
    - 2 matrices de pixels pix_visible et pix_invisible de mêmes
        dimensions et profondeur
    - mode de type str pour désigner la profondeur de l'image ('L' ou '
        RGB')
    Retourne une matrice de pixels pix_but de mêmes dimensions et
        profondeur que pix_visible
    l'ecriture binaire d'une composante de pix_but[y][x] est constituée
        ainsi !
    5 bits de poids forts = 5 bits de poids fort de la composante de
        pix_visible[y][x]
    3 bits de poids faible = 3 bits de poids fort de la composante de
        pix_invisible[y][x]
    """

def extraire_image(pix, mode):
    """Prend en paramètres :
    - pix qui est 1 matrice de pixels
    - mode de type str pour désigner la profondeur de l'image ('L' ou '
        RGB')
    Retourne une matrice de pixels pix_but de mêmes dimensions et
        profondeur que pix
    Les 3 bits de poids forts d'une composante de pix_but[y][x]
    sont les 3 bits de poids faible de pix[y][x]
    """
```

2. Écrire deux fonctions similaires qui prennent chacune un paramètre supplémentaire `nbits` qui correspond au nombre de bits de poids forts conservés dans l'image visible (5 dans la version précédente).

3. Écrire deux fonctions de même signatures que les précédentes mais qui utilisent les opérateurs de division euclidienne plutôt que les opérateurs bit à bit.

4. Proposer une interface graphique avec Tkinter.

   Voir https://www.geeksforgeeks.org/loading-images-in-tkinter-using-pil/ pour une intégration d'images PIL dans Tkinter.

3.1.2 Sujet 2 : traitement d'image, filtres de Sobel (*moyen*)

On ne travaille que sur des images en nuances de gris.

Matrice de convolution

a	b	c
d	e	f
g	h	i

Voisinage du pixel de coordonnées (x, y)

$(x-1, y-1)$	$(x+0, y-1)$	$(x+1, y-1)$
$(x-1, y+0)$	(x, y)	$(x+1, y+0)$
$(x-1, y+1)$	$(x+0, y+1)$	$(x+1, y+1)$

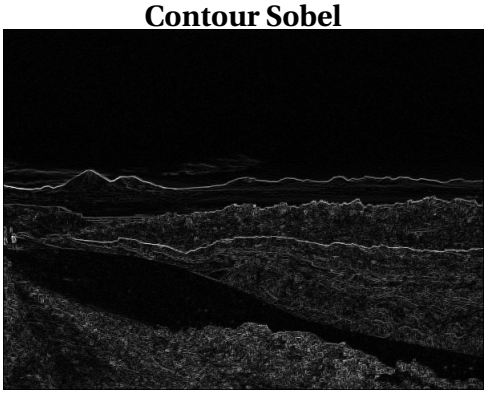
Le filtre de Sobel est un filtre de détection de contour qui fait appel à des opérations de convolution entre le voisinage carré 3×3 d'un pixel de coordonnées (x, y) et deux matrices de coefficients de mêmes dimensions. On calcule ainsi une approximation du gradient de l'intensité du pixel, équivalent de la dérivée dans un espace à deux dimensions. Ce gradient mesure donc la variation d'intensité au voisinage du pixel et il est d'autant plus grand et proche de 255 (blanc) si le pixel est sur un contour.
Avec les notations du schéma ci-dessus le produit de convolution de la matrice de coefficients à gauche par le voisinage du pixel à droite s'obtient comme une somme des produits de coefficients par les pixels en même position. On note $\text{pix}[y][x]$ le pixel d'abscisse x et d'ordonnée y.

$$\begin{aligned} &a \times \text{pix}[y-1][x-1] + b \times \text{pix}[y-1][x] + c \times \text{pix}[y-1][x+1] \\ &\quad + d \times \text{pix}[y][x-1] + e \times \text{pix}[y][x] + f \times \text{pix}[y][x+1] \\ &\quad + g \times \text{pix}[y+1][x-1] + h \times \text{pix}[y+1][x] + i \times \text{pix}[y+1][x+1] \end{aligned}$$

Par exemple le produit de convolution de la matrice de coefficients $A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$ par le voisinage de pixels $B = \begin{pmatrix} 50 & 10 & 80 \\ 60 & 40 & 20 \\ 70 & 30 & 90 \end{pmatrix}$ est $1 \times 50 + 2 \times 10 + 1 \times 80 + 2 \times 60 + 4 \times 40 + 2 \times 20 + 1 \times 70 + 2 \times 30 + 1 \times 90$.
Le filtre de Sobel calcule d'abord pour chaque pixel de coordonnées (x, y) les produits de convolution de son voisinage carré 3×3 par les matrices $C = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$ pour la variation d'intensité horizontale et $D = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$ pour la variation d'intensité verticale. En notant p_h et p_v ces deux produits, on calcule ensuite leur norme euclidienne $\sqrt{p_h^2 + p_v^2}$. Enfin on la normalise pour avoir une valeur entière entière entre 0 et 255, en prenant sa partie entière et en minorant par 255 si elle excède 255.
Voici un exemple :

Original

Contour Sobel



1. Écrire les fonctions dont on donne les prototypes ci-dessous :

```
from math import sqrt

def moyenne_rgb(pixel, coef):
    """Prend en paramètres :
    - pixel un tableau de trois entiers entre 0 et 255 représentant un
        pixel RGB
    - coef un tableau de 3 nombres positifs
    Retourne un entier qui est la partie entière
    de la moyenne pondérée des composantes RGB de pixel par les
        coefficients de coef
    """


def monochrome(pix, coef):
    """Prend en paramètres :
    - pix une matrice de pixels RGB
    - coef un tableau de 3 nombres positifs
    Retourne une matrice de pixels pix_but représentant la conversion
    en nuances de gris de l'image de pix,
    avec les coefficients de pondération coef"""


def detection_contour(pix, filtre_contour):
    """Prend en paramètres :
    - pix une matrice de pixels RGB
    - filtre_contour une fonction qui peut modifier la valeur d'un pixel
      en marquant les pixels de contour, les pixels du bord ne sont pas
          traités
    Retourne une nouvelle matrice de pixels pix_but
    obtenue en appliquant filtre_contour à chaque pixel de coordonnées (x
        ,y)"""

sobel_vertical = [[-1, -2, -1],
                  [0,0,0],
                  [1, 2, 1]]

sobel_horizontal = [[-1,0,1],
                    [-2,0,2],
```

```
                    [-1,0,1]]

def normalise(val):
    """Prend en paramètre un nombre val
    Retourne 0 si val <= 0
    partie entière de val si 0 <= val <= 255
    et 255 sinon"""


def convolution_bloc3(pix, x,y, mat_conv):
    """Prend en paramètres :
    - pix une matrice de pixels
    - x et y deux entier désignant les coordonnées d'un pixel de pix
    - mat_conv une matrice de convolution 3 x 3
    Retourne le produit de convolution de mat_conv
    par le voisinage carré 3x3 du pixel en (x,y)
    Contrainte : utiliser un seul opérateur +
    """


def filtre_contour_sobel(pix, x, y):
    """Prend en paramètres :
    - pix une matrice de pixels
    - x et y deux entier désignant les coordonnées d'un pixel de pix
    Retourne l'entier calculé par le filtre de Sobel
    pour le pixel en (x, y), voir l'énoncé
    et https://fr.wikipedia.org/wiki/Filtre_de_Sobel"""
```

3.1.3 Sujet 3 : programmation d'un jeu de pendu (*facile*)

- L'ordinateur choisit aléatoirement un mot dans le fichier `dicoSansAccents.txt` fourni.
- Le joueur doit deviner le mot en un nombre fixé de tours, à chaque tour il peut choisir entre la proposition d'une lettre ou du mot complet. Si le lettre proposée se trouve dans le mot l'ordinateur affiche le mot avec les lettres trouvées ou des tirets bas à la place des lettres non découvertes. Si le lettre a déjà été proposée ou si elle n'appartient pas au mot, l'ordinateur le signale au joueur. La boucle se termine si le joueur a proposé le mot ou si le nombre de tours maximal est atteint.
- A la fin de la partie, l'ordinateur affiche un message de conclusion.
- Proposer une petite interface graphique avec Tkinter.

3.1.4 Sujet 4 : programmation d'un moteur basique de système de saisie prédictive T9 utilisé sur les téléphones mobiles à touches (*moyen*)

Source : CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=111558

Voici le cahier des charges :

1. Lire l'article de Wikipedia sur ce système https://fr.wikipedia.org/wiki/Saisie_intuitive#T9.
2. Récupérer auprès de l'enseignant le fichier `motsFrSansAccents.txt` qui contient la liste des fréquences par million d'occurrences de 123 150 mots sans accents de la langue française. Ce fichier a été constitué à partir du fichier `liste_mots.txt` récupéré sur le site http://www.lexique.org.

```
def fichier2dico(fichier, delimiteur = '\t', charset = 'latin1'):
    """Parcourt un fichier comme motsFrSansAccents.txt
    Renvoie un dictionnaire Python associant à chaque mot sa fréquence.
        """
```

```
In [9]: dico = fichier2dico('motsFrSansAccents.txt')

In [10]: dico['la']
Out[10]: 25094.9
```

3. Écrire une fonction `mot2code` qui prend en argument un mot sans accent et qui renvoie son codage T9 sous la forme d'une chaîne de caractères.

```
In [33]: mot2code('python')
Out[33]: '798466'
```

4. Écrire une fonction `listPrefixe(mot)` qui prend en argument un mot et qui renvoie la liste de tous ses préfixes.

```
In [34]: listPrefixe('python')
Out[34]: ['p', 'py', 'pyt', 'pyth', 'pytho', 'python']
```

5. Écrire une fonction `dico2freq(dico)`.

   Elle prend en argument un dictionnaire Python, des fréquences de mots, retourné par la fonction `fichier2dico`, et elle renvoie un dictionnaire qui associe à une chaîne de caractères, la somme des fréquences des mots qu'elle préfixe. On parle de poids du préfixe.

```
In [35]: freq = dico2freq(dico)

In [36]: freq['scie']
Out[36]: 254.38000000000002
```

6. Écrire une fonction `freq2proposition(freq)`, qui prend en argument un dictionnaire `freq` retourné par la fonction `dico2freq` et qui retourne un dictionnaire associant à chaque séquence de touche saisie sur un clavier T9 la chaîne de caractères qu'elle code et qui constitue un préfixe de poids maximal.

   Par exemple, la séquence de touches `'7243'` peut coder `'scie'` ou `'chaîne'` mais `'scie'` est le préfixe de poids maximal pour cette séquence [1], c'est donc la proposition choisie. En pratique, le logiciel du téléphone permet de parcourir la liste des préfixes dans l'ordre décroissant de leur poids.

```
In [37]: prop = freq2proposition(freq)

In [38]: prop['7243']
Out[38]: 'scie'

In [39]: freq['scie']
Out[39]: 254.38000000000002

In [40]: freq['sage']
Out[40]: 57.41
```

7. Proposer une interface graphique avec Tkinter : clavier T9 et affichage du mot le plus probable.

3.1.5 Sujet 5 : chiffrements de César et Vigenère (*moyen*)

- La **cryptographie** (du grec *kruptos* : caché, et **graphein** : écrire) qui est l'art de coder le message d'une façon connue uniquement de l'émetteur et du récepteur. On appelle **chiffre** un procédé de cryptage d'un texte caractère par caractère.

- D'après la légende, César aurait chiffré sa correspondance avec un **chiffre par substitution monoalphabétique** : chaque lettre de l'alphabet est remplacée dans le texte chiffré par une autre lettre, toujours la même. Il existe aussi des **chiffres par substitution polyalphabétique** comme le chiffre de Vigenère : chaque lettre de l'alphabet est remplacée dans le texte chiffré par une autre lettre, mais qui varie selon la position dans le message.

  Dans le chiffre de César, chaque lettre du texte chiffré s'obtient par un décalage de la lettre du texte en clair. Ce décalage est la clef du chiffre, pour le chiffre de César cette clef est 3 : A est chiffré par D, B par E,W par Z et X par A.

  Notre alphabet comptant 26 lettres, on peut repérer A par 0, B par 1 ...Z par 25.

  Le chiffre de César peut alors se modéliser sous la forme d'une fonction mathématique qui à une lettre en clair repérée par x avec $0 \leqslant x \leqslant 25$ associe une lettre chiffrée repérée par $y \equiv x+3 \bmod 26$.

  Cette notation se lit y congru à $x+3$ modulo 26 et signifie que y est égal au reste de la division euclidienne de $x+3$ par 26. On peut changer la clef de décalage et si on prend 13 on obtient le chiffrement `rot13` déjà rencontré.

Lettre en clair	A	B	...	W	X	Y	Z
x	0	1	...	22	23	24	25
$y \equiv x+3 \bmod 26$	3	4	...	25	0	1	2
Lettre chiffrée	D	E	...	Z	A	B	C

1. D'après notre fichier `motsFrSansAccents.txt`

- Pour déchiffrer un message codé par le chiffre de César, si le message comporte suffisamment de caractères, on peut procéder par *analyse fréquentielle*.

  On recherche la clef de déchiffrement, entre 0 et 25, qui minimise la distance entre l'histogramme du texte déchiffré et l'histogramme de répartition des lettres dans la langue ciblée.

  Par exemple, en Français, la liste des fréquences des lettres minuscules de `'a'` d'indice 0 à `'z'` d'indice 25 est :

```
freqref = [8.4, 1.06, 3.03, 4.18, 17.26, 1.12, 1.27, 0.92, 7.34, 0.31,
    0.05, 6.01, 2.96, 7.13, 5.26, 3.01,0.99, 6.55, 8.08, 7.07, 5.74,
    1.32, 0.04, 0.45, 0.3, 0.12]
```

- Le chiffre de Vigenère est un **chiffre de substitution polyalphabétique** : chaque lettre en clair est remplacée par une lettre chiffrée mais cette traduction varie selon la position de la lettre en clair dans le message.

  Il s'agit d'un chiffre de César dont le décalage change selon la position de la lettre en clair dans le message.

  Par exemple, soit le texte en clair IL FAIT BEAU et la clef PLUIE.

  Sur la première ligne on écrit le texte en clair sans espaces, sur la deuxième on répète la clef sur la même longeur, sur la troisième on écrit le texte chiffré :

  ILFAITBEAU

  PLUIEPLUIE

  XWZIMIMYIY

  Pour chaque lettre en clair, le rang de la lettre de la clef en dessous donne le décalage qu'on applique pour déterminer la lettre chiffrée. Les rangs varient de 0 pour A à 25 pour Z.

  P de rang 15 en dessous de I donc I décalée de 15 donne X

  L de rang 11 en dessous de L donc L décalée de 11 donne W

  U de rang 20 en desous de F donc F décalée de 20 donne Z ...

Voici le cahier des charges :

➟ Compléter le code de la fonction `chiffreCesar(source, decalage)` ci-dessous qui prend en argument une chaîne de caractères `source` et `decalage` un entier entre 0 et 25. La chaîne `source` ne doit pas comporter de caractères diacritiques, elle est convertie d'abord en majuscules, puis une chaîne `sortie` est construite en remplaçant chaque caractère alphabétique par son image par le chiffre de César de clef `decalage`.

```
def chiffreCesar(source, decalage):
    source = source.upper()
    sortie = ''
    for c in source:
        if c.isalpha():
            #à  compléter
        else:
            sortie += c
    return sortie
```

- Écrire une fonction `dechiffreCesarClef(source, decalage)` où `decalage` est la clef de chiffrement utilisée.

```
In [10]: chiffreCesar('Un python est un serpent constricteur.', 1)
Out[10]: 'BU WFAOVU LZA BU ZLYWLUA JVUZAYPJALBY.'

In [11]: dechiffreCesarClef('BU WFAOVU LZA BU ZLYWLUA JVUZAYPJALBY.', 1)
Out[11]: 'GZ BKFTAZ QEF GZ EQDBQZF OAZEFDUOFQGD.'
```

- Écrire une fonction `calculerFrequence(source)` qui retourne l'histogramme des fréquences (en %) des 26 caractères alphabétiques, dans une chaîne de caractères `source` composée de caractères non accentués et en majuscules.

- Écrire une fonction `calculerDistance(freq1, freq2)` qui calcule la distance euclidienne entre deux listes d'histogrammes de fréquences des 26 caractères alphabétiques.

  Si on note $(f_i)_{0\leqslant i\leqslant 25}$ et $(g_i)_{0\leqslant i\leqslant 25}$ les séries de fréquences des deux histogrammes, la distance euclidienne entre les deux histogrammes est égale à :

$$\sqrt{(f_0-g_0)^2+(f_1-g_1)^2+\cdots+(f_{25}-g_{25})^2}$$

- Écrire une fonction `calculerDecalage(source, freqref)`, qui prend en argument une chaîne de caractères `source` et une liste `freqref` représentant l'histogramme des caractères alphabétiques dans la langue ciblée (voir ci-dessus pour le Français). Cette fonction doit retourner le décalage entre 0 et 26 pour lequel l'histogramme de l'image de `source` par le chiffre de César présente une distance minimale avec `freqref`.

  En déduire une fonction `dechiffreCesar(source, freqref)` qui déchiffre un message crypté par le chiffre de César.

  Tester ces fonctions avec le contenu du fichier texte `clair.txt` qui contient le début du roman « *Les trois mousquetaires* ».

```
In [34]: f = open('clair.txt','r')

In [35]: message = f.read().upper()

In [36]: chiffre = chiffreCesar(message, 3)

In [38]: dechiffreCesar(chiffre, freqref) == message
Out[38]: True
```

- Écrire une fonction `chiffreVigenere(source, clef)` qui chiffre une chaîne de caractère `source` avec le chiffre de Vigenère pour une `clef` donnée.

  En déduire une fonction `chiffreVigenereClef(source, clef)` qui déchiffre une chaîne de caractères cryptée avec le chiffre de Vigenère pour une `clef` donnée.

```
In [62]: chiffreVigenere('AMSTERDAM', 'ROME')
Out[62]: 'RAEXVFPED'

In [63]: dechiffreVigenereClef('RAEXVFPED', 'ROME')
Out[63]: 'AMSTERDAM'
```

➟ Proposer une petite interface graphique en Tkinter qui permette de chiffre ou déchiffre un texte saisi par l'utilisateur avec le chiffre de César ou celui de Vigenère.

3.2 8 Sujets non guidés

- **Chou, Chèvre, Loup** ⇒ *Moyen*

  À partir de la ressource https://alainbusser.frama.io/NSI-IREMI-974/WGC.html, résoudre le problème **Chou, Chèvre, Loup** sans utiliser le paradigme de programmation objet ni l'instruction `exec`.

  Proposer une interface graphique en Tkinter permettant à un joueur de choisir un déplacement du chou, de la chèvre ou du loup d'une rive à l'autre, lui indiquant si ce déplacement est possible et le félicitant s'il résout le problème.

- **Le jeu de la vie** ⇒ *Moyen*

  Sur un damier carré, on dispose des créatures de manière aléatoire. La population évolue d'un état au suivant selon les règles suivantes :

  – une créature survit si elle a 2 ou 3 voisines dans les 8 cases adjacentes et elle meurt à cause de son isolement ou de la surpopulation sinon,
  – une créature naît dans une case vide s'il y a exactement 3 créatures dans les 8 cases voisines, et rien ne se passe dans cette case sinon.

  Écrire avec la bibliothèque Tkinter de Python un programme qui simule le développement d'une population.

  Commencer par traiter ce problème https://www.codingame.com/training/medium/game-of-life.

  Voir https://www.academie-sciences.fr/fr/Seances-publiques/le-jeu-de-la-vie.html.

- **Jeu de Pong** ⇒ *Difficile*

  Écrire une version Python du jeu Pong (ou de Space Invaders, Snake ...) On utilisera la bibliothèque Tkinter pour Python.

- **Jeu de puissance 4** ⇒ *Difficile*

  Réaliser une version du jeu de Puissance 4 pour 2 joueurs avec interface graphique.

  Prolongement possible : un joueur joue contre l'ordinateur.

- **Princesse de Clèves en Emoji** ⇒ *Moyen*

  Niveau 1 du sujet proposé par Laurent Abbal sur https://github.com/hackathon-nsi/h7n-nsi-02.

  – Lire l'article Wikipedia sur les Émoji.
  – Revoir les encodages ASCII et Unicode.
  – Visiter et étudier : https://unicode.org/emoji/charts/full-emoji-list.html
  – Étudier la première partie de La Princesse de Clèves et faire une première liste de termes qui pourraient être remplacés par des Émoji.
  – Implémenter deux algorithmes de recherche d'un motif dans un texte :

    * l'algorithme naif
    * l'algorithme de Boyer-Moore Horspool.

  Voir ce document

  et https://pixees.fr/informatiquelycee/n_site/nsi_term_algo_boyer.html.

- Écrire un programme en Python qui remplace les termes de la liste précédente par les Émojis correspondants en utilisant l'algorithme de Boyer-Moore Horspool.

- **Fractales et L-systèmes** ⇒ *Facile*

  Écrire un programme avec un interface graphique qui permet de saisir un motif initial puis des règles de réécriture et quelques paramètres graphiques puis trace des représentations approchées de fractales selon un algorithme de L-système : flocon de Von Koch, triangle de Sierpinski ...

  Proposer une petite interface graphique avec Tkinter.

- **Réalisation d'une calculatrice en notation Polonaise inverse** ⇒ *Difficile*

  - Écrire un programme d'émulation de calculatrice qui effectue des calculs élémentaires (addition, soustraction, multiplication, division) en notation *postfixée* ou *polonaise inverse*. Proposer une petite interface graphique en Tkinter. Commencer par résoudre ce problème : https://www.codingame.com/ide/puzzle/reverse-polish-notation
  - Rajouter une fonctionnalité qui traduit une expression en notation postfixée dans une expression en notation infixée avec un nombre minimal de parenthèses. Commencer par résoudre ce problème : https://www.codingame.com/training/medium/the-polish-dictionary.

- **Sudoku** ⇒ *Difficile*

  Réaliser un solveur de Sudoku par backtracking, avec ou sans interface graphique. Source : le problème 1 de https://www.apmep.fr/IMG/pdf/CAPES_avril_2017_epreuve_info.pdf.

4 Quelques bibliothèques de Python qui pourraient vous être utiles

☞ Réalisation d'interfaces graphiques plutôt statiques : **tkinter** (sous Python 3) ou **Tkinter** (Python2) :

http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html

☞ Traitement d'images : **PIL** ou son fork **Pillow** : https://pypi.python.org/pypi/Pillow/

☞ Bibliothèques pour le calcul scientifique : **numpy** ou plus généralement **scipy** :

http://scipy.org/

☞ Réalisation de graphiques pour les mathématiques ou la physique (possibilité d'animation) : **matplotlib** : http://matplotlib.org/

☞ Collecte de données sur le Web :

  - récupérer les données avec **requests** : https://realpython.com/python-requests/
  - « parser » du HTML avec **beautiful soup** : https://realpython.com/beautiful-soup-web-scraper-

5 Quelques outils pour la gestion de projet ou le travail collaboratif

- ☞ Cours et ressources en ligne sur https://parc-nsi.github.io/premiere-nsi/.
- ☞ Vous pouvez créer un dépôt privé de fichiers sur Github ou Gitlab. Pour l'utilisation de git, voir cette https://rogerdudler.github.io/git-guide/index.fr.html.
- ☞ Partage de fichiers, pad, articles de blog, notebooks Python dans l'ENT : https://le-parc.ent.auvergnerho fr/
- ☞ Partage de scripts Python et test en ligne : https://console.basthon.fr/.
- ☞ Documents textuels collaboratifs (avec possibilité de chat) : **Framapad** accessible depuis https://framapad.org/.
- ☞ Feuilles de calculs collaboratives : **Framacalc** accessible depuis https://framacalc.org/.
- ☞ Réalisation de carte mentale : **Framindmap** accessible depuis https://framindmap.org/c/maps/.

ANNEXE 1 : Fiche de suivi du projet (à remplir pour le premier point étape)

Date	**Description**	**Etat actuel** rien/en cours/fini
Objectifs et motivations du projet / Cahier des charges		
Architecture du projet		
Structures de données principales		
Principales fonctions		
Interaction Homme Machine		
Tests prévus		
Bibliothèques utilisées		
Références / Sitographie		
Planification et répartition des tâches		