



UNIL | Université de Lausanne

GUIDE BOTTICELLI

July 13, 2021

Amin Mekacher
Centre des Littératures en Suisse Romande

1 Présentation

BOTticelli est une gamme d'outils développée afin de simplifier l'acquisition et l'entrée de données sur la base en ligne Atom, en automatisant certaines tâches répétitives par le biais d'un bot.

2 Description des étapes

Le fonctionnement de BOTticelli est défini par des étapes qui sont implémentées via des méthodes Python spécialisées. L'exécution du programme combine des tâches effectuées automatiquement par le code, et des corrections qui doivent être prises en charge par un utilisateur. L'exécution du code suit la logique suivante :

Acquisition des données. Un bot est employé pour parcourir automatiquement toutes les notices actuellement présentes sur Atom et pour récolter les données relatives à chacune d'entre elles. Seules les notices décrivant de la correspondance sont collectées et une détection automatique de noms propres est utilisée sur le titre de la notice pour tenter d'identifier les personnes dont il est fait mention. Il est possible de fournir une liste de fonds à analyser, si l'on veut limiter la portée de l'analyse.

Nettoyage des données. Les données obtenues via la détection automatique étant imprécises, cette étape permet d'effectuer un premier filtre automatique, notamment pour détecter des doublons qui sont formulés différemment dans la base de données. Pour cela, une métrique de distance est employée pour détecter deux noms qui sont quasiment similaires. Dans le cas où certains noms ont été spécifiés incorrectement à de nombreuses reprises dans les fonds, qu'il s'agisse par exemple d'initiales entrées à la place du nom entier, il est possible d'entrer une liste de substitutions supplémentaires, que le bot effectuera durant cette étape.

Correction humaine. Cette étape est nécessaire pour s'assurer que les données qui seront entrées automatiquement lors de l'indexation automatique sont correctes. Les noms, qui ont été identifiés et filtrés lors des étapes précédentes, sont stockés dans une colonne spécifique du fichier CSV, qui peut ainsi être contrôlée manuellement pour corriger toute erreur d'identification.

Comparaison avec les notices d'autorités. Au cours de cette étape, le bot va parcourir la base de données Atom pour collecter toutes les notices d'autorité qui ont été créées ultérieurement. Cette liste va être employée comme base de comparaison vis-à-vis des noms qui ont été détectés et filtrés au préalable, pour identifier les acteurs pour lesquels une notice existe déjà.

Indexation automatique. : Dans cette dernière étape, un bot va parcourir tous les fonds pour lesquels des noms ont été identifiés. Ce bot va automatiquement accéder à la page de modification de la notice, et va entrer les noms dans le champ associé aux mots-clés. Si une notice d'autorité existe déjà sur Atom, ce nom sera sélectionné par le bot. Dans le cas contraire, une nouvelle notice d'autorité sera créée.

Les sous-sections suivantes vont davantage détailler chacune de ces étapes, ainsi que les paramètres qui peuvent être sélectionnés par l'utilisateur pour l'acquisition et le traitement des données. Le schéma ci-dessous illustre le fonctionnement de l'algorithme, les sections en vert étant celles requérant une intervention humaine :

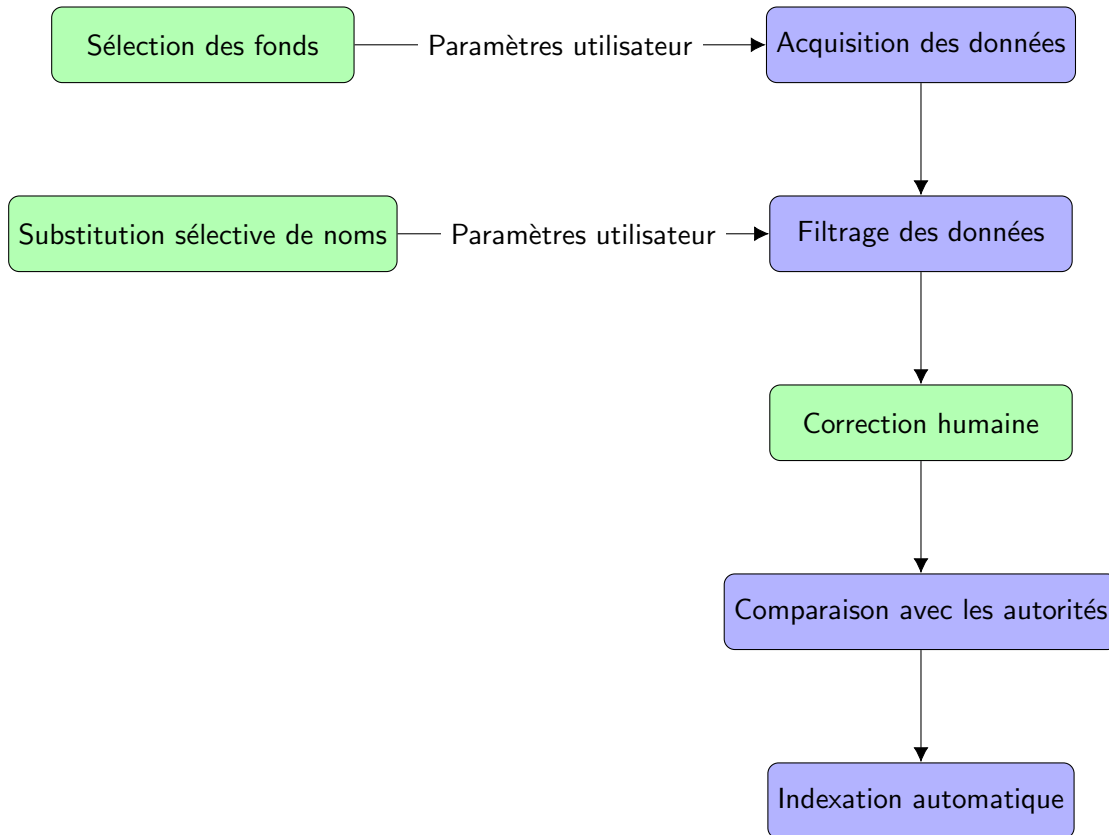


Figure 1: Schéma d'exécution de BOTticelli

2.1 Etape 0 : Connexion à Atom

Afin d'accéder aux pages relatives au fonds, il est nécessaire que le bot puisse se connecter sur le site, à l'aide d'identifiants qui lui sont fournis dans un fichier JSON. Cette étape doit être réalisée avant de lancer l'étape 1 ou l'étape 4. Elle est implémentée à l'aide de la fonction `access_atom`, qui prend comme argument le bot défini au préalable. Les étapes suivantes doivent alors être appelées avec le même bot en argument, pour s'assurer qu'il soit bien connecté.

Le fichier JSON contient deux champs, "email" et "password", auxquels doivent être associés les identifiants d'un utilisateur qui a accès à Atom. En ouvrant le fichier sur un navigateur tel que Firefox, les champs doivent apparaître de la manière suivante :

JSON	Raw Data	Headers
Save	Copy	Collapse All
Expand All	Filter JSON	
email:	"demo@example.com"	
password:	"demo"	

Figure 2: Fichier *credentials.json* ouvert sur un navigateur Web

2.2 Etape 1 : Acquisition des données

L'acquisition des données est lancée à l'aide de la fonction `browse_pages`, qui prend pour argument le bot, le tagger qui est employé pour effectuer du Named Entity Recognition (NER), ainsi qu'un mode, qui permet de définir si l'on veut uniquement collecter la correspondance ou les fonds d'archive dans leur intégralité. Les deux derniers arguments, `includeList` et `excludeList`, permet à l'utilisateur de définir s'il veut uniquement collecter certains fonds spécifiques (qu'il doit alors énumérer dans `includeList`), ou s'il souhaite en exclure, auquel cas c'est dans `excludeList` qu'il faut les mentionner. Il suffit d'indiquer la cote de ces fonds pour que ces paramètres soient pris en compte. Si les deux listes ne sont pas fournies comme arguments à la fonction, le bot va parcourir tous les fonds d'archive. A noter que ces deux paramètres sont **exclusifs**, à savoir qu'il n'est pas possible de mentionner à la fois une liste de fonds à exclure et une sélection de fonds à analyser dans un même appel de la fonction.

L'appel de la fonction se fait via une cellule du notebook Jupyter, comme le montre la capture d'écran ci-dessous :

```
[ ]: driver = Browser()
importlib.reload(botticelli)
botticelli.access_atom(driver)
tagger = SequenceTagger.load('fr-ner-wikiner-0.4.pt')
botticelli.browse_pages(driver, tagger, mode='Correspondance', includeList=['P19', 'P11', 'P25'])
```

Figure 3: Appel de la fonction `browse_pages`, avec le paramètre `includeFonds` spécifié

Cette étape est implémentée par le biais de trois fonctions, chacune accomplissant une tâche spécifique :

browse_pages. Cette fonction va parcourir successivement chaque page de fonds d'archive, et va appeler la fonction `browse_entries` sur chaque page.

browse_entries. Une fois avoir été invoquée par `browse_pages`, cette fonction va détecter tous les fonds d'archive présents sur la page. Elle va ensuite itérer sur chacun de ces fonds, afin de conduire le bot sur la page contenant le code XML du fond. A partir de ce moment-là, elle va appeler la fonction `correspondance_retrieval` pour l'analyse du code XML.

correspondance_retrieval. Cette fonction va étudier le code XML de la page du fonds, afin d'y détecter les catégories relatives à de la correspondance. L'exécution de cette fonction est divisée en plusieurs étapes :

1. En tout premier, toutes les notices mentionnant le mot "Correspondance" sont collectées, et celles se trouvant à un haut niveau dans l'arborescence sont conservées.
2. Toutes les notices enfant de celles collectées lors de l'étape précédente sont ajoutées à la liste de correspondances.
3. Le code va ensuite parcourir les notices sélectionnées pour obtenir les informations qui seront incluses dans le fichier CSV final.
4. Le titre de chaque notice est fourni au tagger pour la détection de noms de personnes, ces noms sont alors inclus dans une liste et ajoutés aux données collectées.

Une fois que toutes les notices ont été parcourues, toutes les données collectées sont agrégées dans un fichier CSV, qui contient les colonnes suivantes :

- **Titre** : Titre de la notice
- **Cote** : Cote de la notice
- **Profondeur** : Valeur indiquant la profondeur de la notice dans l'arborescence
- **Acteurs** : Liste des noms identifiés par l'algorithme dans le titre de la notice

2.2.1 Détection automatique de noms

L'étape de détection automatique de noms de personnes est réalisée à l'aide de la bibliothèque Flair, dont le [modèle de détection pour la langue française](#) atteint un F1-score de 90.61. Ce modèle est automatiquement appelé tout au long de l'exécution du code, toutefois il est nécessaire de télécharger au préalable le fichier contenant le *training dataset*. Ce fichier est téléchargeable à [cette adresse](#). Il pèse au total 1.2 GB et doit ensuite être déplacé dans le même dossier où se trouvent les scripts Python de BOTticelli.

2.3 Etape 2 : Filtrage des noms

Cette étape est divisée par l'appel de deux fonctions, `name_cleaning` et `remove_duplicates`, chacune itérant sur la totalité des données qui ont été collectées pour y effectuer des corrections automatiques. Le but de cette étape est de procéder à un nettoyage automatique des données aussi intensif que possible, pour réduire la charge de travail lors de la correction humaine. Pour cela, chacune des deux fonctions précédemment mentionnées jouent un rôle spécifique :

name_cleaning. : Cette fonction vise à détecter des duplicatas dans la liste des noms identifiés lors de l'acquisition des données. Pour cela, tous les noms actuellement sauvegardés dans le fichier CSV sont traités pour distinguer ceux qui sont très similaires. Afin de pouvoir regrouper toutes les variantes d'un même nom et de permettre une marge en cas d'erreur de frappe, les lettres de chaque nom sont tout en premier lieu rendues minuscules et triées dans l'ordre alphabétique. Cette liste de lettres est alors employée comme base de comparaison. Effectuer ainsi la comparaison permet d'identifier deux noms dont le nom de famille et le prénom sont inversés (par exemple *Paul Budry* et *BUDRY Paul*) comme étant associés à la même personne. Afin de palier à des erreurs de frappe mineures lors de la saisie de l'entrée, deux noms sont aussi considérés comme étant identiques s'il y a une différence dans la séquence de lettres qui leur est respectivement associée. Le schéma ci-dessous donne un exemple d'exécution de cette fonction :

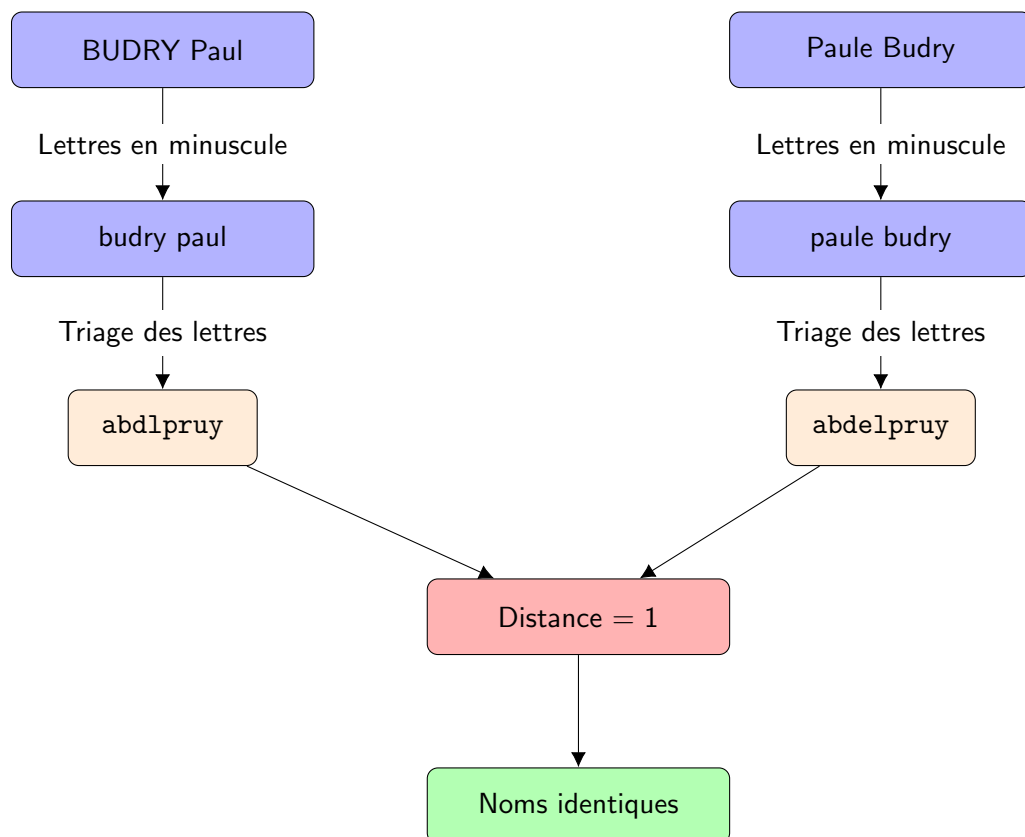


Figure 4: Exemple de comparaison de noms

La fonction prend aussi en argument `changeDict`, qui permet de spécifier des substitutions à effectuer systématiquement lorsqu'un nom est détecté. Ceci permet de corriger des erreurs qui ont été faites régulièrement, par exemple sur l'orthographe du nom d'un auteur ou des noms qui ont été entrés avec des initiales.

remove_duplicates. : Cette fonction va parcourir le fichier CSV et enlever toute entrée qui apparaît plus qu'une fois dans la base de données. Elle va aussi considérer les noms détectés dans une notice, et les enlever des notices enfants si ces noms se retrouvent dans leurs listes respectives. Le but est d'associer chaque notice d'autorité au niveau le plus supérieur les concernant, et d'éviter ainsi de créer une redondance au niveau de l'indexation.

Les cellules suivantes permettent de faire appel aux deux fonctions relatives à cette étape :

```

[9]: importlib.reload(botticelli)
    entryDF = pd.read_csv('CLSR_Correspondance.csv')
    changeDict = {'Charles-Ferdinand Ramuz' : 'Charles Ferdinand Ramuz', 'I. Strawinsky' : 'Igor Strawinsky'}
    botticelli.name_cleaning(entryDF, changeDict)

[10]: importlib.reload(botticelli)
    entryDF = pd.read_csv('CLSR_Correspondance.csv')
    botticelli.remove_duplicate_names(entryDF)
  
```

Figure 5: Appel des fonctions `name_cleaning` et `remove_duplicates`

Cet exemple illustre aussi comment le paramètre `changeDict` doit être défini, afin de pouvoir être pris en considération.

2.4 Etape 3 : Correction humaine

Cette étape vient compléter le filtrage des données, en corrigeant manuellement les anomalies qui sont toujours présentes dans le fichier CSV. Un utilisateur doit parcourir le fichier et apporter des modifications aux noms inscrits dans la colonne **Acteurs**, qu'il s'agisse d'ôter des noms qui ont été faussement identifiés comme étant des personnes, ou corriger des fautes de frappe qui ont pu échapper au filtrage des noms.

Une fois avoir effectué les modifications, il est simplement nécessaire de sauvegarder le fichier. Ce dernier sera relu par les fonctions appelées dans les étapes suivantes, et tout changement sera ainsi pris en compte.

2.5 Etape 4 : Comparaison avec les notices d'autorités



Atom permet d'ajouter des notices d'autorité, qui peuvent ensuite être associées à des fonds d'archive en tant que mots-clés. Le but de cette étape est de comparer chaque nom, après les étapes d'acquisition et de correction, aux noms actuellement présents dans la liste des notices, afin de voir si une notice correspondante est déjà présente. De plus, si un nom qui n'est pas présent initialement dans les notices d'autorité apparaît plus d'une fois dans les données, il est nécessaire pour l'algorithme de savoir que la notice aura été créée en cours de route, pour éviter de se retrouver avec des notices identiques.

Trois fonctions ont été implémentées pour satisfaire ces besoins : `autorite_retrieval`, `autorite_comparison` et `find_previous_appearance`. Leurs rôles respectifs sont détaillés dans les paragraphes suivants :

autorite_retrieval. Cette première fonction va lancer un bot, dont le rôle est de parcourir les notices d'autorité actuellement présentes sur Atom et de regrouper leurs noms dans une liste, qui servira par la suite de canevas pour les comparaisons.

autorite_comparison. De la même manière que la fonction `name_cleaning` dans la deuxième étape, cette fonction va procéder par une comparaison individuelle de chacun des noms présents dans la base de données avec les notices d'autorité qui viennent d'être extraites, afin de voir quels noms peuvent être associés à une notice préexistante. Les notices associées à chaque nom sont inscrites dans une nouvelle colonne, **Autorités**, tandis que la colonne **Found** est utilisée pour indiquer s'il s'agit d'une notice qui est déjà présente sur Atom : un booléen `True` indique qu'il y a bel et bien une notice préexistante, tandis qu'une valeur `False` signifie qu'une nouvelle notice devra être créée.



find_previous_appearance. Cette dernière fonction va parcourir les deux colonnes qui ont été ajoutées précédemment dans le fichier CSV, pour détecter les notices actuellement inexistantes sur Atom qui seront créées en cours de route. Ainsi, si un nom se retrouve plus d'une fois dans le fichier et qu'il n'existe pas de notice associée, un booléen `False` sera associé à la première occurrence de ce nom, et un booléen `True` aux suivantes.

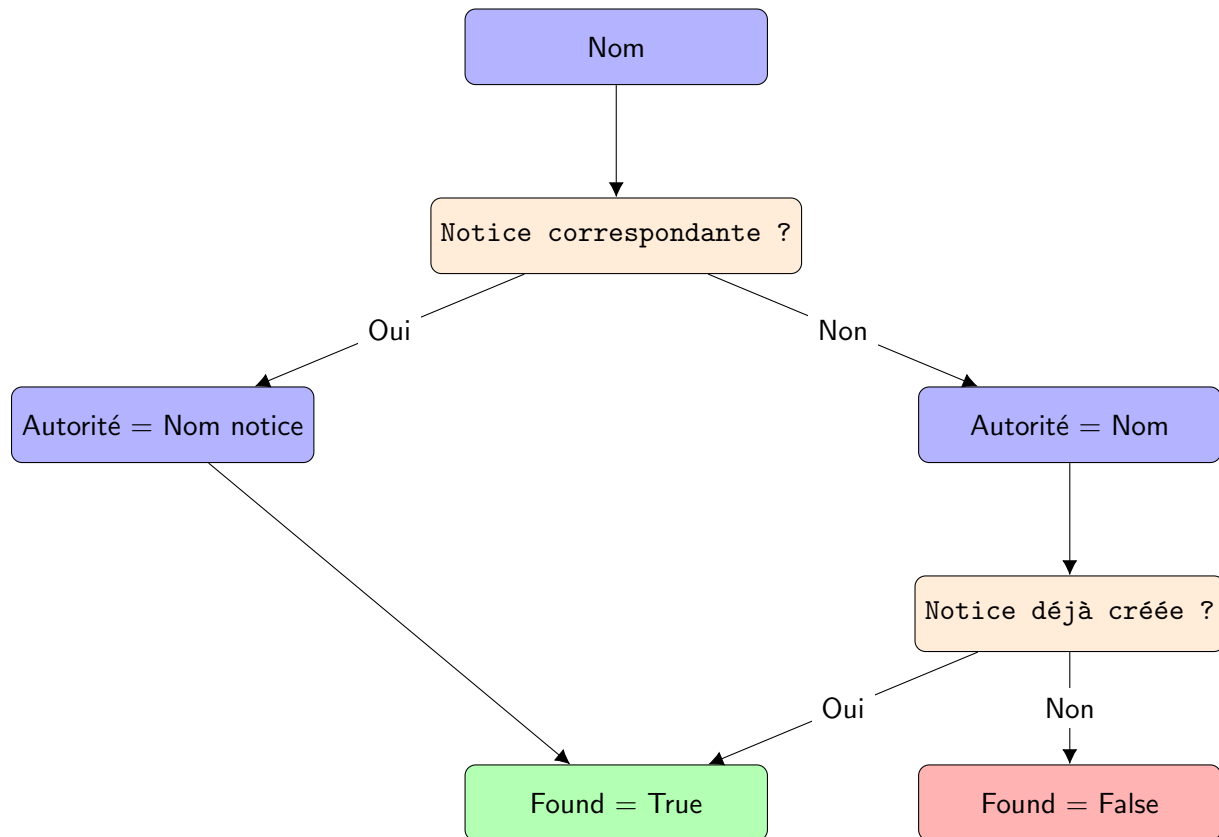


Figure 6: Affectation des valeurs des colonnes **Autorités** et **Found**

Les cellules suivantes sont employées pour appeler les fonctions relatives à cette étape :

```

[5]: importlib.reload(botticelli)
      driver = Browser()
      autoriteList = botticelli.autorite_retrieval(driver)

[6]: importlib.reload(botticelli)
      entryDF = pd.read_csv('CLSR_Correspondance.csv')
      botticelli.autorite_comparison(entryDF, autoriteList)

[7]: importlib.reload(botticelli)
      entryDF = pd.read_csv('CLSR_Correspondance.csv')
      botticelli.find_previous_appearance(entryDF)
  
```

Figure 7: Appel des fonctions `autorite_retrieval`, `autorite_comparison` et `find_previous_appearance`

2.6 Etape 5 : Indexation automatique

Durant cette dernière étape, le bot va de nouveau parcourir les fonds d'archives pour y entrer le nom des notices d'autorité, qui ont été identifiées et corrigées dans la colonne **Autorites** du fichier CSV. Le bot va visiter successivement chaque fond et va entrer les noms indiqués dans la colonne **Autorites**. Si le nom est associé à un booléen **True** dans la colonne **Found**, le bot va sélectionner le nom suggéré par Atom. Dans le cas contraire, à savoir un booléen **False**, une nouvelle notice d'autorité sera générée sur Atom.

Le bot va itérer au travers de toutes les entrées présentes dans le fichier CSV. Pour chacune d'entre elles, il va suivre l'algorithme suivant pour effectuer l'indexation automatique :

Recherche de l'entrée. Le bot va employer la barre de recherche d'Atom pour trouver l'entrée associée aux données définie dans la ligne du fichier CSV traité. Cette étape doit permettre au bot de sélectionner sans ambiguïté la bonne entrée. Pour cela, deux options de recherche sont actuellement définies, permettant au bot d'avoir une solution de back-up si la première ne retourne pas de résultats satisfaisants. La première recherche consiste à simplement entrer la cote de l'entrée, tandis que le titre est ajouté pour la seconde.



Accès à la page d'édition et entrée successive des noms. Une fois que le bot se trouve sur la page de données de l'entrée, il accède directement aux champs d'édition et va entrer successivement les noms associés à cette entrée, listés dans le champ **Autorites** du fichier CSV.

Suggestions automatiques. Lorsqu'un nom est entré dans le champ des mots-clés, Atom va offrir une liste de suggestions si l'entrée est similaire à des noms déjà présents dans la liste. La colonne **Found**, définie lors de l'étape précédente, permettra ainsi au bot de savoir s'il doit sélectionner le nom parmi les suggestions offertes par Atom, ou s'il doit créer une nouvelle notice. En fonction de la valeur du booléen, le bot va définir une pause de 2 secondes après avoir entré le nom pour attendre la suggestion dans le cas d'une valeur **True**, et ne va pas attendre dans le cas contraire.



Sauvegarde des modifications. Une fois que tous les noms ont été entrés, le bot presse sur le bouton **Sauvegarder** afin que les mots-clés soient dorénavant associés à la notice.

La cellule suivante doit être exécutée dans le notebook Jupyter afin d'appeler la fonction d'indexation :

```
[ ]: driver = Browser()
     entryDF = pd.read_csv('CLSR_Correspondance.csv')

     importlib.reload(botticelli)
     botticelli.access_atom(driver)
     botticelli.indexation_automatique(driver, entryDF)
```

Figure 8: Appel de la fonction `indexation_automatique`

3 Présentation de BOTorités

L'exécution de BOTticelli va mener à la potentielle création de nombreuses notices d'autorité, pour lesquelles aucune information n'est fournie par défaut. C'est pourquoi un autre bot, BOTorités, a été développé de manière à pouvoir ajouter de manière automatique certaines informations basiques, qui doivent être annexées à chaque notice.

L'exécution de BOTorités se fait via la cellule suivante :

ETAPE 6 : Annotation des notices d'autorité

```
[ ]: driver = Browser()
importlib.reload(botorites)
botticelli.access_atom(driver)
botorites.modify_new_autorites(driver)
```

Figure 9: Cellule d'exécution de BOTorités

Cette cellule appelle la fonction `modify_new_autorites`, dont le rôle est de parcourir une à une les pages de notices d'autorités présentes sur Atom et de les fournir comme paramètre à la seconde fonction, nommée `browse_autorites`. Cette dernière est chargée d'identifier, pour chaque notice, l'URL de la page de modification et de mener le bot à ces pages de manière itérative afin que la dernière fonction, `annotate_autorites`, puisse entrer les informations nécessaires et les sauvegarder.

Trois champs sont remplis pour chaque notice d'autorité, qui sont les suivants :

Type d'entité. Les acteurs étant indexé par BOTticelli correspondant uniquement à des personnes, la valeur *Personne* est sélectionnée par BOTorités.

Identifiant de notice d'autorité. L'identifiant de chaque notice d'autorité suit un format uniforme, à savoir que le nom de famille est écrit, suivi par le prénom, et que chaque nom est lié par un underscore à la place d'un espace.

Entretien du dépôt. Chaque notice d'autorité est associée au Centre des littératures en Suisse romande.

Pour éviter que des notices d'autorité représentant des collectivités soient modifiées par erreur, le bot a été programmé de manière à détecter si le champ *Type d'entité* contient déjà la valeur *Collectivité*. Dans ce cas-là, aucune modification ne sera apportée à cette notice.

Il est aussi possible de modifier le script de BOTorités pour y ajouter d'autres champs à remplir de manière automatique. Pour ce faire, il suffit de suivre la même logique que celle employée pour les deux lignes de code suivantes, qui remplissent le champ pour l'identifiant de notice d'autorité :

```
# Modification de l'identifiant de notice d'autorité  
idAutorite = autorite.replace(' ', '_')  
driver.type(idAutorite, id='descriptionIdentifiant')
```

Figure 10: Lignes de code employées pour remplir l'identifiant de notice d'autorité

Ainsi, si vous souhaitez pouvoir remplir un autre champ, pour lequel il est nécessaire de taper du texte, il suffit de trouver l'identifiant associé à sa balise dans le code HTML et de lui fournir en paramètre le texte à y entrer.



4 Annexe : Guide d'utilisation de Jupyter

Jupyter est une application Web permettant d'écrire et d'exécuter des scripts Python, à l'aide d'un environnement de développement Conda. Ces deux outils sont donc essentiels pour l'exécution du script de BOTticelli, la section suivante va donc offrir une brève démarche pour leur mise en place.

4.1 Création d'un environnement Conda

Un environnement Conda est un dossier créé localement dans une machine afin d'y stocker tous les packages installés par un utilisateur. En ayant un tel environnement, il est alors possible d'exécuter des scripts et des notebooks Jupyter.

Afin de pouvoir bénéficier des commandes Conda dans le terminal, il faut tout d'abord installer Anaconda. [Le guide d'installation du site officiel](#) détaille chaque étape nécessaire pour l'installation, et [cette page](#) explique la marche à suivre pour créer et activer un environnement. La commande la plus importante à maîtriser est `conda activate`, qui permet de lancer l'environnement à l'aide du terminal. Le nom de l'environnement est défini lors de sa création, à l'aide de la commande `conda create`. L'utilisation de cette commande est détaillée dans le tutoriel présent sur la page officielle de Conda.

4.2 Installation de packages

Python est un langage de programmation open-source, dont les fonctionnalités sont regroupées dans des *libraries* développées par la communauté scientifique. Ces packages peuvent être facilement installés à l'aide de l'utilitaire *pip*, dont [le site web](#) permet de rechercher un package et d'obtenir la commande à entrer dans le terminal pour l'installer.

Avant d'installer un package, il est nécessaire de s'assurer d'avoir activé l'environnement Conda au préalable. Ainsi, le package sera ajouté à l'environnement et ne devra pas être réinstallé dans le futur. En cas de doute sur les packages d'ores et déjà présents dans l'environnement, une fois avoir activé l'environnement, la commande `conda list` fournit une liste exhaustive des packages localement installés. Par exemple, pour installer le package `webbot`, qui est employé pour implémenter les bots décrits précédemment, la commande suivante doit être entrée dans le terminal, après avoir activé l'environnement Conda :

```
pip install webbot
```

4.3 Installation de Jupyter Lab

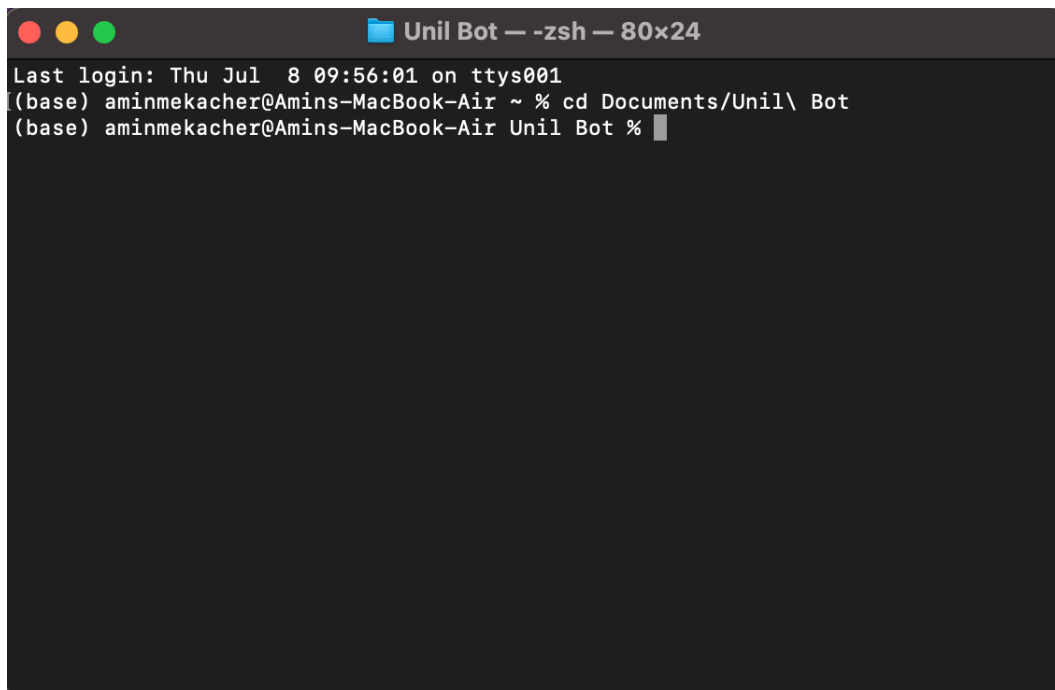
Jupyter Lab est un éditeur, permettant d'ouvrir les scripts et Notebooks dans un serveur local, de les modifier et d'exécuter *on the fly* les cellules des notebooks. Jupyter Lab offre une interface utilisateur, permettant ainsi de simplifier l'ouverture de fichier et le contrôle de l'exécution. Toutefois, n'étant pas nativement présent dans l'utilitaire Anaconda, il faut l'installer au préalable. Une fois avoir activé l'environnement, il suffit de lancer la commande suivante :

```
conda install -c conda-forge jupyterlab
```

4.4 Exécution de Jupyter

L'exécution de Jupyter se fait via le terminal et permet de lancer un serveur local sur un navigateur Web, hébergeant l'environnement de travail. Après avoir initialisé un environnement de développement Conda, comme spécifié dans les sous-sections précédentes, les étapes à suivre pour lancer Jupyter sont les suivantes :

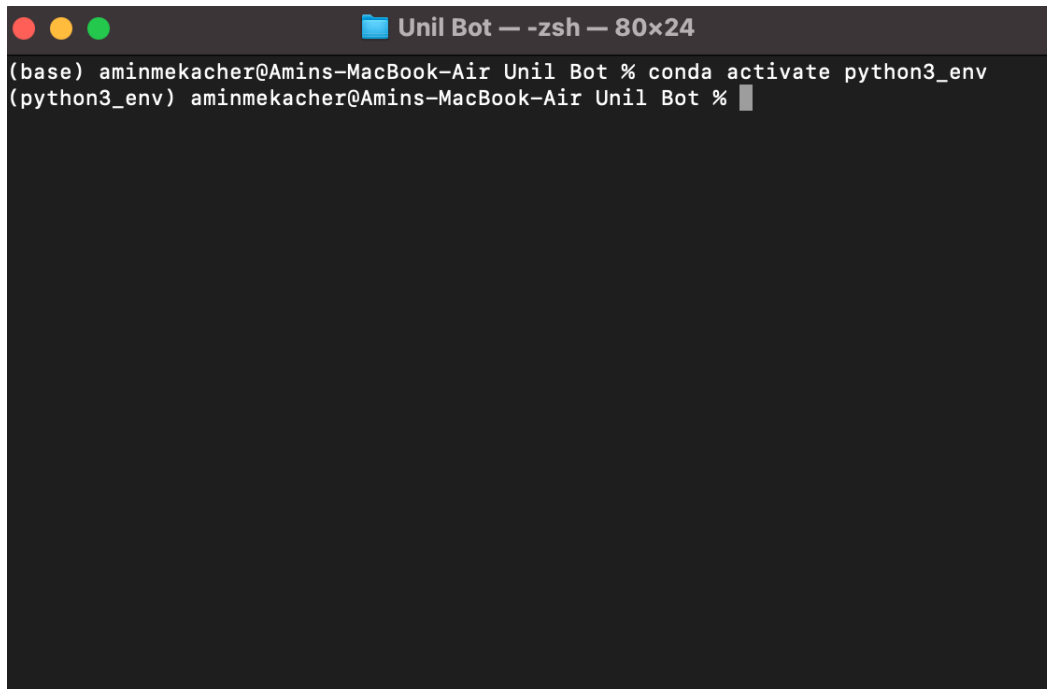
Déplacement dans le dossier de travail. Après avoir lancé le terminal, utilisez la commande `cd` pour vous rendre dans le dossier dans lequel vous souhaitez stocker vos fichiers de travail. La capture d'écran ci-dessous en est un exemple, dans lequel le dossier *Unil Bot* est employé comme dossier de travail.

A screenshot of a macOS terminal window titled "Unil Bot — -zsh — 80x24". The terminal shows the following text: "Last login: Thu Jul 8 09:56:01 on ttys001", "(base) aminmekacher@Amins-MacBook-Air ~ % cd Documents/Unil\ Bot", and "(base) aminmekacher@Amins-MacBook-Air Unil Bot %". The prompt changes from "~" to "Unil Bot %" after the directory change command is executed.

```
Last login: Thu Jul 8 09:56:01 on ttys001
(base) aminmekacher@Amins-MacBook-Air ~ % cd Documents/Unil\ Bot
(base) aminmekacher@Amins-MacBook-Air Unil Bot %
```

Figure 11: Déplacement dans le dossier de travail

Activation de l'environnement Conda. Pour activer l'environnement, la commande à employer est `conda activate envname`, où *envname* est le nom attribué à l'environnement Conda lors de sa création. La commande ci-dessous permet, par exemple, d'activer l'environnement *python3_env*. Pour s'assurer que la commande a été effectuée correctement, le terminal indique dorénavant le nom de l'environnement entre parenthèses, au tout début de la ligne d'invitation :

A terminal window titled "Unil Bot — -zsh — 80x24" with standard macOS window controls (red, yellow, green buttons). The terminal shows the command `conda activate python3_env` being executed. The prompt changes from `(base)` to `(python3_env)`.

```
(base) aminmekacher@Amins-MacBook-Air Unil Bot % conda activate python3_env
(python3_env) aminmekacher@Amins-MacBook-Air Unil Bot %
```

Figure 12: Activation de l'environnement Conda

Lancement de Jupyter. La commande `jupyter lab` permet de lancer le serveur local sur votre navigateur internet. A partir du moment où l'application est en train de tourner, le terminal est dédié à Jupyter et ne doit par conséquent pas être fermé jusqu'à la fin de l'utilisation de Jupyter.

```
Unil Bot — python3.8 ~/anaconda3/envs/python3_env/bin/jupyter-lab — 8...
(python3_env) aminmekacher@Amins-MacBook-Air Unil Bot % jupyter lab
[I 14:59:27.391 LabApp] The port 8888 is already in use, trying another port.
[I 14:59:27.392 LabApp] The port 8889 is already in use, trying another port.
[I 14:59:27.404 LabApp] JupyterLab extension loaded from /Users/aminmekacher/anaconda3/envs/python3_env/lib/python3.8/site-packages/jupyterlab
[I 14:59:27.404 LabApp] JupyterLab application directory is /Users/aminmekacher/anaconda3/envs/python3_env/share/jupyter/lab
[I 14:59:28.747 LabApp] Serving notebooks from local directory: /Users/aminmekacher/Documents/Unil Bot
[I 14:59:28.747 LabApp] Jupyter Notebook 6.1.4 is running at:
[I 14:59:28.747 LabApp] http://localhost:8890/?token=c5eadd4b2681a1fc21db1d86cfb2c530cb56758563f70fc0
[I 14:59:28.747 LabApp] or http://127.0.0.1:8890/?token=c5eadd4b2681a1fc21db1d86cfb2c530cb56758563f70fc0
[I 14:59:28.747 LabApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 14:59:28.757 LabApp]

To access the notebook, open this file in a browser:
    file:///Users/aminmekacher/Library/Jupyter/runtime/nbserver-3831-open.html
Or copy and paste one of these URLs:
    http://localhost:8890/?token=c5eadd4b2681a1fc21db1d86cfb2c530cb56758563f70fc0
```

Figure 13: Lancement de Jupyter