

# Sujet 5 : Analyse des dialogues dans l'Avare de Molière

## Rappel du contexte du sujet :

L'Observatoire de la vie littéraire ([OBVIL \(http://obvil.sorbonne-universite.site/obvil/presentation\)](http://obvil.sorbonne-universite.site/obvil/presentation)) promeut une approche de l'analyse des textes littéraires fondée sur le numérique. Dans le cadre du [Projet Molière \(http://obvil.sorbonne-universite.site/projets/projet-moliere\)](http://obvil.sorbonne-universite.site/projets/projet-moliere), des pièces de cet auteur ont été numérisées et sont accessibles librement dans différents formats utilisables par un programme informatique.

Grâce à ces numérisations, il est possible d'écrire des programmes pour réaliser des analyses syntaxiques et sémantiques. Ce sujet se propose de reproduire une étude réalisée par l'OBVIL sur les dialogues de l'Avare de Molière.

## Rappel des objectifs de ce sujet :

1. Classez les personnages selon la quantité de parole grâce à une analyse syntaxique du texte (scènes / répliques / mots). En particulier, quel est celui qui parle le plus ? Quel est celui qui ne parle pas du tout ? Attention, les noms des personnages ne sont pas forcément homogènes (casse et accents par exemple).
2. Réalisez un graphique qui montrera le nombre de mots que chaque acteur prononce dans chaque scène. Pour cela, vous pouvez vous inspirer de l'[étude de l'Avare de Molière réalisée par l'OBVIL \(https://obvil.sorbonne-universite.fr/corpus/moliere/moliere\\_avare\)](https://obvil.sorbonne-universite.fr/corpus/moliere/moliere_avare) (graphe de gauche). Dans ce graphique, les lignes sont de longueur égale et la hauteur représente le nombre de mots prononcés au total dans la scène. La largeur de chaque rectangle indique le pourcentage de la scène qu'un acteur occupe.
3. Facultatif : Construisez un graphe d'interlocution permettant de visualiser les échanges entre les personnages. Pour cela, vous pouvez vous inspirer de l'[étude de l'Avare de Molière réalisée par l'OBVIL \(https://obvil.sorbonne-universite.fr/corpus/moliere/moliere\\_avare\)](https://obvil.sorbonne-universite.fr/corpus/moliere/moliere_avare) (graphe de droite).

La version numérisée que l'on se propose d'utiliser est le fichier texte au format markdown disponible ici [moliere\\_avare \(http://dramacode.github.io/markdown/moliere\\_avare.txt\)](http://dramacode.github.io/markdown/moliere_avare.txt).

```
In [1]: data_url = "http://dramacode.github.io/markdown/moliere_avare.txt"
```

On va s'assurer qu'un fichier texte en local au format markdown contienne la pièce. Si le fichier "moliere\_avare.md" existe on considère que c'est bon et s'il n'existe pas nous allons télécharger le contenu disponible à l'URL renseignée ci-dessus et l'écrire dans ce fichier local "moliere\_avare.md".

```
In [2]: from os import path as path
import requests

local_filename = "moliere_avare.md"
# Si le fichier csv des données d'incidence existe en local
# il n'est pas nécessaire de le télécharger par l'URL
if not path.exists(local_filename):
    print("Le fichier local contenant la pièce de théâtre n'existe pas.")
    # Si le fichier n'existe pas en local dans le dossier courant
    # nous téléchargeons les données et les écrivons
    # dans un fichier en local
    # Téléchargement des données
    response = requests.get(data_url)
    # Ecriture des données téléchargées dans le fichier local
    with open(local_filename, "wb") as f:
        f.write(response.content)
```

Maintenant que nous sommes assurés d'avoir un fichier en local contenant le texte de l'Avare dont on va faire l'analyse, on va donc l'ouvrir, parcourir son contenu et le traiter au fur et à mesure.

Ce que l'on sait déjà c'est que l'on va devoir créer une structure de données pour l'analyse. On va passer par la bibliothèque Pandas et la création d'un dataframe, permettant de différencier les différents personnages et de qualifier leur "activité" au travers des différents actes et scènes. Néanmoins, après avoir parcouru le web, il est recommandé de passer par une structure intermédiaire pour la création du dataframe pandas. Nous allons choisir la structure de données native de python des dictionnaires en tant que structure de données intermédiaire.

Il y a plusieurs choses auxquelles il est déjà nécessaire de penser vis-à-vis de la problématique posée et des représentations graphiques demandées, notamment celle relative à la question facultative.

Pour commencer, afin d'obtenir une bonne lisibilité des graphiques, il sera intéressant d'associer à chaque personnage une couleur différente.

Ensuite, la question facultative demande de déterminer à qui s'adresse chaque réplique afin d'avoir le graphe directionnel des interactions entre les personnages de la pièce. La structure de données devra donc permettre de savoir pour chaque réplique l'auteur mais aussi le destinataire de cette dernière. Ce sont des informations assez simples à obtenir mais à prendre en compte dans la manière de "parser" le fichier texte.

```
In [3]: # On va d'ores et déjà utiliser une instruction afin que les graphiques s'affichent directement au sein du notebook
%matplotlib inline
```

```
In [4]: # On déclare l'utilisation de la bibliothèque pandas et on crée également l
e dictionnaire
# qui va nous servir d'intermédiaire avant la création du dataframe pandas
import pandas as pd
# Le dictionnaire établit une structure en tableau à 5 colonnes permettant
d'enregistrer
# l'auteur, le destinataire, l'acte, la scène, le contenu de la réplique, a
insi que sa longueur en termes de mots
# pour chaque réplique de la pièce
avareAnalysisDict = {'author':[], 'recipient':[], 'act':[], 'scene':[], 'speec
h':[], 'speech_length':[]}
textDataSynthesisTableDf = None

# Nous créons également d'ores et déjà un dictionnaire des personnages de l
a scène
# permettant d'enregistrer les informations de liens avec les autres person
nages
# de définir une couleur de représentation.

# Ce dictionnaire est initialisé vide car il sera rempli en utilisant le no
m de chaque personnage
# comme clés associées à des valeurs qui seront des dictionnaires à deux en
trées
# 'links' donnant les liens avec les autres personnages sous la forme d'une
liste
# 'color' permettant de régler une couleur de représentation
avarePersoDict = {}
```

Afin de parser le fichier en s'appuyant notamment sur les symboles de titres utilisés par le format Markdown, il nous faut avoir recours à l'utilisation d'un outil d'analyse des expressions régulières (cf. [Wikipedia Expression régulière](https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re) ([https://fr.wikipedia.org/wiki/Expression\\_r%C3%A9guli%C3%A8re](https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re))). La bibliothèque [re](https://docs.python.org/3/library/re.html) (<https://docs.python.org/3/library/re.html>) disponible nativement dans python permet de faire ce travail. Un rapide parcours du fichier montre que:

- les actes sont indiqués par des titres header 2, par une ligne commençant par ##
- les scènes sont indiquées par des titres header 3, par une ligne commençant par ###
- que les personnages d'une scène sont donnés sur la ligne suivant l'indication de la scène
- que la ligne précédant chaque réplique contient le "nom" de son auteur en majuscule

Voici un extrait illustrant ces propos: \ " ## Acte Premier.

### Scène Première. \ Valère, Élise

VALÈRE .

*Hé quoi, charmante Élise, vous devenez mélancolique, après les obligeantes assurances que vous avez eu la bonté de me donner de votre foi ? Je vous vois soupirer, hélas, au milieu de ma joie ! Est-ce du regret, dites-moi, de m'avoir fait heureux ? et vous repentez-vous de cet engagement où mes feux ont pu vous contraindre ?*

ÉLISE .

*Non, Valère, je ne puis pas me repentir de tout ce que je fais pour vous. Je m'y sens entraîner par une trop douce puissance, et je n'ai pas même la force de souhaiter que les choses ne fussent pas. Mais, à vous dire vrai, le succès me donne de l'inquiétude ; et je crains fort de vous aimer un peu plus que je ne devrais. \ "*

*Nota Bene:* En continuant à travailler sur le sujet et en approfondissant le parcours du texte, on peut se rendre compte qu'il y a parfois quelques indications de mise en scène qui viennent perturber ce schéma. Différente options s'offrent à nous:

- modifier le texte afin d'utiliser les mécanismes du markdown et mettre ces indications en italiques ou en gras --> utilisations de balises permettant de reconnaître facilement ces éléments
- ou bien repérer les numéros de ces lignes car il n'y en pas beaucoup afin de les ignorer (il nous faudra alors un moyen de connaître dans les fonctions d'analyse le numéro de la ligne du fichier en cours d'analyse.)
- pour être plus générique, l'idéal serait d'avoir un traitement intelligent, un réseaux de neurones par exemple spécialisé dans l'analyse du langage pour identifier ces éléments de mis en scène.

De plus les lignes indiquant la fin des actes sont également à ignorer en tant que réplique. Il y a au moins une scène ne donnant pas la liste des personnages, Harpagon étant le seul protagoniste, il faudra prendre en compte ce cas. Pour finir, deux lignes du fichier texte posent encore un problème:

- une ligne du fichier texte possède à sa fin, une déclaration de fin d'acte, ce qui tranche avec le schéma global,
- la dernière ligne du texte comprend la balise indiquant la fin de la pièce.

On traitera ces deux cas particulier dans la fonction de comptage des mots d'une ligne de texte.

```
In [5]: # Déclaration des numéros de lignes à ignorer
playInstructionsLineNumbersList = [201, 204, 213, 251, 272, 384, 623, 782,
815, 1280, 1283,\
                                     1342, 1369, 1411, 1527, 1610, 1633, 166
5, 1866, 1874, 1970, \
                                     2299, 2319, 2327, 2357, 2521, 2524]
```

```
In [6]: import re
import copy
```

```
In [7]: # Chaîne de caractère indiquant le début d'analyse et de récupération des p
ersonnages
persoCaptureStartLine = "# ACTEURS."
# L'expression régulière suivante permet de valider qu'une chaîne de caract
ère
# contenue dans une ligne lue correspond à la ligne précédant la définition
# de la liste des personnages
m = re.search('(?!<=# )ACTEURS\.$', persoCaptureStartLine)
m.group(0)
```

```
Out[7]: 'ACTEURS.'
```

En effet, le résultats obtenus n'est pas un objet null, à savoir une valeur None en python

```
In [8]: # Exemple d'une ligne de la liste des personnages
persoLineExample = " - Harpagon, Père de Cléante et d'Élise, et Amoureux de
Mariane."
# L'expression régulière proposée est la suivante
# On cherche une chaîne de caractères commençant par une majuscule --> [A-Z
À-ÿ]{1}
# potentiellement accentuée et ensuite composées de lettres potentiellement
# accentuées elles aussi, contenant des espaces, des virgules, des apostrophes
--> [a-zA-ZÀ-ÿ\s,\']+'
# et précédée d'un espace suivi d'un tiret suivi d'un espace (ce n'est pas
le tiret du 6) --> (?<=\s-\s)
# , chaîne de caractères qui ne sera pas capturée
m1 = re.search('(?<=\s-\s)[A-ZÀ-ÿ]{1}[a-zA-ZÀ-ÿ\s,\']+',persoLineExample)
m1.group(0)
```

Out[8]: "Harpagon, Père de Cléante et d'Élise, et Amoureux de Mariane"

Voici un lien vers un site de test en ligne d'expressions régulières python ([regexp\\_test \(https://pythex.org/\)](https://pythex.org/)) qui a aidé à mettre en place l'expression régulière précédente.

```

In [9]: # Fonction qui permet d'extraire la liste des personnages
# de la stocker dans un dictionnaire en parcourant ligne par ligne
# un fichier texte passé en entrée.
def fill_perso_dict(fileToAnalyse, emptyPersoDict):
    """
        Fonction remplissant un dictionnaire d'entrée avec la liste
        des personnages de la pièce de théâtre lue dans le fichier
        passé en entrée. On considère que le fichier en entrée est lu
        à partir du début, de sa première ligne.

        :param fileToAnalyse: pointeur obtenu par ouverture d'un fichier
        :param emptyPersoDict: dictionnaire à remplir par la fonction
        :return: numéro de la dernière ligne lue
    """
    lineNum = 0
    currentLine = fileToAnalyse.readline()
    lineNum += 1
    isStartPersoListLine = False
    while (currentLine and not isStartPersoListLine):
        m = re.search('(?!<=# )ACTEURS\.$', currentLine)
        if m is not None:
            isStartPersoListLine = True
            currentLine = fileToAnalyse.readline()
            lineNum += 1

    # La lecture s'est arrêtée car la ligne de début de définition de la list
    # e des personnages a été rencontrée.
    # Nous avons néanmoins lu la ligne suivante qui est obligatoirement un pe
    # rsonnage.
    # Nous devons maintenant lire ligne par ligne, la liste des personnages a
    # u format suivant:
    # "- NomPersonnage, Lien, Lien, ..."
    # Ainsi dès que la ligne ne commence plus par un tiret nous pouvons arrê
    # ter la lecture et le remplissage
    # du dictionnaire.
    isAPersoLine = True
    # On extrait la chaîne de caractères qui nous intéresse
    m = re.search('(?!<=\\s-\\s)[A-ZÀ-ÿ]{1}[a-zA-ZÀ-ÿ\\s,\\']+', persoLineExample)
    extractedString = m.group(0)
    while (currentLine and isAPersoLine):
        # Traitement de la ligne courante qui est obligatoirement
        # une ligne listant un personnage de la pièce.
        # Comme le montre l'exemple au-dessus, le nom ainsi que
        # les différents types de liens sont séparés par une virgule.
        parts = extractedString.split(',')
        # La ligne au-dessus crée une liste dont chaque élément
        # est séparé par des virgules
        # typiquement "tata, titi, toto".split(',') --> ['tata', 'titi', 'tot
        o']

        # Création d'un dictionnaire vide temporaire pour enregistrer les liens
        # ainsi que la couleur à paramétrer
        persoCaracs = {"links": [], "color": None}

        # On va traiter la ligne particulière qui introduit le commissaire ains

```

```

i que son clerc
m = re.match("le commissaire", parts[0], re.IGNORECASE)
if not m:
    # Ce n'est pas la ligne du commissaire.
    # Le premier élément donne le nom du personnage, indice 0 de la liste
    # Les éléments suivants donnent les liens avec les autres personnages
    # On parcourt le reste de ces derniers
    for elt in parts[1:]:
        # En regardant cette partie dans le fichier, on voit que soit les p
        arties commencent
        # par un espace et une lettre, soit un espace et un "et" que l'on n
        e souhaite pas capturer
        # On vérifie si l'élément commence par " et "
        if elt.startswith(" et "):
            # L'élément commence bien par " et " on enlève cette partie
            currentLink = elt[4:]
        else:
            # Ce n'est pas le cas, on enlève juste l'espace
            currentLink = elt[1:]
        persoCaracs["links"].append(currentLink)
    # Fin de la boucle for

    # On enregistre le dictionnaire temporaire des caractéristiques
    # du personnage courant rempli, dans le dictionnaire
    # global des personnages
    emptyPersoDict[parts[0]] = persoCaracs
else:
    # C'est la ligne du commissaire, la seconde partie définit le personn
    age du clerc et commence par " et "
    emptyPersoDict[parts[0]] = persoCaracs
    # On fait une copie profonde pour ne pas que l'objet partage la même
    adresse mémoire
    clercCaracs = copy.deepcopy(persoCaracs)
    clercCaracs["links"].append("assistant du commissaire")
    emptyPersoDict[parts[1][4:]] = clercCaracs

currentLine = fileToAnalyse.readline()
lineNum += 1
m = re.search('(?!<=\\s-\\s)[A-ZÀ-ÿ]{1}[a-zA-ZÀ-ÿ\\s,\\']+', currentLine)
if m is not None:
    extractedString = m.group(0)
else:
    isAPersoLine = False

    # Fin de la boucle while

return lineNum

```



Définition de la fonction permettant de compter le nombre de mots d'une ligne constituant une réplique. Pour implémenter celle-ci nous nous sommes appuyés sur des informations glanées au travers de la consultation de plusieurs pages internet et notamment la suivante [geeksforgeeks\\_comptage\\_de\\_mots\\_dans\\_une\\_phrase](https://www.geeksforgeeks.org/python-program-to-count-words-in-a-sentence/) (<https://www.geeksforgeeks.org/python-program-to-count-words-in-a-sentence/>). Nous nous sommes également appuyés sur le site de test des expressions régulières et avons également fait des tests dans la ligne de commande python. Pour traiter plus facilement le comptage des mots, nous allons nous servir du module natif string de python et allons donc importer celui-ci, pour utiliser l'attribut punctuation.

```
In [10]: import string
```

```

In [11]: def count_words(speechLine):
        """
        Fonction qui compte le nombre de mots contenus dans une ligne de texte.

        :param speechLine: ligne de texte en entrée dont il faut compter le nombre de mots
        :return: nombre de mots contenu dans la chaîne de caractère analysée
        """
        # Création d'une variable temporaire contenant la chaîne de caractère à traiter
        strToProcess = speechLine
        # Définition de l'expression régulière permettant d'exclure la déclaration de fin d'acte
        # à la fin de la ligne dans le cas particulier, on cherche un enchaînement de n'importe
        # quels caractères suivi d'un texte de la forme "Fin du nième Acte"
        patternExcludingActEndDeclaration = re.compile('(.+)(?=Fin du [a-zA-ZÀ-ÿ]+ Acte)')
        # Définition de l'expression régulière permettant d'exclure la déclaration de fin de la pièce
        # sur le même principe que la précédente
        patternExcludingPieceEndDeclaration = re.compile('(.+)(?=< Fin >)' )
        # Réalisons le matching du premier pattern
        m = patternExcludingActEndDeclaration.match(speechLine)
        # Si le résultat n'est pas vide c'est que nous sommes dans le cas particulier
        # et qu'il faut travailler sur la chaîne résultat de la fonction match.
        if m:
            strToProcess = m.group(0)

        # Maintenant réalisons le matching du second pattern
        m1 = patternExcludingPieceEndDeclaration.match(speechLine)
        if m1:
            strToProcess = m1.group(0)
        # Maintenant, grâce au module string, contruisons une expression régulière qui va chercher toute suite
        # de plus de 1 caractère excluant les espaces et signes de ponctuations
        wordsPattern = re.compile("[^{}\s]+".format(string.punctuation))
        # Traitons la chaîne de caractères avec la fonction de regexp findall
        # qui va extraire toutes les occurrences du pattern recherché.
        # Cela va nous donner en sortie la liste des mots de la chaîne traitée
        # en considérant par exemple l'article "l'", la contraction du pronom je en "j'"
        # comme des mots.
        speechPartWordsList = re.findall(wordsPattern, strToProcess)
        # Le nombre de mots est donc la taille de cette liste,
        # retournons la en sortie.
        return len(speechPartWordsList)

```

Définition de la fonction principale d'analyse du fichier texte générant un tableau de données de synthèse. Pour la création d'un dataframe pandas à partir d'un dictionnaire cf. [DataFrame.from\\_dict \(https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.from\\_dict.html\)](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.from_dict.html).

```

In [12]: import pandas as pd
def generate_text_synthesis_data_table(fileToAnalyse, persoDict, emptyAnalysisDict, lineNum):
    """
        Fonction principale d'analyse du texte générant un dataframe pandas
        de synthèse des informations permettant l'analyse globale.

        :param fileToAnalyse: pointeur obtenu par ouverture d'un fichier
        :param persoDict: dictionnaire donnant la liste des personnages de la pièce
        :param emptyAnalysisDict: dictionnaire à remplir par la fonction
        :param lineNum: numéro de la ligne à partir de laquelle nous commençons la lecture
        :return: dataframe pandas dont le template est donné par le dictionnaire
    """
    currentLine = fileToAnalyse.readline()
    lineNum += 1
    # Expression régulière de détection de déclaration des actes
    # Ligne qui commence par exactement 2 #, ni plus ni moins, donc tout autre caractère qu'un #
    actLinePattern = re.compile('^#{2}[^#]')
    # Expression régulière de détection de déclaration des scènes
    # Ligne qui commence par exactement 3 #, ni plus ni moins, donc tout autre caractère qu'un #
    sceneLinePattern = re.compile('^#{3}[^#]')
    # Variable pour enregistrer le numéro de l'acte courant
    actNum = 0
    # Variable pour enregistrer le numéro de la scène courante
    sceneNum = 0
    # Variable indiquant si la ligne précédente déclare une scène
    previousLineIsScene = False
    # Variable contenant l'auteur courant de la réplique analysée
    # Nous l'initialisons à la valeur vide None
    currentAuthor = None
    # Variable qui va contenir le contenu des répliques initialisée à chaîne de caractères vide
    currentSpeechContent = ''
    # Variable donnant le nombre de mots de la réplique courante analysée. On l'initialise à 0.
    currentSpeechLen = 0

    # Boucle principale de parcours du fichier texte
    while (currentLine):
        if actLinePattern.match(currentLine):
            # Déclaration d'un acte de la pièce
            # On incrémente le numéro d'acte
            actNum += 1
            # Si ce n'est pas le premier acte,
            # il faut traiter le cas de la dernière réplique de
            # de la dernière scène de l'acte précédent.
            # On considère que l'auteur courant s'adresse forcément
            # au dernier auteur enregistré.
            if actNum > 1:
                # Récupération du dernier élément de la colonne des auteurs

```

```

# qui est l'auteur de la réplique précédente
previousAuthor = emptyAnalysisDict['author'][-1]
if len(scenePersoList)==1:
    # Si la liste des personnages ne contient qu'un
    # seul protagoniste, c'est la scène qui fait exception
    # avec Harpagon.
    previousAuthor = 'Harpagon'
    emptyAnalysisDict['author'].append(currentAuthor)
    emptyAnalysisDict['recipient'].append(previousAuthor)
    # On a incrémenté Le numéro d'acte, il faut donc
    # le décrémenter pour obtenir celui concerné
    # par la réplique considérée ici.
    emptyAnalysisDict['act'].append(actNum-1)
    emptyAnalysisDict['scene'].append(sceneNum)
    emptyAnalysisDict['speech'].append(currentSpeechContent)
    emptyAnalysisDict['speech_length'].append(currentSpeechLen)
    # On remet à zéro Le numéro de la scène
    sceneNum = 0
    # Variable qui va contenir le contenu des répliques initialisée à cha
    îne de caractères vide
    currentSpeechContent = ''
    # Variable qui va contenir le nombre de mots des répliques initialisé
    e à 0
    currentSpeechLen = 0
elif sceneLinePattern.match(currentLine):
    # Déclaration d'une scène de la pièce
    # On incrémente le numéro de scène
    sceneNum += 1
    # Si ce n'est pas la première scène de l'acte,
    # il faut traiter le cas de la dernière réplique.
    # On considère que l'auteur courant s'adresse forcément
    # au dernier auteur enregistré.
    if (sceneNum > 1):
        # Récupération du dernier élément de la colonne des auteurs
        # qui est l'auteur de la réplique précédente
        previousAuthor = emptyAnalysisDict['author'][-1]
        emptyAnalysisDict['author'].append(currentAuthor)
        emptyAnalysisDict['recipient'].append(previousAuthor)
        emptyAnalysisDict['act'].append(actNum)
        emptyAnalysisDict['scene'].append(sceneNum-1)
        emptyAnalysisDict['speech'].append(currentSpeechContent)
        emptyAnalysisDict['speech_length'].append(currentSpeechLen)

    # On met temporairement la variable
    # previousLineIsScene à vrai pour indiquer
    # que l'on a détecté le début d'une scène
    previousLineIsScene = True
    # On déclare la liste des protagonistes de la scène courante
    scenePersoList = []
    # On réinitialise les autres variables de l'auteur et de la taille d'
    une réplique
    currentAuthor = None
    # Variable qui va contenir le contenu des répliques initialisée à cha
    îne de caractères vide
    currentSpeechContent = ''
    # Variable qui va contenir le nombre de mots des répliques initialisé

```

```

e à 0
    currentSpeechLen = 0
else:
    # La ligne courante est une ligne autre,
    # comme on a pu le voir dans l'exemple un peu plus haut
    # elle peut être:
    # - vide
    # - une déclaration des personnages de la scène
    # - une identification de l'auteur de la réplique qui va suivre
    # - une ligne composant une réplique
    # - une ligne donnant des indications sur la mise en scène
    # - une ligne indiquant la fin d'un acte
    if not previousLineIsScene:
        # La ligne courante ne contient forcément pas la liste des protagon
        istes de la scène
        # Testons si la ligne est vide auquel cas on peut quasiment passer
        directement à la suivante.
        # Notons qu'une chaîne de caractères vide est interprétée par pyth
        on comme une valeur booléenne fausse
        if currentLine:
            # La ligne n'est pas vide, c'est soit la déclaration de l'auteur
            d'une réplique,
            # soit une ligne constituant une réplique.
            # On va donc traiter cette ligne soit de manière à identifier l'a
            uteur qui,
            # il faut le noter est le destinataire de la réplique de l'auteur
            précédent,
            # soit de manière à compter le nombre de mots total d'une répliqu
            e afin
            # de remplir avec toutes les informations glanées à cette étape
            # de synthèse des données de la pièce.
            # On peut noter également que l'on peut facilement différencier l
            es lignes de répliques
            # des lignes donnant l'auteur d'une réplique. En effet, les ligne
            s de répliques commencent
            # par une lettre majuscule tandis que les lignes donnant l'auteur
            d'une réplique
            # commencent par des espaces.
            if currentLine.startswith(" "):
                # La ligne commence avec un espace, elle déclare donc l'auteur
                d'une réplique
                # on va parcourir la liste des protagonistes de la scène et ide
                ntifier l'auteur
                for perso in scenePersoList:
                    #####
                    #####
                    #####
                    #####
                    # La poursuite du travail nous a permis de voir qu'il y avait
                    un problème avec l'approche suivante
                    # qui a été mise en commentaires:
                    # _____

                    ## On fait une recherche ignorant si les lettres sont en majuscules ou minuscules
                    ## grâce à l'option passée à la fonction de recherche d'expres

```

ssion régulière

## re.IGNORECASE. On traite également du problème posé par les maitre/maîtres

## car les valeurs utilisées dans scenePersoList sont celles issues du dictionnaire

## des personnages créé à partir de la liste en début de pièce.

```
#persoRegex = perso
```

```
#if perso.startswith("Maitre"):
```

```
#    persoRegex = persoRegex.replace('Maitre', 'Maître')
```

```
#m = re.search(persoRegex, currentLine, re.IGNORECASE)
```

```
#
```

---

# En effet, la ligne suivante extraite du texte pose problème à cette approche:

# HARPAGON \*bas à Valère\*. --> il y a deux personnages cités dans cette ligne de déclaration

# de l'auteur de la réplique ce qui va fausser notre traitement.

# Nous allons donc implémenter la correction suivante en ignorant pas si les lettres sont en majuscule

# ou en minuscule, mais en faisant en sorte de transformer la chaîne de caractères qui est contenue

# dans 'perso' en son équivalent tout en majuscule.

# Malheureusement, en continuant l'investigation entreprise à ce moment là, nous nous sommes rendus compte

# d'un autre problème non identifié auparavant. Il arrive qu'un personnage intervenant dans une scène

# ne soit pas listé en début de scène (ex. Valère Acte 3, scène 9)

# La liste des personnages d'une scène que nous avons constituée pour gagner du temps dans l'analyse

# n'est donc pas assez fiable et l'on doit systématiquement parcourir la liste de l'ensemble des personnages

# de la pièce.

```
persoRegex = perso
```

```
if perso.startswith("Maitre"):
```

```
    persoRegex = persoRegex.replace('Maitre', 'Maître')
```

```
persoRegex = persoRegex.upper()
```

```
m = re.search(persoRegex, currentLine)
```

# Pour traiter le cas où un personnage d'une scène ne serait pas déclaré dans la ligne prévue

# à cette effet, on va vérifier si l'auteur déclaré existe dans le dictionnaire global des personnages

```
if not m:
```

```
    for perso in avarePersoDict:
```

# On ne teste que les personnages qui ne sont pas dans la liste des personnages de la scène

```
        if perso not in scenePersoList:
```

```
            persoRegex = perso
```

```
            if perso.startswith("Maitre"):
```

```
                persoRegex = persoRegex.replace('Maitre', 'Maître')
```

```
            persoRegex = persoRegex.upper()
```

```
            m = re.search(persoRegex, currentLine)
```

```
            if m:
```

```
                # Le personnage existe dans le dictionnaire mais n'a
```

```

pas été correctement déclaré
        # on l'ajoute à la liste des personnages de la scène
        scenePersoList.append(perso)
#####
#####
    if m:
        # Le résultat de la recherche n'est pas vide, l'auteur est
        trouvé.
        # Vérifions si quelqu'un a déjà parlé dans la scène auquel
        cas l'auteur
        # courant est le destinataire de l'auteur précédent.
        if currentAuthor is not None:
            # Il y avait déjà un auteur d'une réplique, currentAuthor
            # le nouvel auteur est donc le destinataire de la répliqu
            e précédente.
            # On peut maintenant remplir le dictionnaire pour la répl
            ique précédente,
            # car nous avons toutes les informations nécessaires sur
            celles-ci.
            # Pour rappel, la structure du dictionnaire,
            # {'author':[], 'recipient':[], 'act':[], 'scene':[], 'speech
            _length':[]}]
            emptyAnalysisDict['author'].append(currentAuthor)
            emptyAnalysisDict['recipient'].append(perso)
            emptyAnalysisDict['act'].append(actNum)
            emptyAnalysisDict['scene'].append(sceneNum)
            emptyAnalysisDict['speech'].append(currentSpeechContent)
            emptyAnalysisDict['speech_length'].append(currentSpeechLe
n)
            # On réinitialise la variable enregistrant le contenu de
            la réplique courante
            currentSpeechContent = ''
            # On réinitialise la variable mesurant le nombre de mot d
            'une réplique
            currentSpeechLen = 0
            # Fin du si sur le test de savoir si il y a un auteur d'une
            réplique
            # précédent
            # Dans tous les cas l'auteur courant est donné par la valeu
            r "perso"
            # qui a été trouvée dans la ligne.
            currentAuthor = perso
            # Le break permet d'arrêter la boucle for car l'auteur cour
            ant a été identifié
            break
        else:
            isNotActEndDeclarationLine = not currentLine.startswith("Fin d
u")
            isNotPlayInstructionRoleLine = lineNum not in playInstructionsL
ineNumbersList
            # Si nous ne sommes pas dans le cas d'une ligne de déclaration
            de fin d'acte
            # ou bien d'une ligne d'instructions de mise en scène, nous som
            mes
            # dans le cas d'une ligne qui constitue une réplique.

```

```

        # Dans ce cas, comptons le nombre de mots de celle-ci et ajoutons-le au nombre de mots
        # déjà comptabilisé pour la réplique en cours.
        # Nous faisons appel ici à une fonction spécialisée dans le comptage des mots d'une ligne.
        if isNotActEndDeclarationLine and isNotPlayInstructionRoleLine:
            # Afin d'illustrer le traitement et vérifier que tout se passe bien
            # nous allons afficher les sorties pour la première scène de l'acte premier
            nbWords = count_words(currentLine)
            #if actNum==1 and sceneNum==1:
            # print("Ligne de dialogue : {}".format(currentLine))
            # print("Nombre de mots comptés : {}".format(nbWords))
            currentSpeechContent += currentLine
            currentSpeechLen += nbWords
            # Il faut tester si c'est la dernière ligne de la pièce
            # et mettre à jour le dictionnaire dans ce seul cas
            patternExcludingPieceEndDeclaration = re.compile('(.+)(?=< Fin >)'
n >'))

            m = patternExcludingPieceEndDeclaration.match(currentLine)
            if m:
                # C'est la dernière ligne de texte de la pièce
                previousAuthor = emptyAnalysisDict['author'][-1]
                emptyAnalysisDict['author'].append(currentAuthor)
                emptyAnalysisDict['recipient'].append(previousAuthor)
                emptyAnalysisDict['act'].append(actNum)
                emptyAnalysisDict['scene'].append(sceneNum)
                emptyAnalysisDict['speech'].append(currentSpeechContent)
                emptyAnalysisDict['speech_length'].append(currentSpeechLen)

            else:
                # La ligne courante présente forcément les personnages de la scène
                # car c'est la ligne qui suit la déclaration de scène,
                # à l'exception d'une seule scène, la 7 de l'acte 4
                # On peut donc réduire l'analyse du texte à ces seuls
                # protagonistes.
                # Boucle sur les clés du dictionnaire de personnage
                # qui sont leur nom.

                # Si la ligne contient autre chose que des caractères "blancs"
                # espaces, tabulations, retour ligne etc...
                # ce n'est pas l'exception.
                if not re.match('^[\\s]+$',currentLine):
                    # La ligne n'est pas vide, ce n'est pas l'exception
                    for namePerso in persoDict:
                        # On vérifie si namePerso fait partie des protagonistes
                        # Pour prendre en compte le problème avec les maîtres/maîtres
                        # il faut adapter namePerso pour être une expression régulière
                        persoRegex = namePerso
                        if namePerso.startswith("Maitre"):
                            persoRegex = persoRegex.replace('Maitre', 'Maître')

                    m = re.search(persoRegex, currentLine, re.IGNORECASE)
                    if m is not None:
                        # Le résultat de la recherche n'est pas vide,

```



```

        # Le personnage fait partie des protagonistes
        # on l'ajoute à la liste des personnages de la scène
        # On met le nom tel qu'il est écrit dans le dictionnaire
        # créé à partir de la liste en début de pièce.
        #scenePersoList.append(m.group(0))
        scenePersoList.append(namePerso)
        #print("scenePersoList : {}".format(scenePersoList))

    else:
        # La ligne est vide on sait que c'est le cas particulier
        # le seul protagoniste est Harpagon
        scenePersoList.append('Harpagon')
        #print("*****")
        #print("Liste des personnages de la scène")
        #print("*****")
        #print(scenePersoList)
        #print("*****")
        # La ligne précédente n'est désormais plus une déclaration
        # de début de scène
        previousLineIsScene = False

    currentLine = fileToAnalyse.readline()
    # On lit une ligne on incrémente donc le numéro de la ligne courante dans le fichier
    lineNumber += 1

    return pd.DataFrame.from_dict(emptyAnalysisDict)

```

```

In [13]: # On ouvre Le fichier local en lecture 'r' pour en faire l'analyse en utilisant l'instruction with qui se chargera de fermer
# Le fichier une fois sortie de l'instruction (pas d'erreur possible par oubli d'appel à l'instruction close).
# Un rapide coup d'oeil au fichier texte nous montre une organisation, des symboles en début de ligne etc, que l'on va utiliser
# pour "parser" le fichier, à savoir le lire de manière à ranger les données de manière intelligente dans une structure de
# données facilitant la manipulation et l'analyse.
with open(local_filename, 'r') as avareFile:
    # On va commencer par parser le fichier afin de récupérer la liste des personnages
    lineNum = fill_perso_dict(avareFile, avarePersoDict)

    #print("#####")
    #print("Contenu du dictionnaire des personnages initialisé")
    #print("#####\n")
    #print(avarePersoDict)
    #print("\n#####")

    textDataSynthesisTableDf = generate_text_synthesis_data_table(avareFile, avarePersoDict, avareAnalysisDict, lineNum)

# Le tableau de données récapitulatif est maintenant prêt à être utilisé
# Affichons celui-ci
textDataSynthesisTableDf

```

Out[13]:

	act	author	recipient	scene	speech	speech_length
0	1	Valère	Élise	1	\n\nHé quoi, charmante Élise, vous devenez mél...	58
1	1	Élise	Valère	1	Non, Valère, je ne puis pas me repentir de tou...	68
2	1	Valère	Élise	1	Hé que pouvez-vous craindre, Élise, dans les b...	14
3	1	Élise	Valère	1	Hélas ! cent choses à la fois : L'emportement ...	51
4	1	Valère	Élise	1	Ah ! ne me faites pas ce tort, de juger de moi...	45
5	1	Élise	Valère	1	Ah ! Valère, chacun tient les mêmes discours.T...	26
6	1	Valère	Élise	1	Puisque les seules actions font connaître ce q...	70
7	1	Élise	Valère	1	Hélas ! qu'avec facilité on se laisse persuade...	63
8	1	Valère	Élise	1	Mais pourquoi cette inquiétude ? \n\n	4
9	1	Élise	Valère	1	Je n'aurais rien à craindre, si tout le monde ...	224
10	1	Valère	Élise	1	De tout ce que vous avez dit, ce n'est que par...	138
11	1	Élise	Valère	1	Ah ! Valère, ne bougez d'ici, je vous prie ; e...	22
12	1	Valère	Élise	1	Vous voyez comme je m'y prends, et les adroite...	215
13	1	Élise	Valère	1	Mais que ne tâchez-vous aussi à gagner l'appui...	24
14	1	Valère	Élise	1	On ne peut pas ménager l'un et l'autre ; et l'...	86
15	1	Élise	Valère	1	Je ne sais si j'aurai la force de lui faire ce...	13
16	1	Cléante	Élise	2	\n\nJe suis bien aise de vous trouver seule, m...	24
17	1	Élise	Cléante	2	Me voilà prête à vous ouïr, mon Frère. Qu'avez...	14
18	1	Cléante	Élise	2	Bien des choses, ma Sœur, enveloppées dans un ...	10
19	1	Élise	Cléante	2	Vous aimez ?\n\n	2
20	1	Cléante	Élise	2	Oui, j'aime.Mais avant que d'aller plus loin, ...	173
21	1	Élise	Cléante	2	Vous êtes-vous engagé, mon Frère, avec celle q...	11
22	1	Cléante	Élise	2	Non ; mais j'y suis résolu ; et je vous conjur...	24

	act	author	recipient	scene	speech	speech_length
23	1	Élise	Cléante	2	Suis-je, mon Frère, une si étrange personne ?\n\n	8
24	1	Cléante	Élise	2	Non, ma Sœur, mais vous n'aimez pas.Vous ignor...	26
25	1	Élise	Cléante	2	Hélas ! mon Frère, ne parlons point de ma sage...	43
26	1	Cléante	Élise	2	Ah ! plutôt au Ciel que votre âme comme la mienn...	10
27	1	Élise	Cléante	2	Finissons auparavant votre affaire, et me dite...	13
28	1	Cléante	Élise	2	Une jeune personne qui loge depuis peu en ces ...	150
29	1	Élise	Cléante	2	J'en vois beaucoup, mon Frère, dans les choses...	27
...	...	...	...	...	...	...
929	5	Harpagon	Valère	5	Maître Jacques.\n\n	2
930	5	Valère	Maitre Jacques	5	C'est toi qui le dis ?\n\n	6
931	5	Maitre Jacques	Harpagon	5	Vous voyez que je ne dis rien.\n\n	7
932	5	Harpagon	Valère	5	Oui. Voilà Monsieur le Commissaire qui a reçu ...	10
933	5	Valère	Harpagon	5	Pouvez-vous me croire capable d'une action si ...	10
934	5	Harpagon	Valère	5	Capable, ou non capable, je veux ravoir mon ar...	9
935	5	Cléante	Harpagon	6	\n\nNe vous tourmentez point, mon Père, et n'a...	41
936	5	Harpagon	Cléante	6	Où est-il ?\n\n	3
937	5	Cléante	Harpagon	6	Ne vous en mettez point en peine. Il est en li...	47
938	5	Harpagon	Cléante	6	N'en a-t-on rien ôté ?\n\n	7
939	5	Cléante	Mariane	6	Rien du tout. Voyez si c'est votre dessein de ...	36
940	5	Mariane	Anselme	6	Mais vous ne savez pas, que ce n'est pas assez...	36
941	5	Anselme	Harpagon	6	Le Ciel, mes Enfants, ne me redonne point à vo...	61
942	5	Harpagon	Cléante	6	Il faut, pour me donner conseil, que je voie m...	11
943	5	Cléante	Harpagon	6	Vous la verrez saine et entière.\n\n	6
944	5	Harpagon	Anselme	6	Je n'ai point d'argent à donner en mariage à m...	13
945	5	Anselme	Harpagon	6	Hé bien, j'en ai pour eux, que cela ne vous in...	13

	act	author	recipient	scene	speech	speech_length
946	5	Harpagon	Anselme	6	Vous obligerez-vous à faire tous les frais de ...	12
947	5	Anselme	Harpagon	6	Oui, je m'y oblige.Êtes-vous satisfait ?\n\n	8
948	5	Harpagon	Anselme	6	Oui, pourvu que pour les Noces vous me fassiez...	12
949	5	Anselme	Le commissaire	6	D'accord. Allons jouir de l'allégresse que cet...	13
950	5	Le commissaire	Harpagon	6	Holà, Messieurs, holà.Tout doucement, s'il vou...	14
951	5	Harpagon	Le commissaire	6	Nous n'avons que faire de vos écritures.\n\n	8
952	5	Le commissaire	Harpagon	6	Oui.Mais je ne prétends pas, moi, les avoir fa...	12
953	5	Harpagon	Maitre Jacques	6	Pour votre paiement, voilà un Homme que je vou...	12
954	5	Maitre Jacques	Anselme	6	Hélas ! comment faut-il donc faire ? On me don...	23
955	5	Anselme	Harpagon	6	Seigneur Harpagon, il faut lui pardonner cette...	8
956	5	Harpagon	Anselme	6	Vous payerez donc le Commissaire ?\n\n	5
957	5	Anselme	Harpagon	6	Soit. Allons vite faire part de notre joie à v...	11
958	5	Harpagon	Anselme	6	Et moi, voir ma chère Cassette.< Fin >\n	6

959 rows × 6 columns

Y-a-t-il des éléments à NaN ou null dans le tableau construit?

```
In [14]: textDataSynthesisTableDf[textDataSynthesisTableDf.isnull().any(axis=1)]
```

```
Out[14]:
```

act	author	recipient	scene	speech	speech_length
-----	--------	-----------	-------	--------	---------------

```
In [15]: textDataSynthesisTableDf[textDataSynthesisTableDf.isna().any(axis=1)]
```

```
Out[15]:
```

act	author	recipient	scene	speech	speech_length
-----	--------	-----------	-------	--------	---------------

## Réponse à la première question

Maintenant, nous pouvons répondre à la première question.

L'observation d'un premier résultat intermédiaire nous a paru suspect. Ce résultat était le suivant:

Harpagon a prononcé 6992 mots dans oute la pièce.  
Cléante a prononcé 3555 mots dans oute la pièce.  
Élise a prononcé 1067 mots dans oute la pièce.  
Valère a prononcé 3016 mots dans oute la pièce.  
Mariane a prononcé 910 mots dans oute la pièce.  
Anselme a prononcé 517 mots dans oute la pièce.  
Frosine a prononcé 2381 mots dans oute la pièce.  
Maitre Simon a prononcé 0 mots dans oute la pièce.  
Maitre Jacques a prononcé 0 mots dans oute la pièce.  
La Flèche a prononcé 1520 mots dans oute la pièce.  
Dame Claude a prononcé 0 mots dans oute la pièce.  
Brindavoine a prononcé 43 mots dans oute la pièce.  
La Merluche a prononcé 47 mots dans oute la pièce.  
Le commissaire a prononcé 0 mots dans oute la pièce.

En effet, de trop nombreux personnages semblaient ne prononcer aucun mot de toutes la pièce. En regardant, de plus près, nous avons pu faire deux constatations:

- le commissaire est accompagné par un clerc qui n'a pas été enregistré dans la liste des personnages, il faut donc corriger l'analyse de la liste des personnages au début de la pièce pour prendre en compte ce cas,
- certains nom de personnage peuvent différer d'un caractère par exemple entre la liste au début et celui utiliser pour annoncer les auteurs de réplique, typiquement Maitre Simon de la liste de début devient MAÎTRE SIMON (non seulement, il est en majuscule mais il prend un ^), Le commissaire devient Le Commissaire dans les protagonistes des scènes Nous avons donc corriger le code pour prendre en compte ces deux problèmes.

```

In [16]: # On va parcourir le dictionnaire des personnages
# et compter le nombre total de mots prononcés
# pour chacun.
import numpy as np
persoList = []
persoNbWords = np.zeros(len(avarePersoDict), dtype=int)
#print(textDataSynthesisTableDf.shape)
for index, perso in enumerate(avarePersoDict):
    m = textDataSynthesisTableDf['author'].str.match(perso, case=False, na=False)
    tmpDf = textDataSynthesisTableDf[m].copy()
    persoList.append(perso)
    persoNbWords[index] = tmpDf['speech_length'].sum()

data = pd.DataFrame({"perso":persoList, "speech_length":persoNbWords})

sortedData = data.set_index('speech_length').sort_index()
sortedData

```

Out[16]:

	perso
speech_length	
0	Dame Claude
0	son clerc
43	Brindavoine
47	La Merluce
197	Maitre Simon
294	Le commissaire
517	Anselme
919	Mariane
1067	Élise
1520	La Flèche
1668	Maitre Jacques
2353	Frosine
2752	Valère
3339	Cléante
6148	Harpagon

Nous pouvons désormais finir de répondre à la première question posée. Le personnage s'exprimant le plus dans la pièce de *L'Avare* de Molière et qui est donc le protagoniste principal, n'est autre qu'**Harpagon**. Quant aux personnages ne s'exprimant pas, nous en avons deux, le **clerc** du commissaire ainsi que **Dame Claude** qui est la servante d'Harpagon.

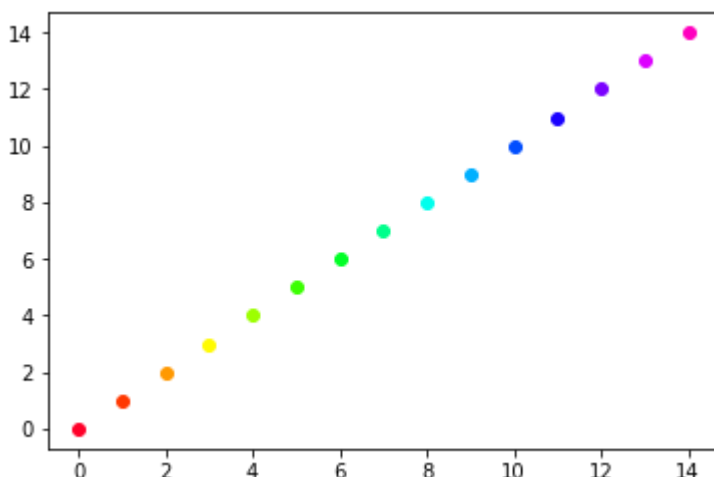
## Réponse à la seconde question

On voulait utiliser le classement pour associer à chaque personnage de la pièce une couleur unique qui reflète son temps de paroles au travers de toute la pièce notamment avec des couleurs plus chaudes, par exemple le rouge pour le personnage avec le nombre de mots le plus élevés, ensuite du orange, puis du jaune etc en allant vers le violet. Néanmoins, le nombre de personnages au total fait qu'une colormap à laquelle nous pensions et qui est illustrée en dessous ne convient pas pour l'affectation des couleurs aux personnages de sorte que l'on puisse les distinguer facilement dans les graphiques.

On va donc reprendre le dictionnaire des personnages et régler la propriété de couleur, mais sans s'occuper d'associer des couleurs en fonction de l'importance des personnages mais surtout de manière à les différencier correctement.

```
In [17]: from matplotlib import pyplot as plt
from matplotlib import cm

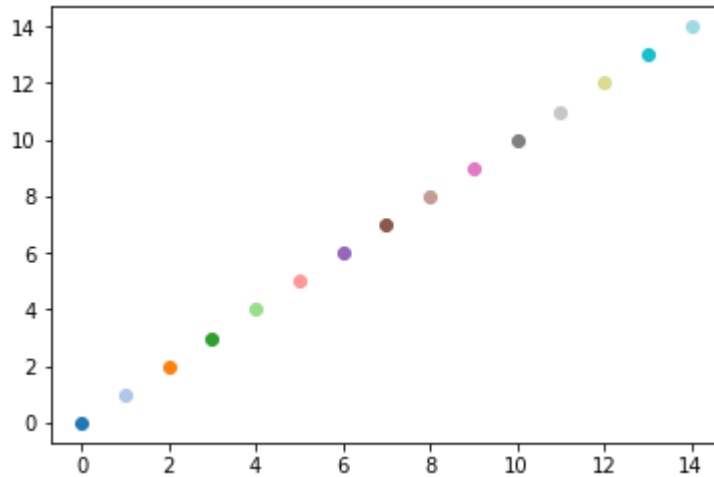
# On affiche une sélection automatique de couleur dans une colormap
# matplotlib
nbPersos = len(persoList)
color = iter(cm.gist_rainbow(np.linspace(0, 1, nbPersos)))
for i in range(nbPersos):
    c = next(color)
    plt.plot(i, i, 'o', c=c)
```



Comme expliqué au-dessus, cette sélection ne convient pas car certaines couleurs sont trop proches les unes des autres. Essayons d'autres colormaps.



```
In [18]: # On affiche une sélection automatique de couleur dans une colormap
# matplotlib
color = iter(cm.tab20(np.linspace(0, 1, nbPersos)))
for i in range(nbPersos):
    c = next(color)
    plt.plot(i, i, 'o', c=c)
```



Cette dernière semble convenir.

```
In [19]: color = iter(cm.tab20(np.linspace(0, 1, nbPersos)))
for i in range(nbPersos):
    c = next(color)
    print(c)
    perso = sortedData["perso"].iloc[i]
    #print(perso)
    avarePersoDict[perso]["color"] = c

avarePersoDict
```

```
[0.12156863 0.46666667 0.70588235 1.      ]
[0.68235294 0.78039216 0.90980392 1.      ]
[1.          0.49803922 0.05490196 1.      ]
[0.17254902 0.62745098 0.17254902 1.      ]
[0.59607843 0.8745098  0.54117647 1.      ]
[1.          0.59607843 0.58823529 1.      ]
[0.58039216 0.40392157 0.74117647 1.      ]
[0.54901961 0.3372549  0.29411765 1.      ]
[0.76862745 0.61176471 0.58039216 1.      ]
[0.89019608 0.46666667 0.76078431 1.      ]
[0.49803922 0.49803922 0.49803922 1.      ]
[0.78039216 0.78039216 0.78039216 1.      ]
[0.85882353 0.85882353 0.55294118 1.      ]
[0.09019608 0.74509804 0.81176471 1.      ]
[0.61960784 0.85490196 0.89803922 1.      ]
```

```
Out[19]: {'Harpagon': {'links': ["Père de Cléante et d'Élise", 'Amoureux de Mariane'],
  'color': array([0.61960784, 0.85490196, 0.89803922, 1.      ])},
  'Cléante': {'links': ["Fils d'Harpagon", 'Amant de Mariane'],
  'color': array([0.09019608, 0.74509804, 0.81176471, 1.      ])},
  'Élise': {'links': ["Fille d'Harpagon", 'Amante de Valère'],
  'color': array([0.76862745, 0.61176471, 0.58039216, 1.      ])},
  'Valère': {'links': ["Fils d'Anselme", "Amant d'Élise"],
  'color': array([0.85882353, 0.85882353, 0.55294118, 1.      ])},
  'Mariane': {'links': ['Amante de Cléante', "aimée d'Harpagon"],
  'color': array([0.54901961, 0.3372549 , 0.29411765, 1.      ])},
  'Anselme': {'links': ['Père de Valère et de Mariane'],
  'color': array([0.58039216, 0.40392157, 0.74117647, 1.      ])},
  'Frosine': {'links': ["Femme d'Intrigue"],
  'color': array([0.78039216, 0.78039216, 0.78039216, 1.      ])},
  'Maitre Simon': {'links': ['Courtier'],
  'color': array([0.59607843, 0.8745098 , 0.54117647, 1.      ])},
  'Maitre Jacques': {'links': ["Cuisinier et Cocher d'Harpagon"],
  'color': array([0.49803922, 0.49803922, 0.49803922, 1.      ])},
  'La Flèche': {'links': ['Valet de Cléante'],
  'color': array([0.89019608, 0.46666667, 0.76078431, 1.      ])},
  'Dame Claude': {'links': ["Servante d'Harpagon"],
  'color': array([0.12156863, 0.46666667, 0.70588235, 1.      ])},
  'Brindavoine': {'links': ["laquais d'Harpagon"],
  'color': array([1.          , 0.49803922, 0.05490196, 1.      ])},
  'La Merluche': {'links': ["laquais d'Harpagon"],
  'color': array([0.17254902, 0.62745098, 0.17254902, 1.      ])},
  'Le commissaire': {'links': [],
  'color': array([1.          , 0.59607843, 0.58823529, 1.      ])},
  'son clerc': {'links': ['assistant du commissaire'],
  'color': array([0.68235294, 0.78039216, 0.90980392, 1.      ]))}
```

On constate ci-dessus que le dictionnaire des personnages de la pièce a bien été complété comme voulu.

En nous appuyant sur une page internet trouvée sur exemple d'affichage similaire à ce qui est fait dans l'étude (à savoir un affichage par Acte, sous forme de barres horizontales pour chaque scène le composant donnant la répartition de la paroles entre les différents protagonistes de chaque scène), on va pouvoir répondre à la seconde question. Voici un lien vers celle-ci: [geeksforgeeks \\_stacked-percentage-bar-plot \(https://www.geeksforgeeks.org/stacked-percentage-bar-plot-in-matplotlib/\)](https://www.geeksforgeeks.org/stacked-percentage-bar-plot-in-matplotlib/).

Pour l'écriture du nombre de mots au sein des graphiques, il a fallu passer par des sorties intermédiaires pour voir comment fonctionnait matplotlib, dans quel sens il parcourt les éléments de son graphique par rapport au dataframe d'entrée que l'on veut représenter. Une sortie intermédiaire obtenue via des prints (un print du dataframe percentDf et de bar dans la boucle for i, bar in enumerate(ax.patches): du code de tracé du graphique un peu plus bas), nous à donner ce qui suit:

	Cléante	Harpagon	La Flèche	Valère	scene	Élise
0	0.000000	0.000000	0.000000	0.561998	1	0.438002
1	0.831694	0.000000	0.000000	0.000000	2	0.168306
2	0.000000	0.644044	0.355956	0.000000	3	0.000000
3	0.141361	0.752618	0.000000	0.000000	4	0.106021
4	0.000000	0.268701	0.000000	0.695866	5	0.035433

```

Rectangle(xy=(0, -0.25), width=0, height=0.5, angle=0)
Rectangle(xy=(0, 0.75), width=0.831694, height=0.5, angle=0)
Rectangle(xy=(0, 1.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0, 2.75), width=0.141361, height=0.5, angle=0)
Rectangle(xy=(0, 3.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0, -0.25), width=0, height=0.5, angle=0)
Rectangle(xy=(0, 0.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0, 1.75), width=0.644044, height=0.5, angle=0)
Rectangle(xy=(0.141361, 2.75), width=0.752618, height=0.5, angle=0)
Rectangle(xy=(0, 3.75), width=0.268701, height=0.5, angle=0)
Rectangle(xy=(0, -0.25), width=0, height=0.5, angle=0)
Rectangle(xy=(0, 0.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0.644044, 1.75), width=0.355956, height=0.5, angle=0)
Rectangle(xy=(0, 2.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0, 3.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0, -0.25), width=0.561998, height=0.5, angle=0)
Rectangle(xy=(0, 0.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0, 1.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0, 2.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0.268701, 3.75), width=0.695866, height=0.5, angle=0)
Rectangle(xy=(0.561998, -0.25), width=0.438002, height=0.5, angle=0)
Rectangle(xy=(0.831694, 0.75), width=0.168306, height=0.5, angle=0)
Rectangle(xy=(0, 1.75), width=0, height=0.5, angle=0)
Rectangle(xy=(0.893979, 2.75), width=0.106021, height=0.5, angle=0)
Rectangle(xy=(0.964567, 3.75), width=0.0354331, height=0.5, angle=0)

```

On voit qu'il y a 25 éléments, qui correspondent à 5 scène x 5 personnages pour l'acte 1. En regardant de près les valeurs de largeur des rectangles width, on voit que les 5 premiers rectangles correspondent au premier personnage (cela correspond aux valeurs de pourcentage de paroles pour les 5 scènes de l'acte) = la première colonne du dataframe qui est affiché juste au dessus, les 5 rectangles suivants correspondent au second personnage et ainsi de suite. Il va falloir calculer les positions des textes en conséquences.

```

In [20]: from matplotlib import colors
# Il va nous falloir compter le nombre d'actes de la pièce
# et faire une boucle sur ces derniers pour construire
# un dataframe par acte à partir du dataframe global
# que dont on va caler la structure en fonction de
# l'exemple disponible sur la page.
# Le dataframe doit contenir une ligne par scène de l'acte
# courant et une colonne par protagoniste de la scène
actNums = pd.unique(textDataSynthesisTableDf["act"])
nbActs = actNums.size

for actNum in actNums:
    actDf = textDataSynthesisTableDf[textDataSynthesisTableDf["act"]==float(actNum)].copy()
    #actDf
    sceneNums = pd.unique(actDf["scene"])
    actPersos = pd.unique(actDf["author"])
    #print("act {0:} persos {1:}".format(actNum,actPersos))
    actDict = {"scene":[]}
    for perso in actPersos:
        # Création d'une colonne par personnage de l'acte
        actDict[perso] = []

    for sceneNum in sceneNums:
        actDict["scene"].append(sceneNum)
        sceneDf = actDf[actDf["scene"]==float(sceneNum)].copy()
        scenePersos = pd.unique(sceneDf["scene"])
        for perso in actPersos:
            tmpDf = sceneDf[sceneDf["author"]==perso].copy()
            if not tmpDf.empty:
                currentPersonNbWordsInScene = tmpDf["speech_length"].sum()
                #print(currentPersonNbWordsInScene)
                actDict[perso].append(currentPersonNbWordsInScene)
            else:
                actDict[perso].append(0)

    # Il faut ensuite transformer le dictionnaire d'acte en dataframe
    # on va en faire une version en pourcentage de paroles par scène.
    # Il faudra récupérer les valeurs du nombre de mots afin d'afficher
    # sur les graphiques.
    #print(actDict)
    actSceneSynthesisGraphDf = pd.DataFrame.from_dict(actDict)
    #print(actSceneSynthesisGraphDf)
    # On va créer une version en pourcentage du dataframe
    # de synthèse par scène d'un acte.
    percentDf = actSceneSynthesisGraphDf.copy()
    # On utilise actPersos en indice pour ne traiter que les colonnes de pers
    onnages
    # et ignorer la colonne donnant le numéro de scène
    percentDf[actPersos] = percentDf[actPersos].div(percentDf[actPersos].sum
    (axis=1), axis=0)
    percentDf = percentDf.sort_index(axis=1)
    #print(percentDf)
    #colors = {perso:colors.rgb2hex(avarePersoDict[perso]['color']) for perso
    in actPersos}

```

```

sortedPersos = np.sort(actPersos)
persoColorsList = []
for perso in sortedPersos:
    persoColorsList.append(avarePersoDict[perso]['color'])

# Rappel: Pour l'affichage en barres empilées à l'horizontale, une recherche nous a permis de tomber
# sur la page suivante qui nous a aidé afin de faire le tracer souhaité:
# --> https://www.geeksforgeeks.org/stacked-percentage-bar-plot-in-matplotlib/

percentDf.plot(\
    x = 'scene', \
    kind = 'barh', \
    color=persoColorsList, \
    stacked = True, \
    title = "Acte {}".format(actNum), \
    mark_right = True)
ax = plt.gca()
ax.invert_yaxis()
sceneCount = len(sceneNums)
for i, bar in enumerate(ax.patches):
    #print(bar)
    # On calcul l'index du personnage dans la liste des personnages de l'acte
    # en prenant la partie entière de la division de i par le nombre de scènes
    # de l'acte. En effet, comme on a pu le montrer un peu plus haut,
    # les données sont parcourues de la manière suivante, les barres du graphique,
    # les rectangles
    # représentant les pourcentages de paroles, sont données pour toutes les
    # scènes pour chaque
    # personnage du dataframe.
    # Pour le premier acte i va aller de 0 à 24 et le résultat de floor(i/s
    ceneCount) va
    # valloir 0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 ... 4 4 4 4 4 où 4 est l'indice
    # du dernier personnage (il y en a 5)
    # On s'assure que persoIdx est un entier pour l'utiliser comme élément
    # d'indexage de l'array numpy des noms
    # des personnages de l'acte courant trié sortedPersos.
    persoIdx = int(np.floor(i/sceneCount))
    #print(persoIdx)
    # On calcul la ligne de la donnée
    # On récupère le nom du personnage
    persoName = sortedPersos[persoIdx]
    #print(persoName)
    # Ensuite, on va calculer le "numéro" de scène auquel correspond la barre
    # courante
    # en utilisant le reste de la division euclidienne de i par le nombre d
    # e scènes
    # Pour le premier acte i va aller de 0 à 24 et le résultat de i % scene
    # Count va
    # valloir 0 1 2 3 4 ... autant de fois qu'il y a de personnages dans l'
    # acte courant
    # et où 0 1 2 3 4 donne le numéro de ligne dans le dataframe
    dataLineIdx = i % sceneCount

```

```

if bar.get_width() != 0:
    # Si la largeur de la barre en cours est différente de 0, l'auteur as
    # associé à prononcé un certain nombre de mots
    # que l'on va récupérer dans le dataframe qui n'a pas été mis en pour
    # centage et on va l'afficher au milieu du rectangle
    # correspondant.
    #print(actSceneSynthesisGraphDf[personName])
    nbWords = actSceneSynthesisGraphDf[personName].iloc[dataLineIdx]
    ax.text(bar.get_x()+bar.get_width()/2, \
            bar.get_y()+bar.get_height()/2, \
            "{:d}".format(nbWords), \
            rotation=20, \
            ha = 'center', \
            va = 'center')

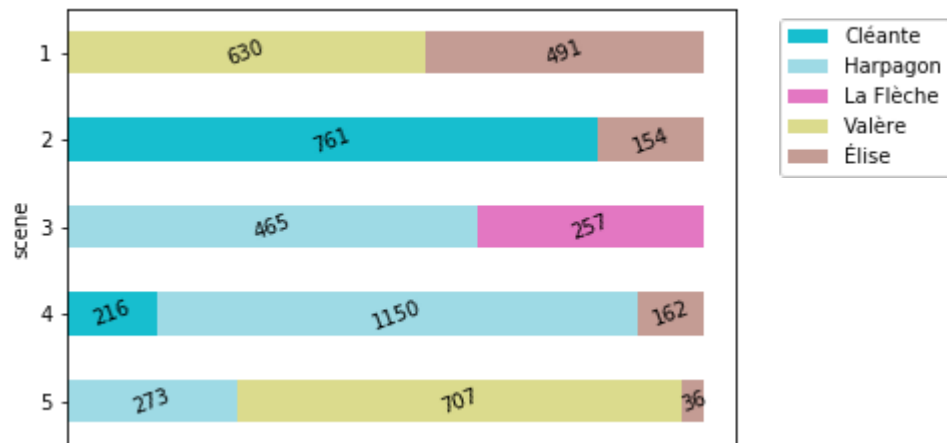
    # cf. (https://stackoverflow.com/questions/66837088/how-to-write-text-inside-the-bar-in-a-horizontal-bar-graph-matplotlib)
    # ax.text(0.1, bar.get_y()+bar.get_height()/2, disease, color = 'white',
    # ha = 'left', va = 'center') --> on comprend
    # que les premiers paramètres donne la position du texte dans la barre notamment grâce à bar.get_y()+bar.get_height()/2
    # qui met la coordonnée en y à la coordonnée de début de la barre + la moitié de sa hauteur. Ensuite, en ayant observé
    # quelques comportements en modifiant les paramètres ha et va on comprend que cela règle l'ancrage du texte, i.e.
    # cela précise à quoi font référence les coordonnées renseignées ha (alignement horizontal) et va (alignement vertical)
    # center indique que la coordonnées renseignée représente le centre de la boîte de texte pour l'axe considéré,
    # center, center --> indique que le centre de la boîte de texte se trouve aux coordonnées passées en entrée.

    # Positionnement de la légende en haut à droite en dehors du plot
    # cf. (https://www.geeksforgeeks.org/how-to-place-legend-outside-of-the-plot-in-matplotlib/)
    plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
    # Hide ticks of x axis on figures
    plt.tick_params(axis='x', \
                    which='both', \
                    bottom=False, \
                    top=False, \
                    labelbottom=False)

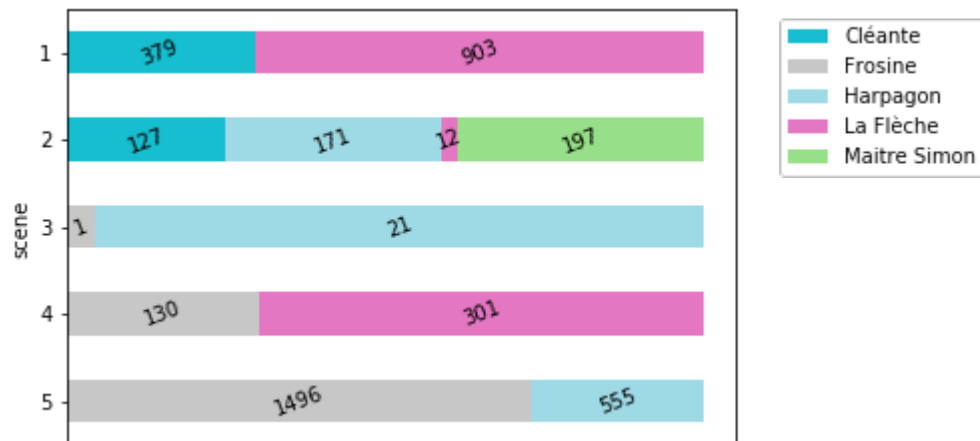
```



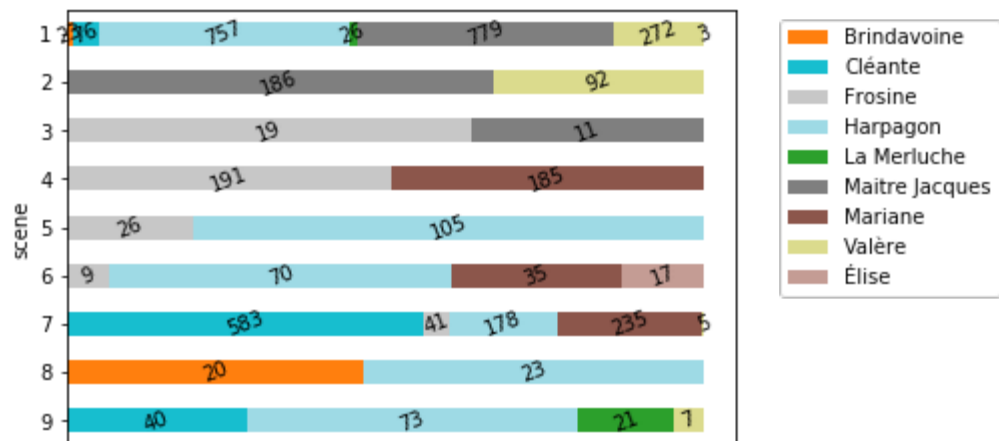
### Acte 1

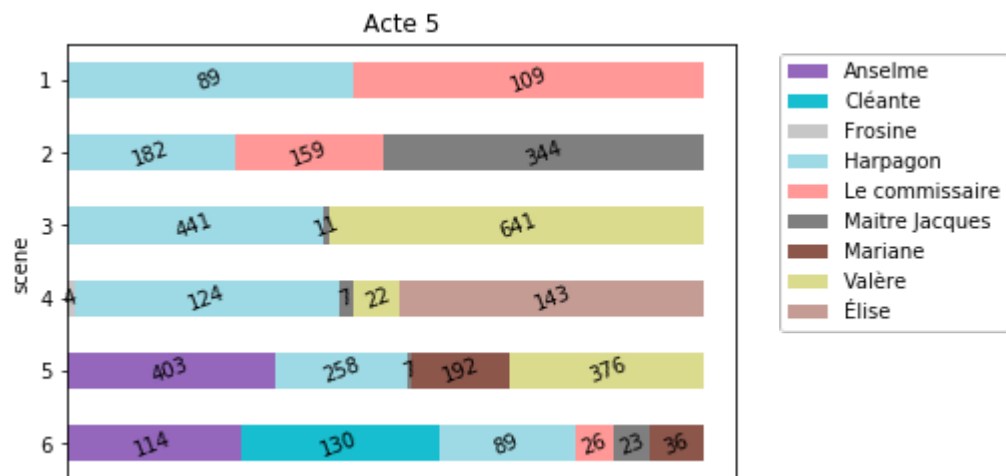
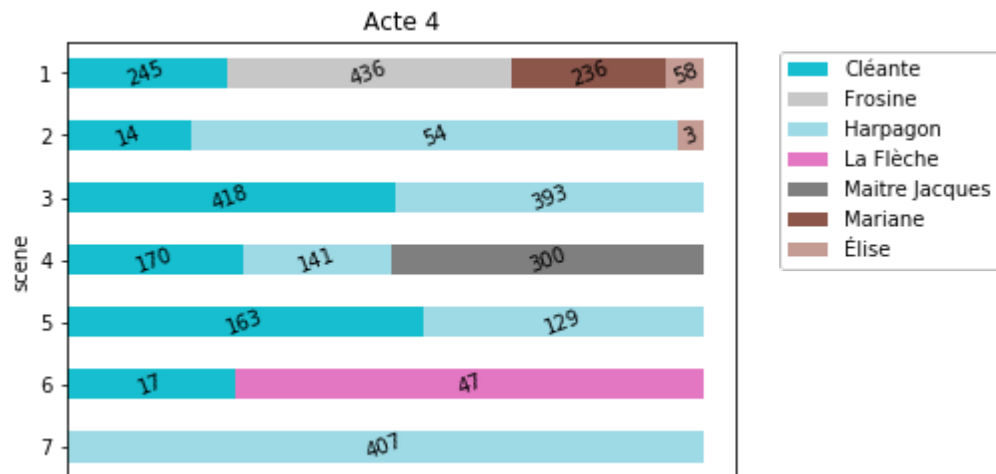


### Acte 2



### Acte 3





Réponse à la question facultative

## Introduction

Un nouveau regard sur le texte de la pièce nous montre que la manière dont nous avons géré auteurs et destinataires des répliques est "naïf". Cela n'a pas d'impact sur les réponses aux deux premières questions mais sur la dernière question oui. Voici un commentaire de notre code:

```
# Il y avait déjà un auteur d'une réplique, currentAuthor  
# le nouvel auteur est donc le destinataire de la réplique précédente.
```

renseignant sur notre approche et qui montre la manière dont nous identifions auteurs et destinataires. Cette approche fonctionne parfaitement lorsqu'il n'y a que deux personnages dans une scène mais n'est plus qu'une approximation lorsqu'il y a plus de deux personnages. En effet, cela peut être plus compliqué comme le montre quelques extraits ci-dessous:

### Extrait 1

---

### Scène IV. (Acte 5)

Élise, Mariane, Frosine, Harpagon, Valère, Maître Jacques, le Commissaire, son Clerc

HARPAGON.

Ah ! fille scélérate ! Fille indigne d'un Père comme moi ! C'est ainsi que tu pratiques les Leçons que je t'ai données ! Tu te laisses prendre d'amour pour un Voleur infâme, et tu lui engages ta foi sans mon consentement ? Mais vous serez trompés l'un et l'autre. Quatre bonnes murailles me répondront de ta conduite ; et une bonne Potence me fera raison de ton audace.

VALÈRE.

Ce ne sera point votre passion qui jugera l'affaire ; et l'on m'écouterà, au moins, avant que de me condamner.

HARPAGON.

Je me suis abusé de dire une Potence ; et tu seras roué tout vif.

ÉLISE, \*à genoux devant son père\*.

Ah ! mon père, prenez des sentiments un peu plus humains, je vous prie, et n'allez point pousser les choses dans les dernières violences du pouvoir paternel : Ne vous laissez point entraîner aux premiers mouvements de votre passion, et donnez-vous le temps de considérer ce que vous voulez faire. Prenez la peine de mieux voir celui dont vous vous offensez : il est tout autre que vos yeux ne le jugent ; et vous trouverez moins étrange que je me sois donnée à lui, lorsque vous saurez que sans lui vous ne m'auriez plus il y a longtemps. Oui, mon Père, c'est celui qui m'a sauvé de ce grand péril que vous savez que j'ai couru dans l'eau, et à qui vous devez la vie de cette même fille, dont...

---

Ci-dessus, la première réplique d'Harpagon s'adresse à sa fille Elise et non à Valère.

## Extrait 2

---

### Scène V. (Acte 5)

Anselme, Harpagon, Élise, Mariane, Frosine, Valère, Maître Jacques, le Commissaire, son Clerc

ANSELME.

Qu'est-ce, Seigneur Harpagon, je vous vois tout ému.

HARPAGON.

Ah ! Seigneur Anselme, vous me voyez le plus infortuné de tous les hommes ; et voici bien du trouble et du désordre au Contrat que vous venez faire ? On m'assassine dans le bien, on m'assassine dans l'honneur ; et voilà un traître, un scélérat, qui a violé tous les droits les plus saints ; qui s'est coulé chez moi sous le titre de Domestique, pour me dérober mon argent, et pour me suborner ma Fille.

VALÈRE.

Qui songe à votre argent, dont vous me faites un galimatias ?

HARPAGON.

Oui, ils se sont donné l'un et l'autre une Promesse de mariage. Cet affront vous regarde, Seigneur Anselme ; et c'est vous qui devez vous rendre partie contre lui, et faire toutes les poursuites de la Justice, pour vous venger de son insolence.

---

Ici la première réplique d'Harpagon n'est pas adressée à Valère mais à Anselme, il lui répond.

### Extrait 3

Dans la scène 2 de l'acte 5:

---

MAÎTRE JACQUES, \*à part\*.

Voici justement ce qu'il me faut pour me venger de notre Intendant : depuis qu'il est entré céans, il est le favori, on n'écoute que ses conseils ; et j'ai aussi sur le cœur les coups de bâton de tantôt.

---

Maître Jacques se parle ici à lui-même.

### Bilan

Afin d'avoir une graphique exacte des échanges, il faudrait donc une analyse du texte bien plus poussée et

intelligente que celle mise en place. Néanmoins, cette approche permet d'avoir une bonne approximation des échanges entre les personnages.\ Partons donc du tableau de données produites afin de générer le graphe des échanges entre personnages même si l'on sait qu'il ne sera qu'approximatif.

## Création du graphe des échanges entre personnages

Pour réaliser le graphe des échanges entre personnage nous nous appuierons notamment sur la bibliothèque python networkx de manipulation de concepts liés aux réseaux, extrait de la page d'accueil de la bibliothèque:

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks

Resources: page officielle de la documentation de networkx sur les types de graphes implémentés [lien \(https://networkx.org/documentation/stable/reference/classes/index.html\)](https://networkx.org/documentation/stable/reference/classes/index.html) --> pour notre besoin, il nous faudra utiliser les graphes MultiDiGraph

lien vers le tutorial de la bibliothèque networkx\ [tutorial\\_networkx \(https://networkx.org/documentation/stable/tutorial.html\)](https://networkx.org/documentation/stable/tutorial.html)

lien stack overflow vers un code de customisation de graphes de la bibliothèque python networkx\ [customisation\\_graphe \(https://stackoverflow.com/questions/25639169/networkx-change-color-width-according-to-edge-attributes-inconsistent-result\)](https://stackoverflow.com/questions/25639169/networkx-change-color-width-according-to-edge-attributes-inconsistent-result)

lien github vers un morceau de code ajoutant de la couleur et le réglage de l'épaisseur des arêtes sur un graphe\ [couleur\\_epaisseur\\_arêtes\\_graphe \(https://gist.github.com/AruniRC/2c53fe7680eeb578593ec816bbfb1653\)](https://gist.github.com/AruniRC/2c53fe7680eeb578593ec816bbfb1653)

Autres ressources trouvées: <https://github.com/paulbrodersen/netgraph/issues/45> (<https://github.com/paulbrodersen/netgraph/issues/45>)

En première étape, il va nous falloir construire la [matrice d'adjacence \(https://fr.wikipedia.org/wiki/Matrice\\_d%27adjacence\)](https://fr.wikipedia.org/wiki/Matrice_d%27adjacence) du graphe des échanges entre personnages afin d'instancier/créer l'objet graphe voulu de la bibliothèque networkx. La matrice d'adjacence sera un dataframe pandas. Mais commençons d'abord par importer la bibliothèque networkx indispensable pour la suite.

```
In [21]: import networkx as nx
```

Pour la matrice d'adjacence, les lignes désignent les auteurs des répliques et les colonnes les destinataires. Cette matrice sera donc une matrice carré nxn où n est le nombre de personnages de la pièce.

```

In [22]: # Création du dictionnaire qui va servir à créer Le dataframe pandas de rep
          # représentation de la matrice d'adjacence
          # qui servira ensuite à la création de L'objet graphe de networkx qui nous
          # permettra d'avoir ensuite de générer
          # une représentation graphique semblable a ce qui est fait dans L'étude OBV
          IL.
sortedPerosList = np.sort(persoList)
adjacencyDict = {"author":sortedPerosList}
#adjacencyDict = {}
for j, recipient in enumerate(sortedPerosList):
    # On initialise une colonne de la matrice
    adjacencyDict[recipient] = []
    for i, author in enumerate(sortedPerosList):
        # Par cette boucle, on va remplir les lignes de la colonne initialisée
        #print("Process Author/Recipient pair : {0}/{1}".format(author, recipie
        nt))
        #tmpDf = textDataSynthesisTableDf[['author','recipient','speech_lengt
        h']]
        tmpDf = textDataSynthesisTableDf
        tmpDf = tmpDf.loc[(tmpDf["author"]==author) & (tmpDf["recipient"]==reci
        pient)].copy()
        totalNbWordsAuthorToRecipient = tmpDf["speech_length"].sum()
        adjacencyDict[recipient].append(totalNbWordsAuthorToRecipient)
        if j==0:
            print("Total words from {0} to {1} = {2}".format(author, recipient, t
            otalNbWordsAuthorToRecipient))

adjacencyDf = pd.DataFrame.from_dict(adjacencyDict)
adjacencyDf = adjacencyDf.set_index('author')
print(adjacencyDf)

```

Total words from Anselme to Anselme = 0  
 Total words from Brindavoine to Anselme = 0  
 Total words from Cléante to Anselme = 0  
 Total words from Dame Claude to Anselme = 0  
 Total words from Frosine to Anselme = 0  
 Total words from Harpagon to Anselme = 126  
 Total words from La Flèche to Anselme = 0  
 Total words from La Merluche to Anselme = 0  
 Total words from Le commissaire to Anselme = 0  
 Total words from Maitre Jacques to Anselme = 23  
 Total words from Maitre Simon to Anselme = 0  
 Total words from Mariane to Anselme = 199  
 Total words from Valère to Anselme = 262  
 Total words from son clerc to Anselme = 0  
 Total words from Élise to Anselme = 0

	Anselme	Brindavoine	Cléante	Dame Claude	Frosine	Harpag
on \						
author						
Anselme	0	0	0	0	0	2
93						
Brindavoine	0	0	0	0	0	
20						
Cléante	0	0	0	0	32	15
95						
Dame Claude	0	0	0	0	0	
0						
Frosine	0	0	436	0	0	15
61						
Harpagon	126	37	1719	0	737	4
07						
La Flèche	0	0	962	0	301	2
57						
La Merluche	0	0	0	0	0	
47						
Le commissaire	0	0	0	0	0	1
46						
Maitre Jacques	23	0	154	0	18	11
56						
Maitre Simon	0	0	0	0	0	1
94						
Mariane	199	0	258	0	256	1
63						
Valère	262	0	5	0	0	14
78						
son clerc	0	0	0	0	0	
0						
Élise	0	0	154	0	10	3
28						

	La Flèche	La Merluche	Le commissaire	Maitre Jacques	\
author					
Anselme	0	0	13	0	
Brindavoine	0	23	0	0	
Cléante	396	0	0	170	
Dame Claude	0	0	0	0	
Frosine	130	0	0	23	

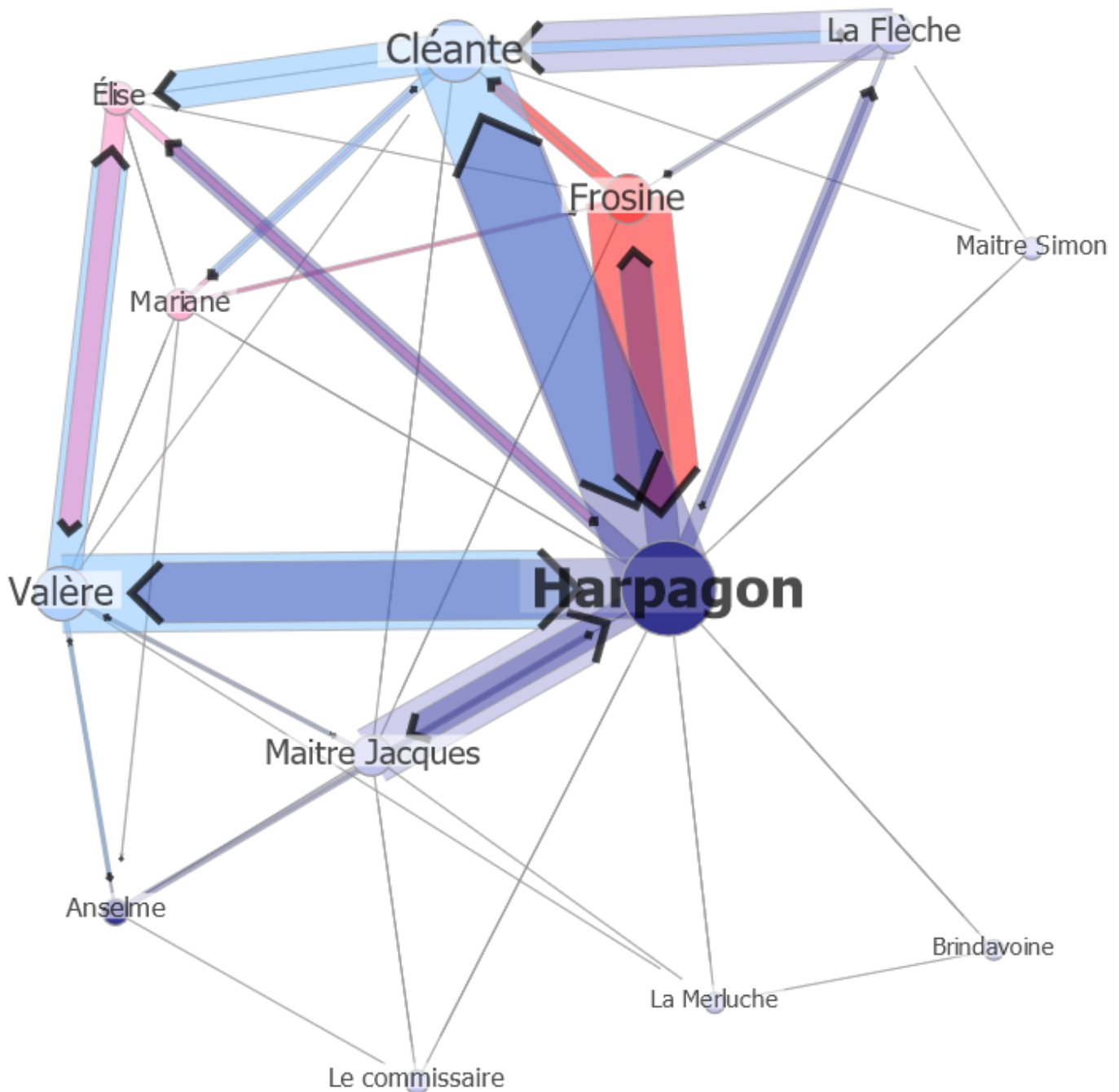


Harpagon	465	6	135	679
La Flèche	0	0	0	0
La Merluche	0	0	0	0
Le commissaire	0	0	0	148
Maitre Jacques	0	8	46	0
Maitre Simon	3	0	0	0
Mariane	0	0	0	0
Valère	0	4	0	218
son clerc	0	0	0	0
Élise	0	0	0	0

	Maitre Simon	Mariane	Valère	son clerc	Élise
author					
Anselme	0	0	211	0	0
Brindavoine	0	0	0	0	0
Cléante	13	350	0	0	783
Dame Claude	0	0	0	0	0
Frosine	0	203	0	0	0
Harpagon	51	94	1178	0	514
La Flèche	0	0	0	0	0
La Merluche	0	0	0	0	0
Le commissaire	0	0	0	0	0
Maitre Jacques	0	0	263	0	0
Maitre Simon	0	0	0	0	0
Mariane	0	0	29	0	14
Valère	0	43	0	0	742
son clerc	0	0	0	0	0
Élise	0	48	527	0	0

In [23]: *# On crée maintenant Le graphe à partir de la matrice d'adjacence générée.*  
initExchangesGraph = nx.MultiDiGraph(adjacencyDf)

Nous allons commencer par visualiser le graphe créé de la manière la plus simple possible avant d'aller plus loin dans sa "customisation". Malheureusement, l'exemple fourni par le site officiel de la bibliothèque [ici](https://networkx.org/documentation/stable/auto_examples/drawing/plot_multigraphs.html) ([https://networkx.org/documentation/stable/auto\\_examples/drawing/plot\\_multigraphs.html](https://networkx.org/documentation/stable/auto_examples/drawing/plot_multigraphs.html)) ne fonctionne pas. On peut se servir de l'image affichée sur le site pour définir la position des noeuds des personnages afin que l'affichage soit le plus lisible possible.



```
In [24]: # Regardons en interne comment sont contruites Les arêtes d'un graphe Multi
         DiGraph comme le notre
         for i, (u, v) in enumerate(initExchangesGraph.edges()):
             if i==0:
                 print(i)
                 print(u)
                 print(v)
                 print(initExchangesGraph[u][v])
             else:
                 break
```

```
0
Anselme
Harpagon
{0: {'weight': 293}}
```

In [25]: `import itertools as it`

```
def draw_labeled_multigraph(G, attr_name, ax=None, posCustom=None, persoCustomDict=None):
    """
    Length of connectionstyle must be at least that of a maximum number of
    edges
    between pair of nodes. This number is maximum one-sided connections
    for directed graph and maximum total connections for undirected graph.
    """
    # Works with arc3 and angle3 connectionstyles
    connectionstyle = f"arc3,rad=0.05"
    # Distribute the positions of the nodes on a kind of circle
    pos = nx.shell_layout(G)
    if posCustom is not None:
        for perso in posCustom:
            pos[perso] = posCustom[perso]
    #pos = nx.spring_layout(G)
    #pos = nx.nx_agraph.graphviz_layout(G)

    #node_color = [G.degree(v) for v in G]
    ## node colour is a list of degrees of nodes
    #
    #node_size = [0.0005 * nx.get_node_attributes(G, 'population')[v] for v
    in G]
    ## size of node is a list of population of cities
    #
    #edge_width = [0.0015 * G[u][v]['weight'] for u, v in G.edges()]
    ## width of edge is a list of weight of edges
    #
    #nx.draw_networkx(G, node_size = node_size,
    #                  node_color = node_color, alpha = 0.7,
    #                  with_labels = True, width = edge_width,
    #                  edge_color = '.4', cmap = plt.cm.Blues)

    # Après quelques essais, il semble qu'une taille de 500 soit intéressante
    # comme taille
    # moyenne pour les noeuds
    defaultAvNodeSize = 1000
    # Après quelques essais, il semble que l'épaisseur par défaut soit 1 et
    # que l'on ne peut guère diminuer
    # l'épaisseur en dessous de 0.5. On va donc régler l'épaisseur des arêtes
    # en tenant compte de cela.
    defaultAvEdgeWidth = 2

    if persoCustomDict is None:
        nodesSize = defaultAvNodeSize * np.ones(len(G.nodes))
        # Affichage des noeuds du graphe
        nx.draw_networkx_nodes(G, pos, node_size=nodesSize, ax=ax)
        nx.draw_networkx_labels(G, pos, font_size=18, ax=ax)
        # Affichage des arêtes du graphe
        # On suppose que les arêtes ont au moins un attribut donnant leur poids
        # on va les parcourir pour créer des épaisseurs représentant ces différences
```

```

# On va créer la liste des arêtes de poids non nul que l'on va utilis
er
# pour l'utiliser en entrée de la fonction de traçage des arêtes du g
raphe
weightsArr = np.array([])
nonZeroEdges = []
for u, v in G.edges():
    edgeWeight = G[u][v][0]['weight']
    if edgeWeight > 0:
        nonZeroEdges.append((u,v))
        weightsArr = np.append(weightsArr, edgeWeight)

# La boucle est finie, on calculer l'importance relative des échanges
# L'échange de taille moyenne va être considéré comme d'importance 1
# Les échanges de taille inférieure entre dans [0, 1] et les échanges
# de taille supérieure [1,+inf]
avEdgeWeight = np.mean(weightsArr)
edgesWidth = defaultAvEdgeWidth * (weightsArr/avEdgeWeight)
edgesWidth[edgesWidth<=0.5] = 0.5
edgesWidth[(edgesWidth>0.5)*(edgesWidth<=1)] = 1
nx.draw_networkx_edges(
    G, pos, edge_color="grey", width=defaultAvEdgeWidth, connectionst
yle=connectionstyle, ax=ax
)
nx.draw_networkx_edges(
    G,
    pos,
    edgelist=nonZeroEdges,
    connectionstyle=connectionstyle,
    width=edgesWidth,
    alpha=0.5
)

labels = {
    tuple(edge[0:2]): f"{attrs[attr_name]} words"
    for *edge, attrs in G.edges(keys=True, data=True)
}

#print(labels)
nx.draw_networkx_edge_labels(
    G,
    pos,
    labels,
    connectionstyle=connectionstyle,
    label_pos=0.2,
    font_color="blue",
    bbox={"alpha": 0},
    ax=ax
)
else:
    # On ajoute les attributs provenant du dictionnaire passé en entrée a
u noeuds
    # du graphe.
    nx.set_node_attributes(G, persoCustomDict)
    # Occupons nous régler la dimension des noeuds relativement à leur im

```

```

portance
    # qui est donnée par l'attribut "node_weight".
    nodesWeightDict = nx.get_node_attributes(G, 'node_weight')
    nodesWeight = np.array([nodesWeightDict[name] for name in nodesWeight
Dict])
    # On cherche le poids median des noeuds dont le poids est non nul pou
r régler la taille
    # des noeuds relativement à cette valeur en considérant que le poids
médian correspond
    # à la taille par défaut.
    nodesAvWeight = np.median(nodesWeight[nodesWeight>0])
    nodesWeight = nodesWeight/nodesAvWeight
    # On remplace les valeurs de poids relatifs des noeuds à poids nul po
ur qu'ils apparaissent
    # aussi petit que le noeud non nul à l'importance minimal
    nodesWeight[nodesWeight==0] = np.min(nodesWeight[nodesWeight>0])
    nodesSize = defaultAvNodeSize * nodesWeight
    # On va récupérer la couleur des noeuds, après avoir regardé ce que
contenait le retour de
    # la fonction nx.get_node_attributes(G, 'color'), elle renvoie un dict
ionnaire avec comme clés
    # les noms des noeuds et comme valeur la valeur de l'attribut demand
é.
    colorsDict = nx.get_node_attributes(G, 'color')
    nodesColor = [colors.rgb2hex(colorsDict[name]) for name in colorsDic
t]

    # Affichage des noeuds du graphe
    #nx.draw_networkx_nodes(G, pos, node_size=nodesSize, ax=ax)
    nx.draw_networkx_nodes(G, pos, node_size=nodesSize, node_color=nodesC
olor, ax=ax)
    nx.draw_networkx_labels(G, pos, font_size=20, ax=ax)

    # Affichage des arêtes du graphe
    # On suppose que les arêtes ont au moins un attribut donnant leur poi
ds
    # on va les parcourir pour créer des épaisseurs représentant ces diff
érences
    # On va créer la liste des arêtes de poids non nul que l'on va utilis
er
    # pour l'utiliser en entrée de la fonction de traçage des arêtes du g
raphe
    # De plus on va également se servir du dictionnaire de customisation
en entrée
    # pour customiser la couleur des arêtes en fonction de l'auteur.
    # On va donc également créer une liste de couleur.
    weightsArr = np.array([])
    edgesColor = []
    nonZeroEdges = []
    for u, v in G.edges():
        # u représente le noeud de départ, donc l'auteur
        # v le destinataire.
        edgeWeight = G[u][v][0]['weight']
        if edgeWeight > 0:
            nonZeroEdges.append((u,v))
            edgesColor.append(persoCustomDict[u]['color'])
            weightsArr = np.append(weightsArr, edgeWeight)

```

```

# La boucle est finie, on calcule l'importance relative des échanges
# L'échange de taille moyenne va être considéré comme d'importance 1
# Les échanges de taille inférieure entre dans [0, 1] et les échanges
# de taille supérieure [1,+inf]
avEdgeWeight = np.mean(weightsArr)
edgesWidth = defaultAvEdgeWidth * (weightsArr/avEdgeWeight)
edgesWidth[edgesWidth<=0.5] = 0.5
edgesWidth[(edgesWidth>0.5)*(edgesWidth<=1)] = 1
#nx.draw_networkx_edges(
#    G, pos, edge_color="grey", width=defaultAvEdgeWidth, connection
tyle=connectionstyle, ax=ax, alpha=0.2
#)
nx.draw_networkx_edges(
    G,
    pos,
    edgelist=nonZeroEdges,
    connectionstyle=connectionstyle,
    edge_color=edgesColor,
    width=edgesWidth,
    alpha=0.98
)

labels = {
    tuple(edge[0:2]): f"{attrs[attr_name]} wd"
    for *edge, attrs in G.edges(keys=True, data=True)
}

#Afin de pouvoir personnaliser chaque label des arêtes du graphe, il
est nécessaire de faire une boucle
# de tracer car l'interface fonctionnelle de la bibliothèque ne perme
t pas de passer des list, dictionnaires
# ou autre collections pour les couleurs des labels etc
# Après avoir constitué la liste des labels que l'on veut écrire sur
Le graphe, on va donc parcourir
# Les arêtes du graphe et les tracer une par une
colorsDict = nx.get_node_attributes(G, 'color')
for u,v in G.edges():
    # u représente le noeuds de départ, donc l'auteur
    # v le destinataire.
    currentFontColor = colors.rgb2hex(colorsDict[u])
    nx.draw_networkx_edge_labels(
        G,
        pos,
        edge_labels={(u,v):labels[(u,v)]},
        connectionstyle=connectionstyle,
        label_pos=0.72,
        font_size=16,
        font_color=currentFontColor,
        font_weight='bold',
        bbox={"alpha": 0.5, "color": (1.0,1.0,1.0)},
        ax=ax,
        rotate=True
    )

```





L'affichage des positions données par shell layout nous montre que les positions sont disposées sur un cercle de rayon 1. cf. la copie des ces positions ci-dessous:

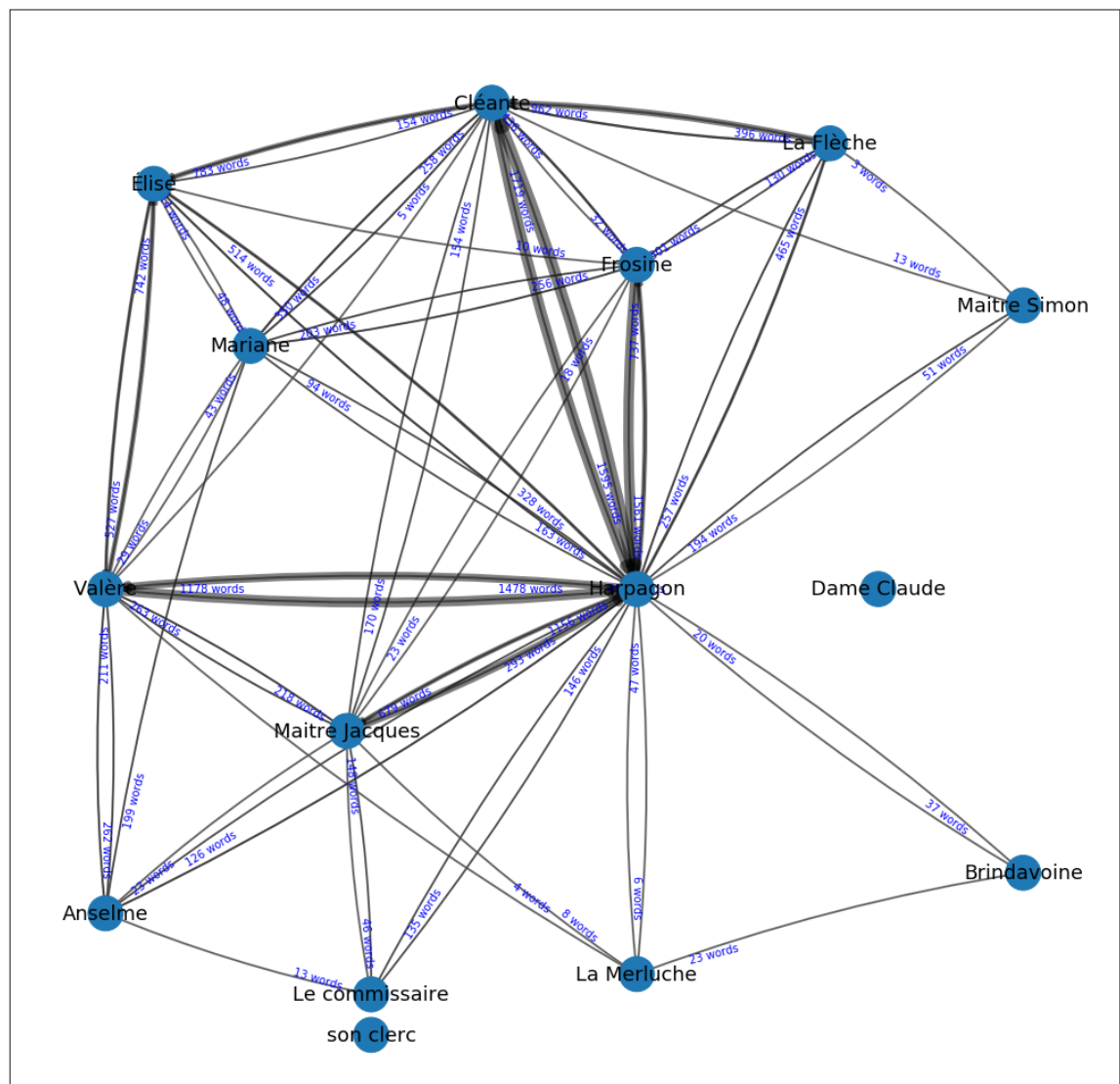
```
#{'Anselme': array([1.00000000e+00, 7.94728727e-09]), 'Brindavoine': array
([0.91354549, 0.4067367 ]),
#'Cléante': array([0.66913056, 0.74314493]), 'Dame Claude': array([0.3090169
7, 0.95105666]),
#'Frosine': array([-0.10452856, 0.99452204]), 'Harpagon': array([-0.5000001
8, 0.86602551]),
#'La Flèche': array([-0.80901724, 0.58778524]), 'La Merluche': array([-0.978
1478 , 0.20791166]),
#'Le commissaire': array([-0.97814775, -0.2079118 ]), 'Maitre Jacques': array
([-0.80901712, -0.58778542]),
#'Maitre Simon': array([-0.5 , -0.86602557]), 'Mariane': array([-0.1045
2839, -0.99452204]),
#'Valère': array([ 0.30901712, -0.9510566 ]), 'son clerc': array([ 0.66913074
, -0.74314481]),
#'Élise': array([ 0.91354561, -0.40673655])}
```

On va tout d'abord corriger les positions des personnages afin que le graphe soit plus lisible. On se chargera ensuite de travailler sur la personnalisation de celui-ci: couleur et taille des noeuds, couleur et épaisseur des arêtes.

```

In [27]: pos = {}
pos['Harpagon'] = np.array([0,0])
pos['Cléante'] = np.array([-0.3,1.2])
pos['Frosine'] = np.array([0, 0.8])
pos['La Flèche'] = np.array([0.4, 1.1])
pos['Maitre Simon'] = np.array([0.8,0.7])
pos['Brindavoine'] = np.array([0.8,-0.7])
pos['Dame Claude'] = np.array([0.5,0])
pos['Élise'] = np.array([-1,1])
pos['Maitre Jacques'] = np.array([-0.6,-0.35])
pos['Le commissaire'] = np.array([-0.55,-1])
pos['son clerc'] = np.array([-0.55,-1.1])
pos['Anselme'] = np.array([-1.1,-0.8])
pos['La Merluche'] = np.array([0,-0.95])
pos['Valère'] = np.array([-1.1,0])
pos['Mariane'] = np.array([-0.8,0.6])
fig = plt.figure(figsize=(18,18))
ax = plt.gca()
draw_labeled_multigraph(initExchangesGraph, "weight", ax=ax, posCustom=pos)

```



Ajoutons au dictionnaire des personnages, le nombre total de mots prononcés dans toute la pièce par chacun. Cela permettra de dimensionner les noeuds relativement à leur importance dans la pièce.



Dans le graphe tracé ci-dessus:

- sont affichés les échanges entre les personnages via des arêtes courbées dont la couleur est associée à l'auteur, au noeud dont ces échanges sont issus,
- sont affichés le nombre total de mots prononcés par chaque personnage à un autre personnage de la pièce dans une couleur associée à la couleur de leur auteur, du noeud dont sont issus les mots et sont plus proches du noeuds d'émission que du noeud donnant le destinataire,
- les noeuds ont une dimension relative au nombre total de mots prononcés par l'auteur associé au noeud,
- les arêtes, qui symbolisent les échanges ont une épaisseur réglée en fonction de l'importance relative des échanges d'un auteur vers un destinataire vis-à-vis de l'importance en nombre de mots des autres échanges entre personnages au travers de la pièce. Le graphe pourrait encore être amélioré en modifiant les positions des noeuds, mais le résultat obtenu ci-dessus se montre plutôt lisible et nous considérons donc la question optionnelle comme résolue.