Nous allons nous beaucoup nous servir des concepts utilisés dans le TD3. Commençons par un résumé des outils

I Quelques outils

1.1 Sur les chaînes de caractères

Une chaîne de caractères est un objet informatique qui représente du texte. Techniquement, en python, on les distingue par des guillemets simples ou doubles.

```
s = ''Une chaîne d'exemple''
len(s)  # le nombre de caractères, espaces compris
s[0]  # le premier caractère
s[5] # le 6ème caractère
```

```
for i in range(len(s)):
    # i est un entier, représentant un indice dans la chaîne.
    # i va prendre les valeurs 0, 1, 2...
    s[i]  # il s'agit du caractère d'indice i
```

ou alors

```
for c in s:
    # c est directement un caractère
    # Avec notre exemple de chaîne s, c va prendre comme valeurs
    # successives ''U'', ''n'', ''e'', '' ''...
```

1.2 Sur les dictionnaires

Un **dictionnaire** est un autre type de données utilisable en python. Il permet de stocker des **valeurs** associées à des **clés**.
Plus précisément, un dictionnaire est un ensemble de (clé, valeur) où chaque valeur est associée à exactement une clé.

Pour créer un dictionnaire, on utilise des accolades de manière similaire aux crochets utilisés pour les listes.

```
d = {}  # un dictionnaire vide
```

```
d = {''a'': 12}  # un dictionnaire contenant une paire de (clé, valeur)
# la clé est la chaîne ''a'' et la valeur le nombre 12
d[''a'']  # c'est la valeur associée à la clé ''a'', ici 12
d[0] = [2 ,4, 6]  # on crée un nouvelle paire (clé, valeur). La clé est un nombre
# et la valeur une liste.
```

- Les clés utilisables pour nous sont : des chaînes, des nombres. En toute généralité, tout objet non modifiable peut être utilisé comme clé.
- Les valeurs sont des objets python quelconques (nous connaissons : nombres entiers ou flottants, booléen, chaîne, liste, dictionnaire)

Syntaxe générale

`d` désigne un dictionnaire.

- `{}` est le dictionnaire vide.
- `d[c]` permet d'accéder à la valeur associée à la clé contenue dans la variable `c`. Par exemple `d["a"]` est la valeur associée à la chaîne de caractère `"a"`.
- `d[c] = v` associe la valeur contenue dans `v` à la clé `c`. Ceci ajoute une paire (clé, valeur) dans `d` ou remplace la valeur associée à `c` suivant que `c` était ou non une clé présente dans `d`
- `c in d` est un booléen indiquant si la clé `c` est présente dans `d`.
- `for c in d:` permet d'obtenir séquentiellement dans la variable `c` toutes les clés contenues dans `d`.

II Un texte chiffré

Le fichier **texte_chiffre.txt** contient un texte qui a été *chiffré* par un procédé assez simple : on a remplacé chaque occurrence de chaque lettre par une autre lettre. Par exemple tous les “a” auraient pu être remplacés par des “j”.
Le but de ce TD est de décrypter ce texte (retrouver le texte de départ, sans connaître a priori la manière exacte dont il a été chiffré). Pour simplifier l'étude, on considère que le de départ est entièrement en minuscules (mais on a conservé les caractères spéciaux comme les caractères accentués, la ponctuation...).
On a également conservé les paragraphes originaux, c'est à dire que le caractère spécial "$\backslash n$" (qui représente un retour à la ligne) n'a pas été changé.

Pour les tests et le but final de notre TP, la variable `chiffre` est créée dans la feuille de script et elle contient notre texte chiffré avec les conventions précédentes.

2.1 Analyse des occurrences

Une première approche simple est de comparer les fréquences d'apparition de chaque lettres dans le texte chiffré avec un tableau de référence. Nous utiliserons [1] le tableau wikipedia.
Pour stocker les occurrences de chaque lettre, nous allons construire un dictionnaire dont les clés sont les lettres du texte et les valeurs le nombre d'occurrences. Par exemple

```
s = ''texte''
# et le dictionnaire que l'on veut calculer sera
d = {''t'': 2, ''e'': 2, ''x'': 1}
```

Exercice 1
Compléter la fonction `toutes_occurrences(texte: str) -> dict` qui prend comme argument un texte (une chaînes de caractères) et retourne le dictionnaire des occurrences de ce texte.

2.2 Premier pas en décryptage

Le caractère d'espace est un caractère comme les autres en informatique. Il est logiquement le plus fréquent dans nos écrits.

Exercice 2
En utilisant la fonction `toutes_occurrences` sur la variable appelées `chiffre` dans votre fichier, construire le dictionnaire des occurrences de notre texte chiffré (et stocker ce dictionnaire dans une variable de la console, pour utilisation future).
Compléter maintenant la fonction `lettre_majoritaire(dico: dict) -> str` qui prend comme argument un dictionnaire d'occurrences et retourne la lettre ayant le plus d'occurrence.
Comment a été chiffré l'espace dans notre texte ?

Rappel : on peut facilement concaténer des chaînes de caractères et c'est la méthode à utiliser pour construire pas à pas une chaîne :

```
s1 = ''ab''
s2 = ''cd''
s = s1 + s2 # maintenant s contient ''abcd''
```

Exercice 3
L'étape suivante consiste à échanger deux caractères dans notre texte : l'espace et le caractère trouvé précédemment, et noter sur une feuille qu'on a déjà décrypté le caractère d'espace.
Compléter la fonctions `remplace(chaine: str, car1: str, car2: str) -> str`.

2.3 Finir le job

On peut maintenant trouver un texte où les espaces sont bien placés. Il nous reste à deviner pas à pas chacune des autres lettres en étant guider par le tableau donné en lien au début du TD. On peut trouver facilement la lettre ”e” pour commencer, et ensuite c'est un jeu de devinette.

Votre feuille de script contient la fonction `tri_dico` qui permet d'avoir la liste des occurrences triées par ordre décroissant

Quelques commandes utiles.

1. A l'adresse https://fr.wikipedia.org/wiki/Fr%C3%A9quence_d%27apparition_des_lettres_en_fran%C3%A7ais

```
L = tri_dico(toutes_occurrences(chiffre))  # à refaire après chaque échange
chiffre = remplace(chiffre, ''p'', '' '') # changer le caractère
affiche_texte(chiffre)  # affiche les 150 premiers caractères
# ou affiche_texte(chiffre, 250) pour les 250 premiers caractères.
```

III Pour aller plus loin

Une autre manière classique d'encoder un texte est d'utiliser le codage de Vigénère. Ce codage n'est pas sensible au décryptage par analyse de fréquence.
Le principe de codage est le suivant :

- On dispose d'une clé secrète sous forme d'une chaîne de caractère (courte).
- Chaque caractère de la clé représente un décalage dans l'alphabet. La valeur du décalage sera donnée par la position dans la table de caractère (que l'on obtient par la fonction `ord` qui prend comme argument un caractère et retourne l'entier qui est son indice dans la table de caractères).
- Chaque caractère du texte sera chiffré en le décalant dans la table de caractère (par exemple un "a" décalé de 3 deviendra un "d"). La valeur du décalage est obtenue pour chaque caractère du texte en répétant la clé autant de fois qu'il le faut.
  Par exemple, pour une clé de longueur 3, les trois premiers caractères du texte à chiffré seront décalés par les 3 caractères de la clé (dans l'ordre), puis le quatrième caractère sera décalé en utilisant le premier caractère de la clé et ainsi de suite.
- Si on sort de la table de caractère (par un décalage positif ou négatif), on considère que la table est périodique (le caractère précédent le premier est le dernier, le caractère suivant le dernier est le premier). On utilise pour représenter ceci l'opération de modulo (reste dans la division euclidienne).

Exercice 4
Expliquer pourquoi une analyse simple des fréquence ne peut pas être efficace dans ce cadre.

Exercice 5
Créer les fonction `codage_vigenere` et `decodage_vigenere`. On pourra utilise la fonction `chr` qui est la réciproque de `ord`.