Exercice 1 *Préparation*

1. Copier le sous-dossier `Activite3` du dossier `ICN\Texte` situé dans le partage `Donnees` de sa classe. Coller ce dossier dans son espace personnel.
2. `Activite3` contient un sous-dossier `ressources` avec les fichiers textes que nous allons manipuler. Ouvrir `Pyzo`, enregistrer dans `ressources` un fichier `Texte-Activite3.py`, puis exécuter ce fichier avec l'option `Run as script`.

1 Top 50 des unigrammes dans un roman

Bloc-Note 1 *Dictionnaires*

Une liste 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.

Fonctions	Rôle
`dico = dict() ou`	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.

```
In [13]: note = dict()

In [14]: note['Paul'] = 10

In [15]: note['Marc'] = 12

In [16]: for clef, valeur in note.items():
    ...:     print(clef, valeur)
    ...:
Marc 12
Paul 10
```

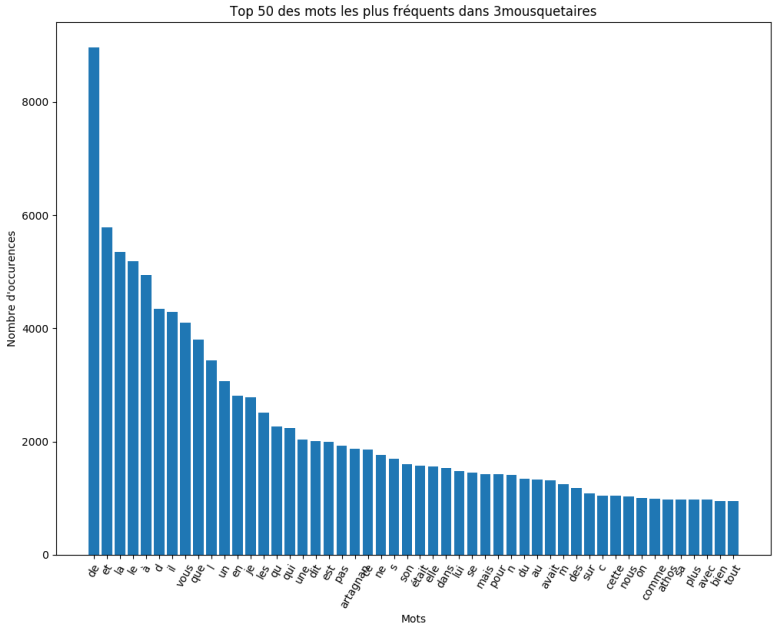
Pour ordonner un dictionnaire `dico`, il faut tout d'abord le transformer en liste en appliquant `list` à `dico.items()` (et non pas à `dico`). Ensuite on applique la fonction `sorted` avec l'option `key` pour indiquer si on classe selon les clefs ou les valeurs, l'option `reverse` précisant éventuellement si on veut un ordre décroissant.

```
In [19]: listnote = list(note.items())

In [20]: listnote
Out[20]: [('Marc', 12), ('Paul', 10)]

In [21]: sorted(listnote, key = lambda t : t[0])
Out[21]: [('Marc', 12), ('Paul', 10)]        #ordre alphabétique croissant

In [22]: sorted(listnote, key = lambda t : t[1]) #ordre des notes croissant
Out[22]: [('Paul', 10), ('Marc', 12)]

In [23]: sorted(listnote, key = lambda t : t[1], reverse = True)
Out[23]: [('Marc', 12), ('Paul', 10)]       #ordre des notes décroissant
```

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 fichier `3mousquetaires.txt` et `ducotedechezswann.txt` se trouvent dans le dossier `ressources`. 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 une liste tous les mots du fichier. Pour distinguer les mots, on nettoie le fichier 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 `nettoyerFichier(fichier)` effectue ce travail à l'aide d'expressions régulières (il faut insérer `import re` au début de votre script). Elle se trouve dans le fichier `cadeau.py`.

a. Décrire en quelques lignes l'utilisation d'expressions régulières en informatique.

Ressources :

https://fr.wikipedia.org/wiki/Expression_régulière

https://www.lucaswillems.com/fr/articles/25/tutoriel-pour-maitriser-les-expressions-regul

b. Écrire une fonction `listmot(source)` qui prend en paramètre un fichier texte, qui le nettoie avec la fonction `nettoyerFichier` et qui retourne la liste des mots (éventuellement des mots vides '') du texte.

```
In [30]: L = listmot('3mousquetaires.txt')

In [31]: len(L)
Out[31]: 239576
```

2. Compléter le code de la fonction `histogramme(L)` qui prend en paramètres une liste de mots et qui retourne un dictionnaire représentant l'histogramme de cette distribution de mots.

```
In [34]: L = ['un','','ami', 'proche','un']

In [35]: histogramme(L)
Out[35]: {'ami': 1, 'proche': 1, 'un': 2}
```

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

3. En s'inspirant du bloc-note sur les dictionnaires, compléter la troisième instruction ci-dessous pour récupérer dans `classement` une liste de couples `(mot, effectif)` ordonnée dans l'ordre décroissant des effectifs.

```
In [36]: L = listmot('3mousquetaires.txt')

In [37]: histo = histogramme(L)

In [38]: classement = sorted(.....................)
```

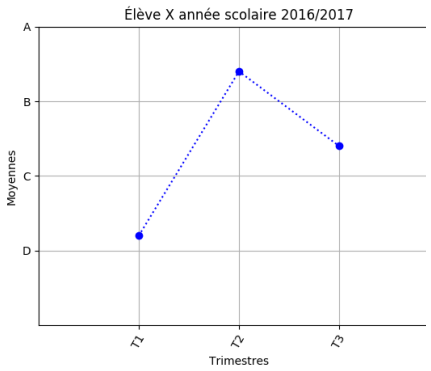
4. Compiler les questions précédentes pour écrire une fonction `topUnigramme(source, but)` qui retourne une liste des couples `(mot, effectif)` du plus fréquent au moins fréquent dans le fichier texte `source`. Cette fonction doit aussi écrire un couple par ligne sous la forme `'de 8964\n'` dans le fichier texte `but`.

```
In [44]: T = topUnigramme('3mousquetaires.txt','top-3mousquetaires.txt')

In [45]: T[:2]
Out[45]: [('de', 8964), ('et', 5792)]  #les deux premiers
```

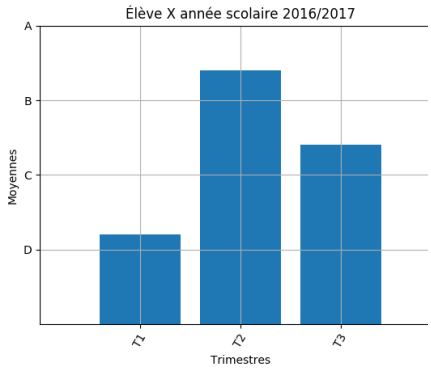
Bloc-Note 2 *Graphiques avec matplotlib*

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`.

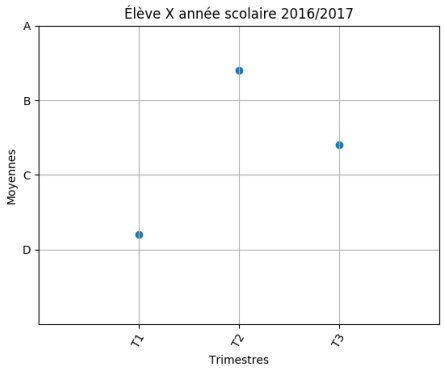
Graphique avec `plot`



Graphique avec `bar`



Graphique avec `scatter`



Le code ci-dessous a permis 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)`.

```
1  #import de la bibliothèque graphique
2  import matplotlib.pyplot as plt
3  #graduations de l'axe des abscisses
4  plt.xticks(list(range(1, 4)), ['T1', 'T2', 'T3'], rotation = 60)
5  #légende de l'axe des abscisses
6  plt.xlabel('Trimestres')
7  #limites des abscisses
8  plt.xlim(0, 4)
9  #graduations de l'axe des ordonnées
10 plt.yticks([5,10,15,20], ['D', 'C', 'B', 'A'])
11 #légende de l'axe des ordonnées
12 plt.ylabel('Moyennes')
13 #limite des ordonnées
14 plt.ylim(0,20)
15 #affichage de la grille
16 plt.grid()
17 #Titre du graphique
18 plt.title('Élève X année scolaire 2016/2017')
19 #liste d'abscisses
20 x = list(range(1, 4))
21 #liste d'ordonnées
22 y = [6,17,12]
23 #graphique avec des points bleus reliés en pointillés
24 plt.plot(x, y,'bo:')
25 #affichage du graphique
26 plt.show()
27 #enregistrement sur disque du graphique
28 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 tutoriel comme celui de Nico-

las 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

Exploiter la liste des couples `(mot, effectif)` retournée par l'appel de fonction `topUnigramme('3mousquetaires.txt', 'top3mousquetaires.txt')` pour réaliser le diagramme en bâtons du Top 50 des unigrammes les plus fréquents dans les « Trois mousquetaires », tel qu'il est présenté dans l'exercice 1. Réalisez un diagramme similaire pour le texte « Un amour de Swann » de Marcel Proust, le fichier `unamourdeswann.txt` se trouve dans `ressources`.

2 Fréquences des digrammes, prédiction du mot suivant et mélanges de texte

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 une liste de mots séparés par des espaces avec la fonction `listmot` de l'exercice 1, puis on va parcourir cette liste 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.

   Recopier et compléter le code de la fonction `prochainHisto(source)` qui prend en paramètre un fichier texte `source` et qui retourne le dictionnaire `prochain` décrit ci-dessus.

```
In [91]: prochain['pose']
Out[91]: {'et': 1, 'gracieuse': 1, 'qui': 2}
```

```
def prochainHisto(source):
    lesmots = listmot(source)
    prochain = dict()
    for k in range(nbmot - 1):
        mot = lesmots[k]
        suivant = lesmots[k + 1]
        if mot != '':
            if mot not in prochain:
                #à  completer
            else:
                #à  completer
    return prochain
```

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

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

3. Recopier et compléter le code de la fonction `prochainFreq(source)` ci-dessous qui prend en paramètre un fichier texte `source` et qui retourne 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'`.

On pourra vérifier ses résultats avec les fréquences contenues dans le fichier `'3mousquetaires-freq-digrammestxt'` dans `'ressources'`.

```
def prochainFreq(source):
    prochain = prochainHisto(source)
    freq = dict()
    for mot in prochain:
        freq[mot] = dict()
        histo = prochain[mot]
        #à  compléter
    return freq
```

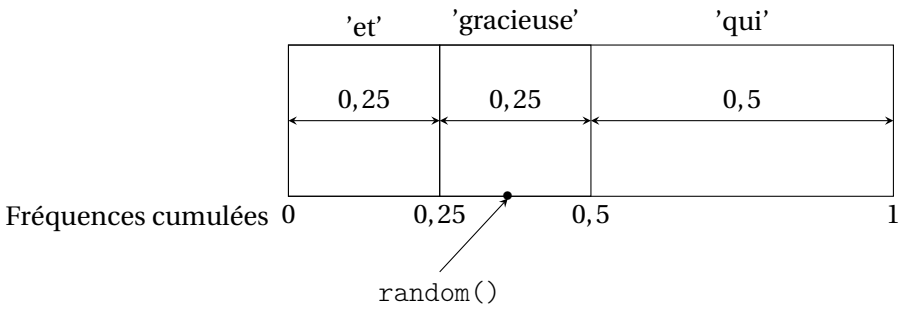
Exercice 5

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

```
In [42]: freq['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 retourne 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 celà 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()` retourne `0.36`, la plus petite fréquence cumulée supérieure est 0,5 et le mot suivant choisi est `'gracieuse'`.



1. Voici comment construire une liste de fréquences cumulées à partir d'une liste de fréquences :

```
In [6]: freq = [0.2, 0.1 ,0.7]

In [7]: cumul = [freq[0]]

In [8]: cumul[-1]
Out[8]: 0.2

In [9]: cumul.append(cumul[-1] + freq[1])

In [10]: cumul
Out[10]: [0.2, 0.30000000000000004]

In [11]: cumul.append(cumul[-1] + freq[2])

In [12]: cumul
Out[12]: [0.2, 0.30000000000000004, 1.0]
```

En pratique, on veut plutôt construire une liste de couples `(mot, fréquence cumulée)`.

Recopier et compléter le code de la fonction `histoCumul(histo)` qui prend en paramètre un histogramme de fréquences sous forme de dictionnaire comme `freq['pose']` et qui retourne une liste de fréquences cumulées. On commence par convertir en liste le dictionnaire `histo`.

```
def histoCumul(histo):
    histo = [(mot, effectif) for mot, effectif in histo.items()]
    histoC = [histo[0]]
    for k in range(1, len(histo)):
        #à  compléter
    return histoC
```

```
In [7]: histoCumul(freq['pose'])
Out[7]: [('qui', 0.5), ('et', 0.75), ('gracieuse', 1.0)]
```

2. Écrire une fonction `motSuivant(mot, freq)` qui retourne le mot suivant le mot `mot` passé en paramètre, choisi aléatoirement selon la distribution de fréquences `freq[mot]`.

```
In [17]: motSuivant('pose', freq)
Out[17]: 'et'
```

3. Recopier et compléter le code de la fonction `autoTexte(source, debut, taille)` qui prend en paramètre un texte `source`, un premier mot `debut` et qui génère un texte automatique de `taille` mots selon la chaîne de Markov décrite précédemment. Tous les 20 mots, on saute une ligne.

```
def autoTexte(source, debut, taille):
    freq = prochainFreq(source)
    racine, ext = source.split('.')
    sortie = racine + '-auto-' + '.' + ext
    f = open(sortie, 'w')
    mot = debut
    f.write(mot + ' ')
    nbmotligne = 0
    for k in range(taille - 1):
        #à  compléter
```

```
        if nbmotligne >= 80:
            f.write('\n')
            nbmotligne = 0
    f.close()
```

4. Écrire une fonction `texteMelange(source1, source2, debut, taille)` qui prend en paramètre deux textes `source1` et `source2`, un premier mot `debut` et qui génère un texte automatique de `taille` mots selon une chaîne de Markov en alternant le choix du mot suivant selon les distributions de fréquences fournies par `source1` ou `source2`. Parfois le mot courant ne figure pas dans l'autre texte et on rompt alors l'alternance.

   On pourra faire des tests avec les textes `'3mousquetaires.txt'` et `'unamourdeswann.txt'`.

3 Enjeux sociétaux et droit du numérique

Exercice 6 *Sujets d'exposés*

Par groupe de trois, traiter l'un des sujets suivants sous forme de présentation numérique (diaporama Impress, page web HTML, prezi ...) que vous devrez présenter en dix minutes devant la classe au cours du premier trimestre.
Pour la réalisation de diaporama avec Impress, la ressource suivante est très bien : https://dane.ac-lyon.fr/spip/Video-Prise-en-main-d-Impress.

- Distance intertextuelle et authentification de textes : le cas de Corneille et Molière.

  Ressources : http://images.math.cnrs.fr/La-classification-des-textes.html

- Prédictions des algorithmes de saisie semi-automatique (comme Google Suggest),un avantage et des problèmes ?

  Ressources :

  https://support.google.com/websearch/answer/106230?hl=fr

  https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4982736/

  http://www.numerama.com/tech/215133-google-nettoie-ses-suggestions-de-recherche-pour-les-renc
  html

  http://www.slate.fr/story/30045/google-suggest-google-instant-racisme