

**RAPPORT DE PROJET**  
Programmation de spécialité – Python  
Université Lumière Lyon 2  
2024-2025

# Mon moteur de recherche

**GitHub :** <https://github.com/blfmelissa/projet-python>

**Réalisé par :**  
Melissa Bouloufa  
Pierrick Dennemont

**Encadré par :**  
Julien Velcin  
Francesco Amato

## Table des matières

<b>Présentation du projet</b> .....	<b>3</b>
<b>Analyse</b> .....	<b>4</b>
Environnement de travail et justifications.....	4
Données identifiées et spécifications .....	4
Organisation des fichiers et des classes .....	5
<b>Conception</b> .....	<b>7</b>
Répartition des tâches entre les binômes.....	7
Exemple d'utilisation.....	7
<b>Validation</b> .....	<b>11</b>
Quelques tests unitaires.....	11
Tests Globaux .....	11
<b>Maintenance</b> .....	<b>13</b>
Pistes d'amélioration.....	13
<b>Conclusion</b> .....	<b>13</b>

# 1. Présentation du projet

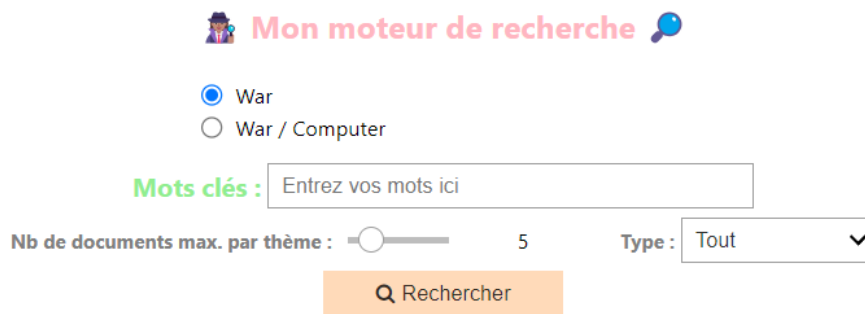
Notre programme consiste en un petit moteur de recherche capable d'explorer des documents issus de ressources en ligne. Pour cela, nous avons choisi les API de :

- **Hacker News** : une plateforme similaire à Reddit permettant des échanges sur les nouvelles technologies
- **The Guardian** : un journal britannique réputé



Un **corpus** est une collection de documents (issus de Hackers News et de The Guardian) traitant un thème choisi ("War", dans notre cas).

Le programme collecte, structure et stocke les données des deux sources dans un corpus. Les documents sont classés par source. L'utilisateur peut effectuer des recherches ciblées, et le programme trie les résultats en fonction des mots-clés saisis et affiche ceux ayant le plus grand taux de similitude (calculé de manière optimisée).



The screenshot shows a web interface for a search engine. At the top, there is a title 'Mon moteur de recherche' with a magnifying glass icon. Below the title, there are two radio buttons: 'War' (selected) and 'War / Computer'. A green label 'Mots clés :' is followed by a text input field containing 'Entrez vos mots ici'. Below this, there is a slider for 'Nb de documents max. par thème' set to 5, and a dropdown menu for 'Type' set to 'Tout'. At the bottom, there is an orange button with a magnifying glass icon and the text 'Rechercher'.

## 2. Analyse

### A. Environnement de travail et justifications

Le projet sera développé en Python. Nous utiliserons plusieurs bibliothèques adaptées à nos besoins, telles que :

- requests : pour interagir avec les API et récupérer les données
- BeautifulSoup : pour parser et extraire des données HTML (web scraping)
- pandas : pour structurer et manipuler les données collectées
- scipy.sparse : pour manipuler des matrices creuses
- nltk : pour gérer les mots vides (stopwords)
- pickle : pour sauvegarder et charger des données structurées

### B. Données identifiées dans les spécifications

Pour construire notre moteur de recherche, les données nécessitent d'être organisées de manière structurée.

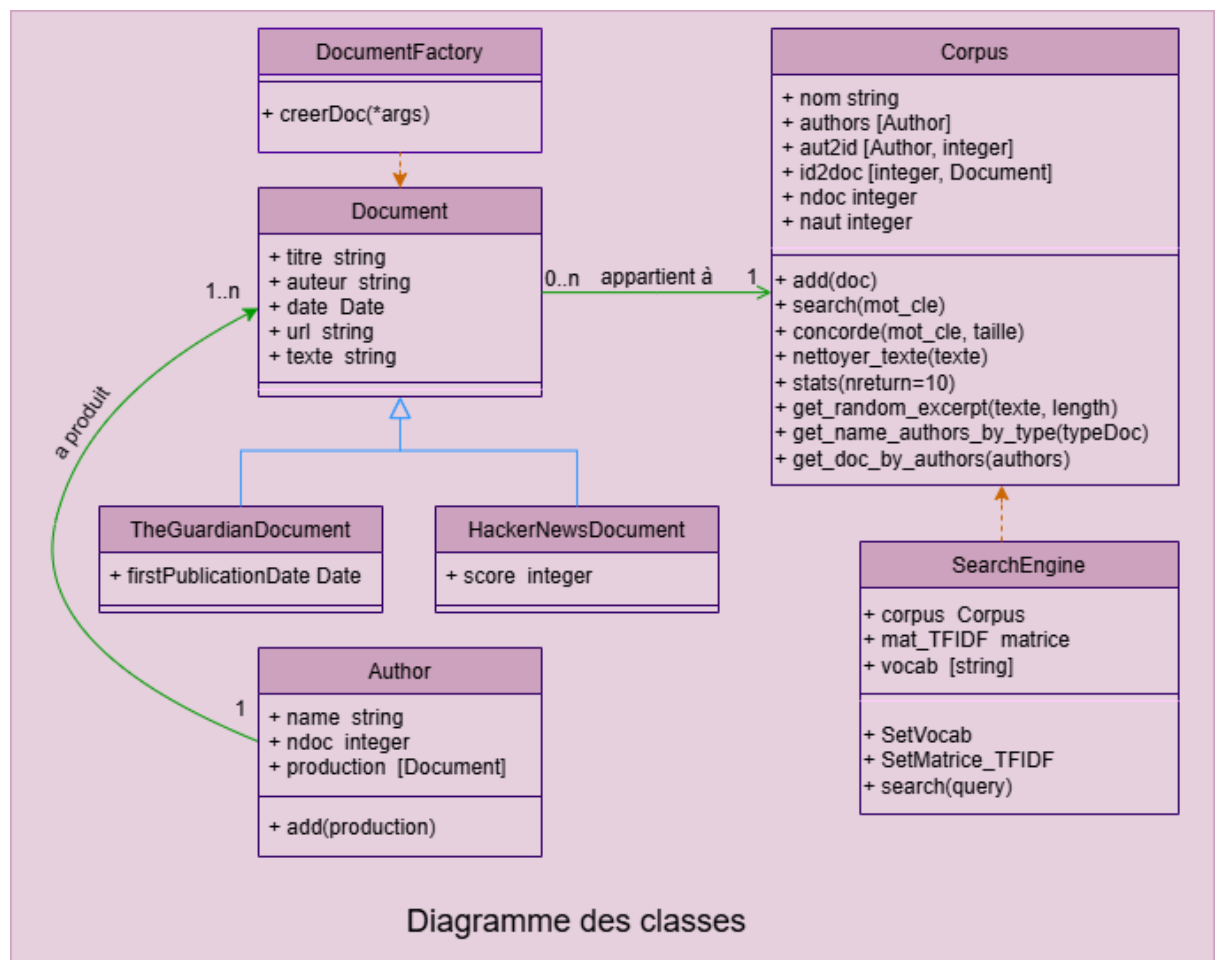
L'utilisateur pourra accéder à des informations issues de deux sources grâce à des fonctionnalités de collecte automatisée de données (Web Scraping). Une fois collectés, les documents sont stockés dans une base de données consultable. Le programme associe à chaque document sa source pour faciliter la navigation et les recherches.

Ces données ont été spécifiées pour répondre aux fonctionnalités principales du programme :

- Permettre des recherches par mots-clés, auteur ou type de document
- Permettre une recherche simultanée entre deux thèmes (deux corpus)

Cela implique un prétraitement des mots saisis pour s'assurer de leur pertinence et du bon fonctionnement des fonctionnalités implémentées, comme le tri des résultats.

## C. Organisation des fichiers et des classes



- **Fichier Classes.py :**
  - Classe Document :  
représente un document générique.
  - Classe HackerNewsDocument :  
hérite de Document, elle représente un document Hacker News.
  - Classe TheGuardianDocument :  
hérite de Document, elle représente un document The Guardian.
  - Classe Author :  
représente un auteur et ses productions.
  
- **Fichier Corpus.py :**
  - Classe Corpus :  
Un Corpus est une collection de documents qu'il permet de manipuler et d'analyser.

**Méthodes principales :**

  - add(doc) : ajoute un document au corpus
  - search(mot\_cle) : recherche les occurrences d'un mot-clé dans les textes de tous les documents du corpus et renvoie les extraits contenant ce mot

- `nettoyer_texte(texte)` : normalise un texte (mise en minuscule, suppression de la ponctuation et des espaces inutiles)
- `stats(nreturn=10)` : calcule les fréquences des mots (TF/IDF) et renvoie un DataFrame des *nreturn* mots les plus fréquents
- Classe DocumentFactory :  
Classe usine permettant de créer des documents en fonction de leur source (Hacker News ou The Guardian)

- **Fichier SearchEngine.py :**

- Classe SearchEngine :  
Représente un moteur de recherche.

**Méthodes principales :**

- `SetVocab()` : construit un vocabulaire (grâce à la méthode `stats()` de `Corpus`) où chaque mot est associé à un id unique et à ses fréquences (term frequency, document frequency)
- `SetMatrice_TFIDF()` : génère une matrice creuse TF-IDF pour optimiser la mémoire et les calculs
- `search(query)` : calcule la similarité cosinus entre la requête de l'utilisateur et chaque document, classe les documents selon le taux de similarité puis renvoie un DataFrame avec les colonnes : Titre, Auteur, Extrait, Similitude, URL, Type

- **Fichier interface.ipynb :**

Constitue l'interface principale de notre programme (sous forme de notebook Jupyter) et est divisé en trois cellules de code :

Cellule 1 : Création des corpus

Cette cellule met un peu de temps à l'exécution (environ 1m 40s)

- Elle comprend la création de deux corpus, en fonction de mots clés spécifiés par les variables "query" et "query2".
- Par souci de simplicité, la création de corpus se fait uniquement par un mot clé unique. Il n'est donc pas possible d'associer aux variables "query" et "query2" un tableau de mots clés.
- Un index des documents et des auteurs est ensuite créé et complété à partir des documents extraits à partir des mots clés précédemment renseignés.

Cellule 2 : Initialisation du moteur de recherche

- Elle initialise le moteur de recherche pour les deux corpus créés dans la cellule précédente.
- La fonction "stats" est appelée pour informer l'utilisateur des mots les plus fréquemment utilisés pour chacun des corpus.
- Ces deux premières cellules sont facultatives lors de l'exécution du programme : l'utilisateur n'est pas contraint de créer des corpus à chaque utilisation.

Cellule 3 : Chargement et gestion des corpus

- Elle est indispensable pour l'exécution de l'application.

- Elle permet de :
  - Charger un corpus construit au préalable.
  - Redéfinir l'index des documents et des auteurs à partir de ceux-ci.
  - Gérer une interface graphique simplifiée, rendant ainsi l'utilisation des deux premières cellules optionnelles.
- Deux corpus de tests sont fournis dans le dépôt Github associé pour tester notre programme avec une taille de données convenable.
- Le code de cette cellule permettant la définition des mots clés et des index est à commenter si l'utilisateur prend la décision de créer de nouveaux corpus.

Il est à noter qu'un fichier Script.py est présent dans les deux premières versions du projet. Il s'agit du fichier à exécuter, pour les versions du projet ne comprenant pas d'interface.

### 3. Conception

#### Répartition des tâches

Nous nous sommes réparti les tâches de la manière suivante :

- Version 1, filtre par auteurs : Pierrick
- Versions 2 et 3, filtre par source : Melissa
- Interface et disponibilité de deux corpus : Melissa et Pierrick

Nous avons néanmoins beaucoup travaillé ensemble sur certaines fonctionnalités, en nous concertant souvent, en expliquant les fonctions et en apportant des améliorations au travail de chacun.

#### Exemple d'utilisation

Notre application est présentée comme ci-dessous :



The screenshot shows a web application titled "Mon moteur de recherche" with a magnifying glass icon. It features a selection of two radio buttons: "war" (unselected) and "war / computer" (selected). Below this, a "Par :" label is followed by two radio buttons: "Mots clés" (selected) and "Auteurs" (unselected). To the right of these are two orange buttons: one with a checkmark and another with a refresh icon. A green label "Mots clés :" is positioned to the left of a text input field containing the placeholder "Entrez vos mots ici". At the bottom, there is a slider control for "Nb de documents max. par thème :" set to the value "5", and a "Type :" dropdown menu currently showing "Tout". A large orange "Rechercher" button with a magnifying glass icon is centered at the bottom.

L'interface reste relativement simple mais permet toutefois quelques fonctionnalités intéressantes pour l'utilisateur.

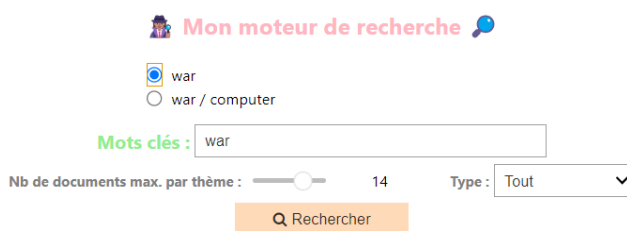
## Choix des sources

L'utilisateur commence par sélectionner les sources de recherche. Il peut choisir d'effectuer des recherches sur un seul corpus ou sur deux corpus simultanément (ici "war" et "war / computer", reprenant les mots clés des corpus fournis avec l'application). Les widgets affichés et activés s'adaptent automatiquement à son choix.

## Recherche avec un seul corpus

Il s'agit du cas d'utilisation le plus simple. L'utilisateur est en mesure de faire une recherche par mots clés. Il dispose d'options pour personnaliser les résultats :

- Nombre maximum de résultats : si le moteur de recherche ne trouve pas suffisamment de documents, il affichera simplement ceux disponibles.
- Type de documents : grâce à une liste déroulante, l'utilisateur peut choisir d'afficher uniquement les documents provenant de "HackerNews", de "The Guardian" ou de combiner les deux.



The screenshot shows a web interface titled "Mon moteur de recherche" with a magnifying glass icon. Below the title, there are two radio buttons: "war" (selected) and "war / computer". A green label "Mots clés :" is followed by a text input field containing "war". Below this, there is a slider for "Nb de documents max. par thème :" set to "14", and a dropdown menu for "Type :" set to "Tout". At the bottom is an orange button with a magnifying glass icon and the text "Rechercher".

**Blackcandy: Self hosted music streaming server**

**Auteur :** nateb2022

**Extrait :** ... process of installation, and it's an ideal choice for self-hosted small server. If you think SQLite is not enough, or ...

[Accès au document sur HackerNews](#)

**Ukraine war briefing: Trump urges China to help end Ukraine war**

**Auteur :** Auteur non trouvé

**Extrait :** ... told NBC's Meet the Press. Trump renewed his warning to Nato allies that he did not see continued US participation ...

## Recherche avec deux corpus

Nous avons un second corpus, similaire au premier mais dont le thème est "Computer". Il sert à offrir à l'utilisateur la possibilité de faire des recherches sur deux corpus simultanément.

On peut choisir la recherche par :

Mots-clés :



Mots clés : animals coding

Nb de documents max. par thème :  2 Type : Tout ▼

Rechercher

#### Résultats pour War :

**macOS menu bar app that shows how full the ISS urine tank is in real time**

Auteur : ajdude

Extrait : ...the epitome of good coding practices since thi...

[Accès au document sur HackerNews](#)

**'The forest will survive': the volunteers saving Kharkiv's war-charred woodland**

Auteur : Luke Harding

Extrait : ...f woodland, killing animals and scaring away co...

[Accès au document sur The\\_Guardian](#)

#### Résultats pour Computer :

**I learned the language of computer programming in my 50s – here's what I discovered**

Auteur : Andrew Smith

Extrait : ...y into the realm of coding and the invaluable ...

[Accès au document sur The\\_Guardian](#)

**Rosetta 2 creator leaves Apple to work on Lean full-time**

Auteur : ladberg

Extrait : ...one line at a time.#codinglife#techinnovation#...

[Accès au document sur HackerNews](#)

#### Auteurs :

Une liste des auteurs disponibles dans la base de données sera proposée, on peut en sélectionner un seul ou plusieurs. Cette méthode (liste plutôt que recherche libre) est utilisée pour éviter toute erreur de saisie. Cette fonctionnalité expose le nom de certaines personnes ainsi que leurs propos, ce qui peut apporter des conséquences. Néanmoins, notre travail reste dans le cadre de notre formation.

Mots clés : Entrez vos mots ici

Nb de documents max. par thème :  2 Type : Tout ▼

☐ Les deux

☐ Thème War

Présence des auteurs : ☒ Thème Computer

Auteurs : Rowena Mason  
Sally Weale  
Paul MacInnes  
unkeen  
Helen Livingstone  
nkko

Rechercher

**Frank Hester: computer programmer who made fortune from public sector contracts**

Auteur : Rowena Mason

Extrait : ... of AI – the company says it is developing an AI model designed to predict the likelihood of patients failing ...

[Accès au document sur The\\_Guardian](#)

Cette fonctionnalité permet de constater les différents thèmes sur lesquels les auteurs ont travaillé, ce qui est bénéfique dans un cadre scientifique, au contraire de la recherche par mots clés. Voici des bénéfices non exhaustifs de la recherche par auteurs :

- Expertise spécifique : Les auteurs sont souvent experts dans un domaine particulier. Rechercher par auteur permet de trouver des travaux de qualité et pertinents dans ce domaine. De plus, ils ont souvent une ligne de recherche cohérente. Cela permet ainsi de suivre l'évolution de leurs travaux et de comprendre leur approche globale.
- Confiance et crédibilité : Certains auteurs sont reconnus pour la qualité de leurs recherches. Rechercher par auteur permet de s'assurer de la crédibilité des informations trouvées. Cela peut aussi mener à d'autres travaux pertinents et à une meilleure compréhension du contexte de la recherche.

Dans cet exemple, l'utilisateur a la liste des utilisateurs présents à la fois sur le corpus "war" et "computer", et cela peu importe le type de document. Le choix "Auteur non trouvé" comprend l'ensemble des auteurs dont le nom n'est pas disponible.

Nb de documents max. par thème :  14    Type :

Présence des auteurs : ☒ Les deux  
☐ Thème war  
☐ Thème computer

Auteurs : 

nateb2022

Auteur non trouvé

Il est important de préciser que l'affichage des résultats est un processus prenant plus de temps que les autres fonctionnalités. Cela est accentué lors d'une recherche avec plusieurs corpus. Il se peut qu'un délai soit présent, il suffit d'attendre quelques instants pour retrouver les résultats correspondants.

## 4. Validation

### Quelques tests unitaires

```
corpus = Corpus("mon corpus")
corpus.add(Document("14/11/2024", "doc1", "me", "url1", "last sunday i was programming a game"))
corpus.add(Document("05/09/2024", "doc10", "my neighbor", "url10", "thank you for watering my plants while I was away, you saved them"))
corpus.add(Document("14/11/2024", "doc3", "my girl", "url3", "your support during the last few weeks has meant the world to me, thank you"))
corpus.add(Document("14/11/2024", "doc4", "my mother", "url4", "i am very very very thankful for what you did last wednesday, you really are the best"))
```

- Résultat d'appel à `corpus.search("thank you")` :

```
2 occurrences trouvées
[' programming a game thank you for wateri', 'nt the world to me, thank you i
am very ']
```

- Résultat d'appel à `corpus.stats(5)` : (les stopwords comme "very" sont ignorés)

```
37 mots différents dans le vocabulaire
Les 5 mots les plus fréquents :
      Mot  TF  DF
3      last  3   3
6     thank  2   2
0      game  1   1
1    sunday  1   1
2 programming 1   1
```

- Maintenant, on crée un objet `SearchEngine` comme ceci :

```
search_engine = SearchEngine(corpus)
```

- On vérifie le contenu de `search_engine.vocab` :

```
{'away': {'id': 0, 'tf': 1, 'df': 1}, 'best': {'id': 1, 'tf': 1, 'df': 1}, 'game': {'id': 2, 'tf': 1, 'df': 1}, 'last': {'id': 3, 'tf': 3, 'df': 3}, 'meant': {'id': 4, 'tf': 1, 'df': 1}, 'plants': {'id': 5, 'tf': 1, 'df': 1}, 'programming': {'id': 6, 'tf': 1, 'df': 1}, 'really': {'id': 7, 'tf': 1, 'df': 1}, 'saved': {'id': 8, 'tf': 1, 'df': 1}, 'sunday': {'id': 9, 'tf': 1, 'df': 1}, 'support': {'id': 10, 'tf': 1, 'df': 1}, 'thank': {'id': 11, 'tf': 2, 'df': 2}, 'thankful': {'id': 12, 'tf': 1, 'df': 1}, 'watering': {'id': 13, 'tf': 1, 'df': 1}, 'wednesday': {'id': 14, 'tf': 1, 'df': 1}, 'weeks': {'id': 15, 'tf': 1, 'df': 1}, 'world': {'id': 16, 'tf': 1, 'df': 1}}
```

- Résultat d'appel à `search_engine.search("game")` :

	Titre	Auteur	Extrait	Similitude	URL
0	14/11/2024	doc1	...i was programming a game...	0.546454	url1
1	14/11/2024	doc4	Extrait non disponible	0.000000	url4
2	14/11/2024	doc3	Extrait non disponible	0.000000	url3
3	05/09/2024	doc10	Extrait non disponible	0.000000	url10

- `search_engine.search("hands")` :

```
Ce(s) mot(s) n'existe(nt) pas dans le vocabulaire du corpus,
ou sont des stopwords
Empty DataFrame
Columns: [Titre, Auteur, Extrait, Similitude, URL, Type]
Index: []
```

### Tests globaux


Aucun résultat pour la requête

## Mon moteur de recherche

- ☒ war  
☐ war / computer

Mots clés :

Nb de documents max. par thème :  5 Type :

 Rechercher

**Aucun résultat trouvé pour la requête : 'fauxMot'**

Aucun auteur choisi

## Mon moteur de recherche

- ☐ war  
☒ war / computer

Par : ☐ Mots clés  
☒ Auteurs




Mots clés :

Nb de documents max. par thème :  5 Type :

Présence des auteurs : ☐ Les deux  
☐ Thème war  
☒ Thème computer

Auteurs :

 Rechercher

**Choisissez un ou plusieurs auteurs pour effectuer une recherche**

## 5. Maintenance

Bien que fonctionnelle, notre application peut être enrichie par de nouvelles fonctionnalités et des optimisations. Voici quelques pistes d'amélioration :

### Exploitation des attributs non utilisés

**Date** : on pourrait utiliser cet attribut de la classe TheGuardianDocument pour implémenter un tri par date. Pour cela, il faudrait d'abord retenir la date de publication de chaque article, puis vérifier simultanément les dates entre les résultats générés par les deux corpus et renvoyer une liste unique et triée.

**Score** : cet attribut de HackerNewsDocument pourrait être utilisé pour afficher les résultats en fonction de la popularité des documents

### Utilisation du deuxième corpus

On peut alors imaginer que travailler sur un résultat englobant les deux corpus prendrait alors encore plus de temps.

- 1) Autoriser l'utilisateur à effectuer des recherches uniquement sur le deuxième corpus.
- 2) Ajouter des fonctionnalités comparatives comme :
  - Statistiques sur les occurrences de termes
  - Identification des auteurs des plus prolifiques
  - Fusion des résultats pour une vue unifiée

Les intérêts de ces potentielles améliorations dépassent l'usage d'un utilisateur classique. Il faudrait travailler avec les résultats des deux corpus de façon simultanée, créer une unique liste de résultats et effectuer les traitements à partir de cette liste. Le potentiel problème que cette fonctionnalité pourrait rencontrer serait le temps de traitement. En effet, travailler de façon conjointe sur deux corpus prend un temps relativement long pour l'utilisateur.

Une autre amélioration possible serait de choisir depuis l'interface les queries (thèmes) utilisés pour les corpus, mais cela ralentirait encore significativement le programme.

De plus, pour optimiser le temps de traitement et améliorer les performances, nous pourrions intégrer des requêtes réseau asynchrones parallèles avec la bibliothèque aiohttp.

## 6. Conclusion

Ce projet représentait pour nous deux notre premier projet en Python.

Nous en avons beaucoup appris et nous avons réalisé à quel point ce langage est bien adapté à ce type de projet, notamment grâce à la richesse de ses bibliothèques et à sa simplicité, qu'on ne retrouve pas toujours dans d'autres langages que nous utilisons habituellement comme Java et C.

Nous sommes plutôt satisfaits de notre travail et du résultat.