Fonction genereCode( ) -> list

- Création d'une liste contenant les entiers de 1 à 27
- Mélange la liste (possibilité d’utiliser shuffle (opération en place donc agit directement sur la liste)
- Ajout du 0 à la liste
- Liste retournée

Principe de la fonction chiffrer :

Entrées :
Chaine : BONJOUR
cle : [0, 5, 13, 3, 2, 25, 19, 9, 20, 16, 14, 15, 8, 4, 6, 22, 18, 1, 23, 26, 10, 21, 11, 24, 17, 12, 7]

	B	O	N	J	O	U	R
Etape 1 :	[2,	15,	14,	10,	15,	21,	18]

=> appel à la fonction encode

Etape 2 :	[13,	22,	6,	14,	22,	21,	23]

=> recherche de la correspondance d'indices => remplissage liste des indices chiffrés
Exemple : A l'indice 2, on trouve la valeur 13 dans la clé,…

Etape 3 :	M	V	F	N	V	U	W

=> appel à la fonction decode

Principe de la fonction dechiffrer :

Entrées :
Chaine : MVFNVUW
cle : [0, 5, 13, 3, 2, 25, 19, 9, 20, 16, 14, 15, 8, 4, 6, 22, 18, 1, 23, 26, 10, 21, 11, 24, 17, 12, 7]

	M	V	F	N	V	U	W
Etape 1 :	[13,	22,	6,	14,	22,	21,	23]

=> appel à la fonction encode

Etape 2 :	[2,	15,	14,	10,	15,	21,	18]

=> recherche de la correspondance d'indices => remplissage liste des indices déchiffrés
Exemple : 13 est à l'indice 2 de la clé, …

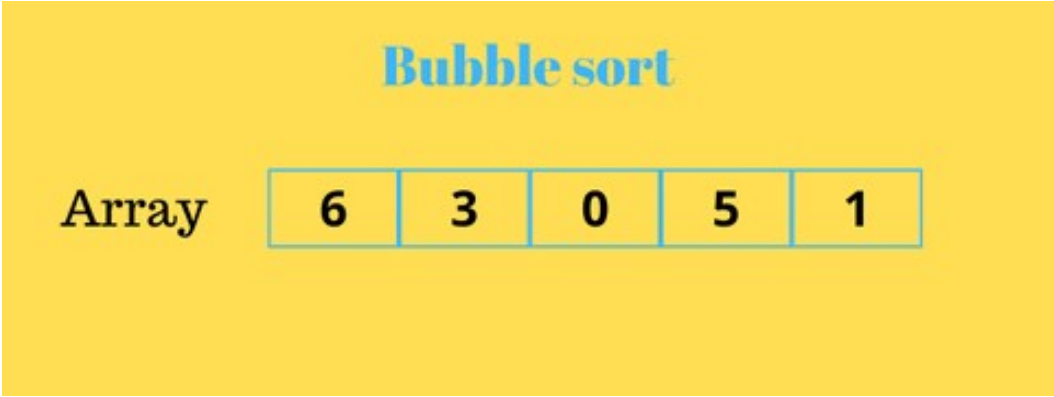
Etape 3 :	B	O	N	J	O	U	R

=> appel à la fonction decode

Rappels : Lecture dans un fichier

```
# ouverture du fichier

# Chemin en absolu. Simple à ecrire mais attention
#: si on veut lancer ce script dans un autre environnement :
#  il faut modifier le chemin d'acces au fichier
fic = open('C:\\Users\\eclermont\\CPGE IPT\\EC\\ITC_S3_TD0\\ducote.txt', 'r')
texte_reference = fic.read() # recuperation du contenu du fichier
fic.close()

#En recuperant le chemin du dossier dans lequel se situe le fichier python courant
import os.path
chemin_dossier = os.path.abspath(os.path.dirname(__file__)) # recuperation du chemin courant du script
print(chemin_dossier)     # affiche c:\Users\eclermont\CPGE IPT\EC\ITC_S3_TD0

# ajout du nom du fichier au chemin (si fichier au meme niveau que le script)
chemin_complet = os.path.join(chemin_dossier, "ducote.txt")
print(chemin_complet)     # affiche c:\Users\eclermont\CPGE IPT\EC\ITC_S3_TD0\ducote.txt
fic = open(chemin_complet, "r")

texte_reference = fic.read() # recuperation du contenu du fichier
fic.close()

#Autre syntaxe avec with open :
with open(chemin_complet) as fic:
        texte_reference = fic.read()
```

Principe du tri à bulles :
On compare répétitivement les éléments consécutifs d'une liste ou d'un tableau,
et on les permute lorsqu'ils sont mal triés.



Il doit son nom au fait qu'il déplace rapidement les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface d'un liquide.

Principe simple mais algorithme lent ($0(n^2)$)

```
tri_bulles( tab )                                   Attention : Ici, tri par ordre croissant
 pour i de 1  à taille(tab)-1
        pour j de 0 à taille(tab)- i-1
                si tab[j+1] < tab[j]
                        tab[j+1], tab[j] = tab[j], tab[j+1] #permutation
        finpour
finpour
```

Optimisation : Interrompre dès qu'un parcours des éléments possiblement encore en désordre (boucle interne) est effectué sans la moindre permutation : cela signifie que tout le tableau est trié. Cette optimisation nécessite une variable supplémentaire (drapeau/flag).

```
tri_bulles_optimise( tab )
        ....                    #flag initialisé à false pour entrer dans la boucle tant que
        i =  ....
        tant que    ....
                ....
                pour j de 0 à taille(tab)- i-1
                        si tab[j+1] < tab[j]
                                tab[j+1], tab[j] = tab[j], tab[j+1]
                                ....        #indique qu'au moins une permutation a été effectuée
                        finsi
                finpour
               i= ....
        fintantque
```

Optimisation : Interrompre dès qu'un parcours des éléments possiblement encore en désordre (boucle interne) est effectué sans la moindre permutation : cela signifie que tout le tableau est trié. Cette optimisation nécessite une variable supplémentaire (drapeau/flag).

```
tri_bulles_optimise( tab )
        est_trie = false #flag initialisé à false pour entrer dans la boucle tant que
        i = 1
        tant que   i <taille(tab) – 1 et    est_trie  == false
                est_trie  = true
                pour j de 0 à taille(tab)- i-1
                        si tab[j+1] < tab[j]
                                tab[j+1], tab[j] = tab[j], tab[j+1]
                                 est_trie  = false
                                        #indique qu'au moins une permutation a été effectuée
                         finsi
                finpour
               i= i+1
        fintantque
```

frequencesCaracteres(texte : str) -> np.array

- Création d'un tableau tab_frequences de 27 cases ne comportant que des 0

⇒Utilisation possible de la fonction np.zeros

- Transformation du texte en liste d'entiers

(manipulation plus simple avec des entiers qu'avec des caractères) => fonction encode

BONJOUR
[2,15,14,10, 15, 21, 18]

- Parcours d'une boucle de 0->26 pour remplir tab_frequences

A l'indice 0, on cherche le nombre de fois que l'on trouve un 0 (possibilité d'utiliser la fonction count sur la liste d'entiers)

…. Et on le stocke à la position 0 de tab_frequences

A l'indice i, on cherche le nombre de fois que l'on trouve la valeur i

…. Et on le stocke à la position i de tab_frequences

….

- Suppression de la case correspondant à 0 (espace) : on ne veut pas stocker la fréquence des espaces

- Calcul de la fréquence : Pour chaque élément de tab_frequences, on divise le nombre d'éléments par le nombre total de caractères (hors espaces) => possibilité d'utiliser la fonction sum des np.array qui permet d'obtenir la somme des valeurs contenues dans un tableau.
- Tableau tab_frequences retourné.

afficheFrequences(freq : np.array)

- Importer la librairie matplotlib.pyplot pour les tracés de graphiques
- Création d'une liste par compréhension comportant les caractères A-Z (qui servira pour afficher sur l'axe des abscisses)

  possibilité de générer la liste en utilisant les codes ASCII des caractères
- Appel à la fonction bar pour définir le graphique à tracer en transmettant la liste de caractères de A à Z, le tableau de fréquences et la couleur
- Définition des titres des abscisses et ordonnées (fonctions xlabel et ylabel) et du titre (fonction title)
- Affichage (fonction show)

def trieFrequencesCaracteres(frequences: np.array)-> list:

- Création d'une liste indices des indices initialisée de 0 à 25

- Tri à bulles (par ordre décroissant)
  - Penser à agir sur les valeurs de la liste indices en plus des valeurs dans le tableau frequences (même permutations à effectuer)

- Liste indices à retourner