

# Programmation Spécialité Python

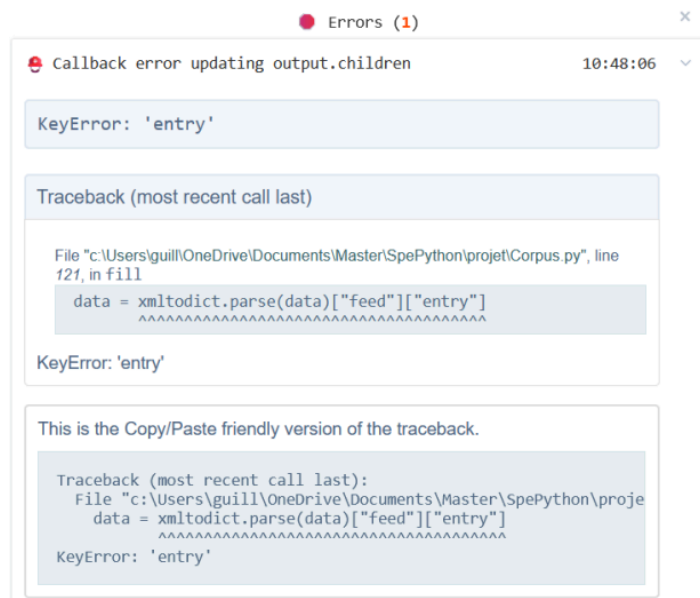
## Elyes KHALFALLAH - Matthieu GUILLEMIN

[https://github.com/MattGuil/ProjetSpePython\\_GUILLEMIN-KHALFALLAH.git](https://github.com/MattGuil/ProjetSpePython_GUILLEMIN-KHALFALLAH.git)

*Avant de commencer, nous aimerions préciser que nous n'allons pas expliquer le code rendu dans ce rapport, mais plutôt nos choix d'implémentation et raisonnements derrière nos décisions: les commentaires présents suffisent pour bien comprendre leur fonctionnement.*

**NOUS AIMERIONS AUSSI PRÉCISER QU'IL EXISTE UN BUG FAISANT QUE L'API CUEILLANT LES DOCUMENTS UTILISÉS PAR LE PROGRAMME NE FONCTIONNE PLUS. CECI EST TOTALEMENT HORS DE NOTRE CONTRÔLE, NOUS A GRANDEMENT GÊNÉ/RALENTIT PENDANT LA RÉALISATION DU PROJET, ET POURRAIT VOUS EMPÊCHER DE LANCER NOTRE CODE PENDANT VOTRE CORRECTION.**

**DANS CE CAS, ATTENDEZ 15-45 MINUTES ET RÉESSEYER. NOUS INSISTONS, CECI EST COMPLÈTEMENT HORS DE NOTRE CONTRÔLE, ET C'EST LA SOLUTION À LAQUELLE NOUS AVONS DÛ RECOURIR À DE NOMBREUSES REPRISES AU COURS DE NOTRE PROJET.**



The screenshot shows an error window titled "Errors (1)" with a close button. The error message is "Callback error updating output.children" with a timestamp of "10:48:06". Below this, the error details are shown: "KeyError: 'entry'". A traceback is provided, showing the file path "c:\Users\guill\OneDrive\Documents\Master\SpePython\projet\Corpus.py", line 121, in fill, with the code snippet "data = xmldict.parse(data)["feed"]["entry"]". Below the traceback, there is a section titled "This is the Copy/Paste friendly version of the traceback." which contains the same traceback information in a more readable format.

```
Callback error updating output.children 10:48:06

KeyError: 'entry'

Traceback (most recent call last)

File "c:\Users\guill\OneDrive\Documents\Master\SpePython\projet\Corpus.py", line 121, in fill
    data = xmldict.parse(data)["feed"]["entry"]
          ~~~~~^~~~~~

KeyError: 'entry'

This is the Copy/Paste friendly version of the traceback.

Traceback (most recent call last):
  File "c:\Users\guill\OneDrive\Documents\Master\SpePython\proje
    data = xmldict.parse(data)["feed"]["entry"]
          ~~~~~^~~~~~
KeyError: 'entry'
```

## Introduction :

Au cours de ce projet, nous avons découvert le fonctionnement interne d'un moteur de recherche complet, notre objectif étant d'en créer un en utilisant des API de collecte de documents capables de récupérer des articles et leurs données sur Reddit et Arxiv.

Pour notre première version du programme (v1.0.0), nous voulons rechercher un nombre donné de documents sur un sujet, puis extraire les données utiles (titre, auteur(s), contenu, etc) de ces pages. Après avoir stocké ces données, nous disposerons d'un corpus prêt à l'emploi pour ce sujet recherché.

Dans la deuxième version du programme (v2.0.0), nous ajouterons des algorithmes permettant de rechercher dans le corpus des mots-clés donnés et de calculer la pertinence de leurs documents.

Dans notre troisième et dernière version (v3.0.0), nous créerons une interface visuelle pour faciliter l'utilisation de notre moteur de recherche.

## Spécifications du programme :

Le programme suit bien les indications données dans les TD 3 à 10 :

- Il commence par saisir un thème et un ou plusieurs mots-clés pour la recherche, ainsi que le nombre demandé de documents
- La recherche et la collecte de documents commence, jusqu'à ce que le nombre total de documents souhaité soit atteint (partagé moitié-moitié entre Arxiv et Reddit)
- Ensuite, les documents sont "nettoyés". Il s'agit de supprimer tous les documents dont le contenu pertinent contient moins de 20 caractères (un effet secondaire de cette opération est que les documents peu présentés mais contenant des fichiers, tels que des PDF ou des images, sont rapidement supprimés).
- Après le nettoyage, toutes les informations restantes sur les documents sont rassemblées dans une liste et sauvegardées dans un fichier pckle.
- Maintenant, un vocabulaire est créé en supprimant tous les symboles spéciaux (c'est à dire tout sauf l'alphabet, les lettres avec accents, les apostrophes, et autres caractères pertinents pour des mots), en concaténant tous les textes en utilisant ` comme séparateur, puis en itérant sur cette longue chaîne, en ajoutant chaque mot nouvellement vu à une variable de vocabulaire (le tout se fait donc en un passage du vocabulaire).
- Une fois le vocabulaire créé, TF et TFxIDF sont calculés en conjonction avec les mots-clés donnés
- Enfin, les résultats sont renvoyés sous forme d'une liste de documents triés en fonction de leurs similarité calculée, de la plus élevée à la plus faible

## Analyse d'Implémentation :

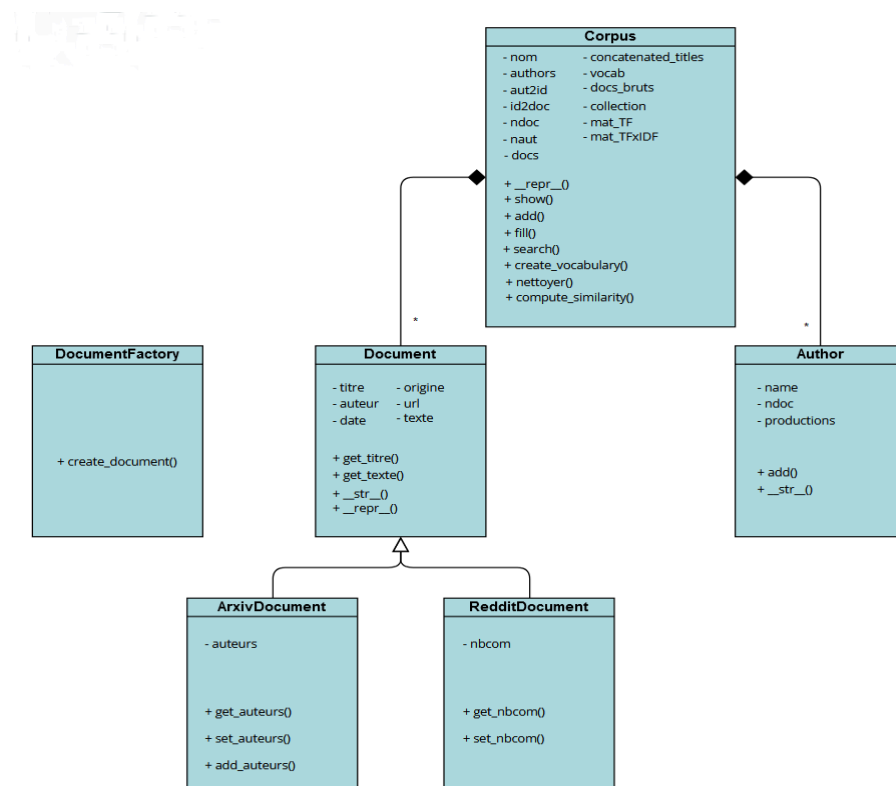
### Base du programme :

Pour l'ensemble de ce projet, nous avons utilisé de la programmation orientée objet. Le fait de traiter un si grand nombre de documents, d'auteurs, de corpus, n'avait de sens que si l'on utilisait la programmation orientée objet. Facilitant la création, la modification et la suppression de nombreuses instances d'objets, cette décision était cruciale pour notre projet.

### Environnement Python :

En ce qui concerne l'environnement, il a été totalement migré vers un fichier requirements.txt disponible en téléchargement dans la dernière version du code sur GitHub. Il contient toutes les bibliothèques essentielles nécessaires au fonctionnement des programmes, notamment : praw (recherche de documents), numpy, pandas (création de dataframes), ipywidgets (affichage de la v2), dash (affichage de la v3), et bien d'autres encore...

### Diagramme de classes :



## Conception du projet :

### Méthodologie de Travail :

Pour l'ensemble de ce projet, nous avons travaillé dans un format de Pair-Programming : alors que l'un est devant l'ordinateur de Matthieu (ce qui explique pourquoi une majorité des commits sont de son compte GitHub), l'autre supervise le code, et commente sur les avancements tout en aidant en proposant des idées/méthodes d'avancements. Pendant les vacances, nous avons travaillé ensemble par appels Discord en utilisant des partages d'écran.

### Explication de quelques choix d'implémentation :

Au cours du projet, nous avons pris quelques décisions basées sur nos résultats, notre expérience du code, ainsi que sur nos goûts personnels. Dans l'ensemble, nous avons bien suivi le sujet donné, mais il y a quelques exceptions :

- Nous avons pris la décision de renverser complètement ce que les mots-clés (pas le sujet, mais seulement les mots-clés) recherchent : au lieu du texte d'un document, ils recherchent le nom. Nous avons pris cette décision parce qu'après avoir lu de nombreux exemples, nous avons remarqué que de plein d'auteurs (en particulier sur Arxiv) incluent des mots-clés importants dans les titres de leurs posts, et non dans le corps du post. En effet, les auteurs incluent généralement un titre, une description *très* brève et un PDF de leurs essais et travaux, document que n'analyse pas notre application
- Nous avons également décidé d'implémenter une recherche simultanée de mots-clés et de sujets, car nous pensons que cela améliore la clarté tout en n'ayant peu ou pas d'impact sur les fonctionnalités de l'utilisateur (comme quoi un utilisateur ne changera pas de mots-clés sans être forcé à refaire une recherche).

### Bugs et bizarreries :

Nous avons aussi rencontré une pléthore de bugs... simples, complexes, résolus ou non, nous avons pu les dépasser pour la plupart. Notre ennemi juré, cependant, était le très agaçant et pénible bug mentionné dans la première page de ce rapport, qui nous arrêtaient complètement à chaque fois que nous essayions d'exécuter le code. Nous pensons tous les deux qu'il s'agit d'un bug occasionnel de l'API pour la recherche de documents. Dans la capture d'écran de la première page, l'erreur semble toujours concerner "entry", l'objet récupéré par l'API Arxiv.

Cependant, en revenant au même code 30 minutes plus tard après sans changer une seule ligne de code, tout fonctionne comme par magie. Nous avons passé des heures à essayer de comprendre et de résoudre pourquoi cela se produit, à la fois en séance de TD ainsi qu'à la maison, mais nous ne sommes jamais parvenus à des conclusions concrètes. Nous avons décidé d'aller de l'avant, en espérant simplement que cela ne se reproduise plus.

Nous avons également rencontré des bugs plus mineurs, tels que le "Nombre de Documents" qui était toujours supérieur ou inférieur de 1 à la réalité, que nous avons pu résoudre dans des versions plus avancées du programme en revenant sur la manière dont la valeur était calculée. Un autre bug que nous avons eu est que `self.documents`, la variable contenant tout le contenu des documents sous la forme d'une très longue chaîne (utilisée pour le concordancier), s'est transformée de manière inattendue en une instance d'objet `ArxivDocument` (et toujours `ArxivDocument`, jamais `RedditDocument`). A ce jour, nous ne savons pas comment ni pourquoi cela s'est produit, mais nous avons pu le résoudre en redéfinissant la façon dont `self.documents` est créé.

Le dernier bug dont nous parlerons concerne la fonction de recherche dans la version 3 du programme. Si un paramètre était modifié, qu'il s'agisse d'un sujet de recherche, de mots clés, d'un appui sur le bouton "recherche" ou d'une modification du nombre de documents souhaités, une recherche se produisait immédiatement. Il a fallu fouiller un peu dans le code pour comprendre que nos valeurs de recherche se mettaient à jour trop rapidement, et pas seulement lorsque l'on appuyait sur le bouton de recherche. Les solutions de ce type sont évidentes en théorie, mais en pratique nous ont pris pas mal de temps.

Les bugs énoncés ne représentent pas la totalité de ceux auxquels nous avons fait face, mais seulement quelques exemples que nous avons décidé d'inclure.

### **Usage du programme :**

L'utilisation de notre programme est plutôt facile :

- **Pour la v1 :** il n'y a qu'à changer le sujet souhaité en début du fichier `main.py`, et lancer le code. Les documents résultants apparaîtront en terminal
- **Pour la v2 :** il faut rentrer dans le fichier `main.ipynb`, et lancer en ordre les sections de code (il n'y en a que deux). A partir de là, une interface utilisateur apparaîtra. Il faudra écrire son thème souhaité, nombre (potentiel) de documents souhaité, et appuyer sur le bouton pour effectuer la recherche.  
*Entre chaque utilisation de la v2, il faut restart le kernel !*

- **Pour la v3 :** après avoir lancé le fichier `app_dash.py`, il faudra accéder au lien <http://127.0.0.1:8000/> (le localhost avec port prédéfini par le code). Arrivé à la page d'accueil, il faudra insérer son thème souhaité, des éventuels mot clés (optionnels), et le nombre de documents souhaités. Après appuyer sur le bouton, la liste des documents triés par ordre de pertinence sera visible.

### **Testes Unitaires (ou, manque de) :**

Nous n'avons pas inclus de tests unitaires dans notre programme par raison de priorités de développement. Nos tests à la main, par contre, ont été nombreux.

### **Maintenance et Évolutions possibles :**

En ce qui concerne les améliorations, nous sommes certains que notre code est fonctionnel, mais pas optimal. Au cours des semaines où nous avons travaillé dessus, nous avons dû trouver de nombreuses solutions de contournement pour parvenir aux solutions nécessaires à un produit final potable. Cela a rendu notre programme assez lourd (pas en taille d'octets, mais en lignes code) et convoluté dans son fonctionnement. Même nous admettons que nous nous perdons de temps en temps en cours de route. C'est pourquoi l'optimisation du code de notre programme serait une première étape importante pour l'améliorer considérablement.

Un autre détail à aborder serait les calculs de Similarité. Plutôt ambiguë dans son fonctionnement, nous avons eu beaucoup de mal à comprendre comment implémenter correctement cette fonction, même avec l'aide des bibliothèques recommandées. Au final, nous disposons bien de matrices TF et TFxIDF fonctionnelles et à valeurs (en moyenne) sensés, bien que leurs résultats ne soient pas toujours les plus précis.

### **Conclusion :**

En conclusion, ce projet a été une exploration du développement d'un moteur de recherche à partir de zéro en utilisant des API de collecte de documents. Les trois versions du programme ont couvert la recherche de documents, l'analyse du corpus, et l'implémentation d'une interface visuelle. Malgré des défis tels que des bugs liés à l'API de recherche, la programmation orientée objet a facilité la gestion efficace de documents, d'auteurs et de corpus, tandis que le format de Pair-Programming a favorisé une collaboration réussie.

Pour la maintenance et les évolutions potentielles, l'optimisation du code est cruciale pour améliorer l'efficacité du programme, tandis que des ajustements peuvent être apportés aux calculs de similarité. Malgré ces opportunités d'amélioration, le projet a atteint son objectif principal en développant un moteur de recherche fonctionnel pour des documents spécifiques sur Reddit et Arxiv.