Dictionnaires TP 1

Thème types construits

Exercice 1 ***Préparation***

1. Récupérer l'archive `materiel-TP1-dictionnaires.zip`, la coller dans un dossier pertinent de son espace personnel et la déballer.
2. Ouvrir le script `TP1-dictionnaires-eleves.py` dans un IDE `Python`. Tous les codes produits pour ce TP seront ajoutés à ce script.

1 Top 50 des unigrammes dans un roman

Méthode ***Dictionnaires***

Un **p-uplet** de type `tuple` est pratique si on veut indexer une collection de valeurs par des entiers, un dictionnaire permet de couvrir les situations où l'on souhaite indexer une collection de valeurs par des valeurs non entières, par exemple par des chaînes de caractères.

Instruction ou expression	Rôle
`dicoc = dict()` ou `dico = {}`	crée un dictionnaire vide
`dico['Paul'] = 18`	associe la valeur 18 à la clef 'Paul' dans le dictionnaire
`dico.keys()`	clefs du dictionnaire (itérable avec une boucle `for`)
`dico.items()`	couples (clef, valeur) du dictionnaire (itérable avec une boucle `for`)

Au contraire des éléments d'une liste ordonnés par leur index, les éléments d'un dictionnaire ne sont pas ordonnés et on ne peut pas prévoir l'ordre d'apparition lorsqu'on itère avec une boucle `for` sur un dictionnaire.

```
>>> note = {}
>>> note['Knuth'] = 18
>>> note['Euclide'] = 20
>>> for (clef, val) in note.items():
...     print('clef = ', clef, '|', 'val = ', val)
...
clef = Knuth | val = 18
clef = Euclide | val = 20
```

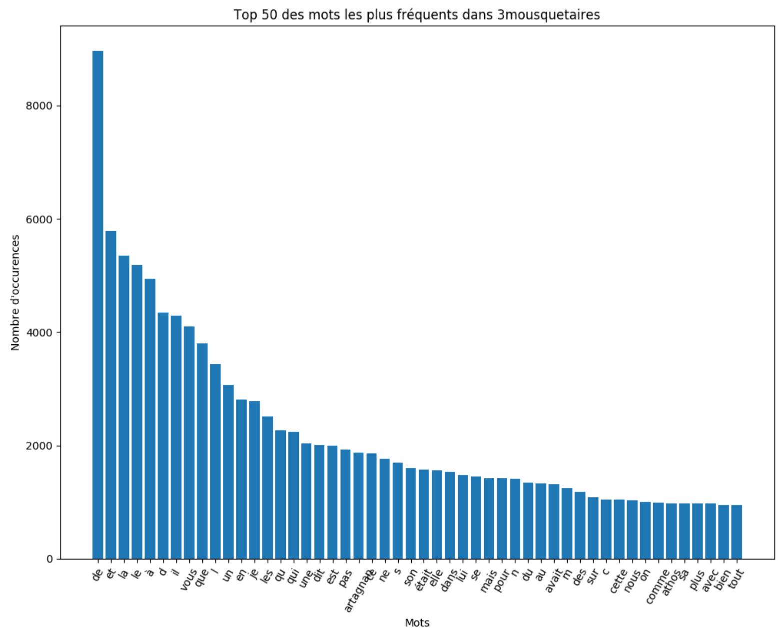
Pour ordonner un dictionnaire `dico`, il faut tout d'abord le transformer en tableau de type `list` en appliquant `list` à `dico.items()` (et non pas à `dico`). Ensuite on applique la fonction `sorted` avec l'option `key` pour indiquer la fonction de tri (ordre alphabétique par clef puis valeur par défaut, ou uniquement par clef ou valeur ...), l'option facultative `reverse` précisant si on veut un ordre décroissant.

```
>>> tab_note = list(note) #ne convertit que les clefs en tableau
>>> tab_note
['Knuth', 'Euclide']
>>> tab_note = list(note.items()) #convertit les paires (clef, valeur)
    en tableau
>>> tab_note
[('Knuth', 18), ('Euclide', 20)]
>>> sorted(tab_note)
[('Euclide', 20), ('Knuth', 18)]
>>> def tri_valeur(paire_clef_valeur):
...     clef, valeur = paire_clef_valeur
...     return valeur
...
>>> sorted(tab_note, key = tri_valeur)
[('Knuth', 18), ('Euclide', 20)]
>>> sorted(tab_note, key = tri_valeur, reverse = True)
[('Euclide', 20), ('Knuth', 18)]
```

Exercice 2

Un **unigramme** est normalement un caractère dans un texte, voir https://fr.wikipedia.org/wiki/N-gramme, néanmoins nous allons prendre la définition prise par l'outil `Google BooksNgram Viewer`, pour lequel un **unigramme** est un mot. L'outil disponible sur https://books.google.com/ngrams/ permet de visualiser la fréquence d'apparition de **n-grammes**, c'est-à-dire de séquences de n mots dans les corpus de textes de Google Books. On donne ci-dessous la visualisation des fréquences des mots *tuberculose* et *sida* dans le corpus de textes français parus entre les années 1800 et 2000.

On souhaite réaliser le graphique ci-dessous représentant le Top 50 des unigrammes dans le roman « Les trois mousquetaires » d'Alexandre Dumas. Les fichiers `3mousquetaires.txt` et `ducotedechezswann.txt` se trouvent dans l'archive fournie. Il s'agit de textes dans le domaine public, téléchargés sur le site du projet Gutenberg http://www.gutenberg.org/.



1. La première étape consiste à récupérer dans un tableau tous les mots du fichier. Pour distinguer les mots, on nettoie le texte en remplaçant tous les symboles de ponctuation et tous les caractères de mise en forme (espace, saut de ligne ...), par un espace simple. On transforme aussi en minuscule tous les caractères pour ne pas tenir compte de la casse. Ainsi on transforme le texte en une longue chaîne de mots séparés par un ou plusieurs espaces. La fonction `nettoyer_fichier(fichier)`, fournie dans `TP1-dictionnaires-eleves.py`, effectue ce travail à l'aide d'expressions régulières : le module re est importé au début du script.

   Écrire une fonction `extraire_tab_mot(source)` qui prend en paramètre un fichier texte, qui le nettoie avec la fonction `nettoyer_fichier` et qui renvoie un tableau de tous les mots.

   ⚠ Le tableau renvoyé ne doit pas contenir de chaîne vide '', on pourra consulter la documentation de la méthode `split` des chaînes de caractères avec `help(str.split)`.

   L'assertion ci-dessous doit être vérifiée :

```
tab_mots_mousquetaires = extraire_tab_mot('3mousquetaires.txt')
assert (len(tab_mots_mousquetaires), tab_mots_mousquetaires[:3]) ==
    (239574, ['introduction', 'il', 'y'])
```

2. Compléter le code de la fonction `histogramme(tab)` qui prend en paramètre un tableau de mots et qui renvoie un dictionnaire représentant l'histogramme de cette distribution de mots.

```
>>> tab_mots = ['un','','ami', 'proche','un']
>>> histogramme(tab_mots)
{'ami': 1, 'proche': 1, 'un': 2}
```

```
def histogramme(L):
    histo = {}
    for mot in L:
        if mot not in histo:
            #à  compléter
```

```
        else:
            #à  compléter
    return histo
```

L'assertion suivante doit être vérifiée :

```
assert(histo_mousquetaires['milady'] == 710 and histo_mousquetaires[
    'cardinal'] == 551)
```

3. Écrire la fonction dont on donne le prototype :

```
def plus_frequent_mot(histo):
    """Prend en paramètre :
    - un dictionnaire histo
    représentant les nombres d'occurences de mots dans un texte
    Renvoie un tuple constitué du plus grand nombre d'occurences
    et du tableau de tous les mots atteignant ce maximum"""
```

L'assertion suivante doit être vérifiée :

```
assert(histo_mousquetaires['milady'] == 710 and histo_mousquetaires[
    'cardinal'] == 551)
```

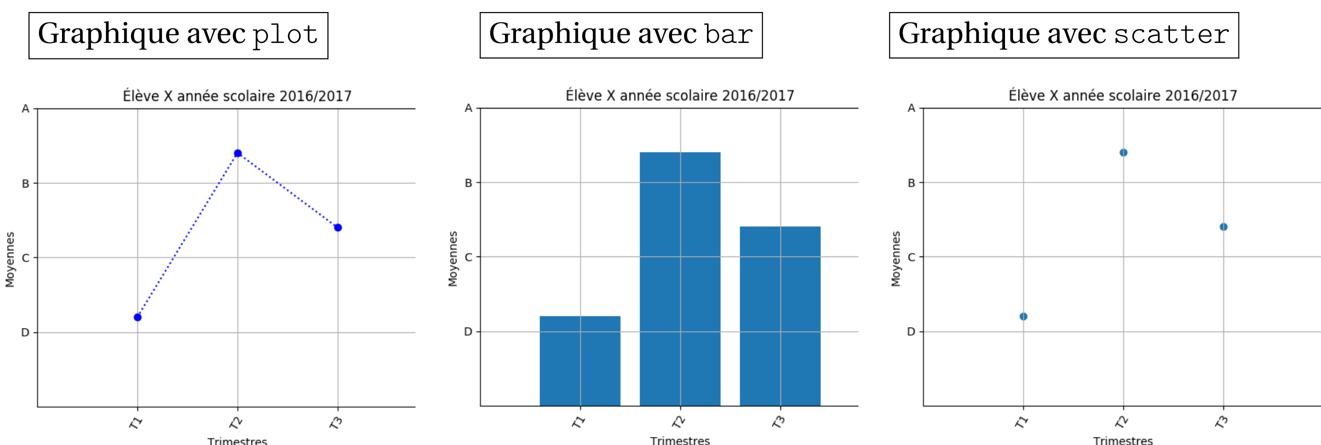
4. À l'aide des informations du bloc méthode sur le tri de dictionnaire et des fonctions précédentes, écrire une fonction `top_unigramme(source, but)` qui renvoie un tableau de paires `(mot, effectif)` du plus fréquent au moins fréquent dans le fichier texte `source`. Cette fonction doit aussi écrire un couple par ligne sous la forme `'de,8964\n'` dans le fichier texte `but`.

Les assertions suivantes doivent être vérifiées :

```
top_mousquetaires = top_unigramme('3mousquetaires.txt', 'top-
    mousquetaires.csv')
assert(top_mousquetaires[:4] == [('de', 8964), ('et', 5792), ('la',
    5357), ('le', 5191)])
top_swann = top_unigramme('ducotedechezswann.txt', 'top-swann.csv')
assert(top_swann[:4] == [('de', 7794), ('la', 3937), ('et', 3898), (
    'à ', 3829)])
```

Méthode *Graphiques avec matplotlib*

La bibliothèque / module `matplotlib` permet de réaliser des graphiques de qualité. On importe usuellement son sous-ensemble `matplotlib.pyplot` sous l'alias `plt` avec `import matplotlib.pyplot as plt`.



Le code ci-dessous fourni dans la fonction `exemple_graphique` permet d'obtenir le graphique de gauche, ceux du centre et de droite s'obtiennent en remplaçant la ligne 24 respectivement par `plt.bar(x, y)` ou `plt.scatter(x, y)`.

```
#import de la bibliothèque graphique
import matplotlib.pyplot as plt
#graduations de l'axe des abscisses
plt.xticks(list(range(1, 4)), ['T1', 'T2', 'T3'], rotation = 60)
#légende de l'axe des abscisses
plt.xlabel('Trimestres')
#limites des abscisses
plt.xlim(0, 4)
#graduations de l'axe des ordonnées
plt.yticks([5,10,15,20], ['D', 'C', 'B', 'A'])
#légende de l'axe des ordonnées
plt.ylabel('Moyennes')
#limite des ordonnées
plt.ylim(0,20)
#affichage de la grille
plt.grid()
#Titre du graphique
plt.title('Élève X année scolaire 2016/2017')
#tableau des abscisses
x = list(range(1, 4))
#tableau des ordonnées
y = [6,17,12]
#graphique avec des points bleus reliés en pointillés
plt.plot(x, y,'bo:')
#affichage du graphique
plt.show()
#enregistrement sur disque du graphique
plt.savefig('eleve-X-2016-2017.png')
```

La bibliothèque `matplotlib` est immense, lorsqu'on veut réaliser un graphique, il est conseillé de parcourir d'abord la galerie d'exemples http://matplotlib.org/gallery.html ou de consulter un tuto-

riel comme celui de Nicolas Rougier http://www.labri.fr/perso/nrougier/teaching/matplotlib/. Ensuite la consultation de la documentation en ligne http://matplotlib.org/api/pyplot_summary.html est souvent indispensable.

Exercice 3

À l'aide de la fonction `top_unigramme` de l'exercice 2 et de l'exemple précédent, écrire une fonction, `diagramme_top50_unigramme(source)` qui génère le diagrammes en bâtons du Top 50 des unigrammes les plus fréquents dans le texte du fichier `source`.
Tester sur les fichiers `3mousquetaires.txt` et `ducotedechezswann.txt`.

2 Fréquences des digrammes et prédiction du mot suivant

Exercice 4

Un **digramme** est une série de 2 mots consécutifs. On souhaite prédire le mot suivant un mot fixé à partir de statistiques réalisées sur un roman. Pour cela, on va calculer pour chaque digramme du texte sa fréquence relative par rapport à l'ensemble des digrammes qui ont le même commencement. Par exemple, dans les « Trois mousquetaires », le digramme *chez milady* a une fréquence de $0,0319$ par rapport à l'ensemble des digrammes commençant par *chez*. Pour simplifier, on ne tient pas compte des symboles de ponctuation : dans *Il parle. Elle écoute*, on compte comme digramme *parle elle*.

1. On va commencer par découper le texte source en un tableau de mots séparés par des espaces avec la fonction `extraire_tab_mot` de l'exercice 2, puis on va parcourir ce tableau en construisant un dictionnaire `prochain` tel que par exemple `prochain['de']` est lui-même un dictionnaire dont les clefs sont les mots suivants `'de'` et les valeurs le nombre de fois où le digramme ainsi constitué se rencontre dans le texte.

Écrire une fonction `prochain_histo(source)` qui prend en paramètre un fichier texte `source` et qui renvoie le dictionnaire `prochain` décrit ci-dessus.

Les assertions suivantes doivent être vérifiées :

```
assert prochain_mousquetaires['pose'] == {'qui': 2, 'gracieuse': 1,
    'et': 1}
assert prochain_mousquetaires['de']['buckingham'] == 59
```

2. Écrire une fonction `somme_histo(histo)` qui renvoie la somme des effectifs d'un histogramme `histo` qui est un dictionnaire de couples `(mot, effectif)`.

```
In [92]: somme_histo({'et': 1, 'gracieuse': 1, 'qui': 2})
Out[92]: 4
```

3. Recopier et compléter le code de la fonction `prochain_freq(source)` ci-dessous qui prend en paramètre un fichier texte `source` et qui renvoie un dictionnaire de dictionnaires `freq`.

`freq['pose']['gracieuse']` vaut 0.25 car c'est la fréquence de digrammes `'pose gracieuse'` parmi les digrammes commençant par `'pose'`.

```
def prochain_freq(source):
    prochain = prochain_histo(source)
    freq_digramme = dict()
    for mot in prochain:
        freq_digramme[mot] = dict()
        histo = prochain[mot]
        #à  compléter
    return freq_digramme
```

Les assertions suivantes doivent être vérifiées :

```
freq_digramme_mousquetaire = prochain_freq('3mousquetaires.txt')
assert freq_digramme_mousquetaire['pose']['gracieuse'] == 0.25
assert freq_digramme_mousquetaire['pose']['et'] == 0.25
assert freq_digramme_mousquetaire['pose']['qui'] == 0.5
```

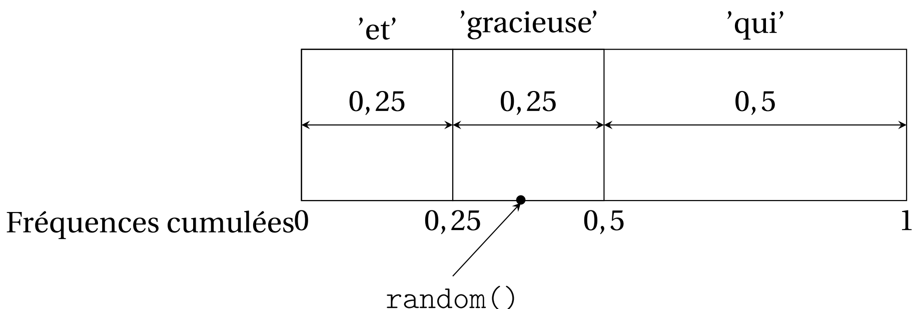
Exercice 5

Que peut-on faire avec le dictionnaire `freq_digramme` obtenu dans l'exercice précédent ?

```
In [42]: freq_digramme_mousquetaire['pose']
Out[42]: {'et': 0.25, 'gracieuse': 0.25, 'qui': 0.5}
```

- Si cette distribution de fréquences était réalisée sur un large corpus de textes d'Alexandre Dumas, on pourrait l'utiliser comme signature de l'auteur pour authentifier un texte apocryphe.
- Dans le corpus constitué par le texte « Les trois mousquetaires », le mot le plus probable après `'pose'` est `'qui'`. Si le corpus recouvrait un ensemble conséquent de textes de langue française, on pourrait utiliser ce résultat dans un logiciel de suggestion de saisie, sur un smartphone par exemple.
- Enfin, on peut jouer avec cette distribution de fréquences, pour générer automatiquement un texte à la manière de Dumas : on écrit un premier mot par exemple `'la'` puis on choisit aléatoirement le mot suivant selon la distribution de fréquences `freq['la']`, on tire au sort par exemple `'ville'` puis on choisit aléatoirement le mot suivant selon la distribution de fréquences `freq['ville']` ... Mathématiquement, il s'agit d'une chaîne de Markov.

Étant donné la distribution `freq['pose']`, on est ramené au problème d'un tirage au sort du mot suivant dans une urne où la probabilité de sortie de `'et'` est $0,25$, celle de `'gracieuse'` est $0,25$ et celle de `'qui'` est $0,5$.
On dispose de la fonction `random` du module `random` qui renvoie un nombre flottant aléatoire dans l'intervalle $[0;1[$.
On peut alors imaginer le découpage de l'intervalle $[0;1[$ en trois sous-intervalles d'amplitudes $0,25$, $0,25$ et $0,5$ associés respectivement à `'et'`, `'gracieuse'`, `'qui'`.
Il suffit de savoir déterminer à quel sous-intervalle appartient le nombre aléatoire `random()` et pour cela on utilise les fréquences cumulées : la plus petite fréquence cumulée supérieure à ce nombre déterminer le sous-intervalle et donc le mot qu'on a tiré au sort. Dans l'exemple, ci-dessous `random()` renvoie `0.36`, la plus petite fréquence cumulée supérieure est $0,5$ et le mot suivant choisi est `'gracieuse'`.



1. Voici comment construire un tableau de fréquences cumulées à partir d'un tableau de fréquences :

```
>>> freq = [0.2, 0.1 ,0.7]
>>> cumul = [freq[0]]
>>> cumul[len(cumul)-1]
0.2
>>> cumul.append(cumul[len(cumul)-1] + freq[1])
>>> cumul
[0.2, 0.30000000000000004]
>>> cumul.append(cumul[len(cumul)-1] + freq[2])
>>> cumul
>>> [0.2, 0.30000000000000004, 1.0]
```

En pratique, on veut plutôt construire un tableau de paires `(mot, fréquence cumulée)`.

Écrire une fonction `histo_cumul(histo)` qui prend en paramètre un histogramme de fréquences sous forme de dictionnaire comme `freq_digramme['pose']` et qui renvoie un tableau de paires `(mot, fréquence cumulée)`.

⚠ L'ordre de parcours d'un dictionnaire n'étant pas prévisible, il est difficile d'écrire une assertion mais on peut vérifier qu'à partir des fréquences cumulées on retrouve la bonne distribution de fréquences :

```
>>> histo_cumul(freq_digramme_mousquetaire['pose'])
[('qui', 0.5), ('gracieuse', 0.75), ('et', 1.0)]
```

2. Écrire une fonction `mot_suivant(mot, freq)` dont voici le prototype :

```
def mot_suivant(freq_second):
    """Prend en paramètre :
    - freq_second un dictionnaire de paires (m : int, f : float)
    représentant la distribution de fréquences de seconds mots
    parmi les digrammes commençant par un certain mot
    """
```

On donne des exemples d'évaluation :

```
>>> mot_suivant(freq_digramme_mousquetaire['pose'])
'et'
>>> mot_suivant(freq_digramme_mousquetaire['pose'])
'qui'
```

3. Écrire une fonction dont on donne la spécification ci-dessous :

```
def frequence_echantillon_mot_suivant(taille:int, premier_mot:str,
     freq:Dict[str,Dict[str,float]])->Dict[str, float]:
    """Paramètres:
    - taille de type int représentant la taille de l'échantillon
    - premier_mot de type str
    - freq un dictionnaire de dictionnaires tel que freq[premier_mot
        ] est la distribution de fréquences
    des seconds mots dans les digrammes commençant par premier_mot
    Précondition : 0 <= taille et len(freq) > à et premier_mot in
        freq
    Valeure renvoyée: un dictionnaire
    Postcondition: renvoie un dictionnaire représentant la
        distribution de fréquences
    des seconds mots parmi les digrammes commençant par premier_mot
    sur un échantillon de taille passée en paramètre"""
```

`frequence_echantillon_mot_suivant(taille, premier_mot, freq)` qui renvoie un dictionnaire représentant la distribution expérimentale de seconds mots sur un échantillon de `taille` digrammes aléatoire commençant par `premier_mot` avec `freq` le dictionnaire de dictionnaire renvoyé par la fonction `prochain_freq` de l'exercice 4. Voici un exemple d'exécution :

```
>>> frequence_echantillon_mot_suivant(1000, 'pose',
     freq_digramme_mousquetaire)
{'gracieuse': 0.246, 'qui': 0.519, 'et': 0.235}
>>> frequence_echantillon_mot_suivant(100000, 'pose',
     freq_digramme_mousquetaire)
{'qui': 0.50156, 'gracieuse': 0.25048, 'et': 0.24796}
```

4. Recopier et compléter le code de la fonction `autoTexte` dont on donne la spécification ci-dessous.

```
def auto_texte(chemin_fichier:str, chemin_sortie:str, premier_mot:
     str, taille:int)->None:
    """Paramètres :
    - deux chemins de fichiers de type str
    - premier_mot de type str un premier mot
    - taille de type int
    Précondition : chemins non vides et premier_mot dans
        prochain_fre(chemin_fichier) et taille >= 0
    Valeur renvoyée : None
    Postcondition : écrit dans un fichier texte, un texte généré alé
        atoirement à partir des distributions
    de fréquences des seconds mots pour les digrammes du texte de
        source"""
    assert chemin_fichier != '' and chemin_sortie != '' #pré
        condition 1
    freq = prochain_freq(chemin_fichier)
    assert premier_mot in freq and taille >= 0 #précondition 2
    #ouverture du fichier de sortie
```

```
    f = open(chemin_sortie, 'w')
    mot = premier_mot
    f.write(mot + ' ')
    taille_ligne = len(mot) + 1
    #à  compléter
    f.close()
```

On donne un exemple de sortie :

```
le moment où il avait plus sévèrement pour le bâtiment
avait accompagné d artagnan rien qu elle avait si son
ambition voulût le silence le rappelle dit athos déclara à
terre l atteste le comte et pour aller plus dévoués pour un
grand nombre suffisant vous faites frémir s arrêtaient
devant une grande jeunesse et à cette fois par votre
```