Séance 2 : une interface en ligne de commande pour coupler des tables CSV

Bonjour 👋 !

Bienvenue dans la seconde partie de la séquence dédiée au développement d'une **interface en ligne de commande** pour coupler les enregistrements de deux fichiers de données textuelles organisées en tables.

Objectifs de la séance 🎯

- lire, manipuler et écrire des fichiers de tables de données au format CSV ;
- interagir avec un utilisateur via l'invite de commandes ;
- créer un outil de couplage d'enregistrements en ligne de commande paramétrable et réutilisable.

> Important
>
> 1. Répondre aux questions de code en complétant le fichier de script Python `cli_couplage.py`.
> 2. 🆘 Une question n'est pas claire ? Vous êtes bloqué(e) ? N'attendez pas, **appelez à l'aide** 🙋. Le fichier `cli_couplage.py` contient aussi des astuces et aides complémentaires.
> 3. 🤖 Vous pouvez utiliser ChatGPT/Gemini/etc. pour vous aider, **mais** contraignez vous à n'utiliser ses propositions **que si vous les comprenez vraiment**. Ne devenez pas esclave de la machine ! 🙏
> 4. 😌 Si vous n'avez pas réussi ou pas eu le temps de répondre à une question, **pas de panique**, le fichier `correction.py` contient une solution !

> Tip
>
> La difficulté d'une question 🧩 est indiquée de ⭐ à ⭐⭐⭐⭐.

A/ Une CLI en Python

Mais qu'est-ce qu'une CLI ? 🤷

Un utilisateur peut communiquer avec un logiciel à travers une **interface homme-machine** qui peut être graphique (avec des fenêtres, menus, etc.) ou uniquement **textuelle**.

Une interface en ligne de commandes (CLI, pour *command line interface*) est une interface textuelle où l'utilisateur interagit avec le logiciel en tapant des commandes sur un **terminal**.

Quelques exemples de logiciels avec une *CLI* :

- l'interpréteurs de commande de votre système d'exploitation : shell (Linux), zshell (MacOs), PowerShell (Windows) et...
- ...tous les utilitaires disponibles via ces interpréteurs : `cat`, `ls`, `echo`, `top`, `git`, etc.
- la commande `python` et la console Python ;
- le navigateur Web Lynx (oui, il existe des navigateurs Web en mode texte ! 🙂)

Comme vous le voyez, la définition d'une *CLI* est assez large !

> ⓘ Note
>
> 📝 **À retenir.**
> Une *interface en ligne de commande* est la partie d'un logiciel qui permet de l'utiliser et interagir avec depuis un terminal.

Python est un bon candidat pour créer des utilitaires facilement réutilisables car il est compatible avec la plupart des systèmes.
Plusieurs excellentes bibliothèques Python ont donc été créées pour aider à construire des *CLI*.
Les deux principales sont `Click` et `argparse`.
La première est extrêmement complète et puissante, la seconde est plus simple d'utilisation et fait partie de la bibliothèque standard de Python.

`Click` exploite cependant des aspects avancés de Python pour cacher la complexité de certains mécanismes.
On utilisera donc `argparse`, tout à fait suffisante pour cette fois !

> ⚠ Warning
>
> Durant toute la séance, aidez-vous de la **documentation officielle de `argparse`**
> https://docs.python.org/3/library/argparse.html

> Important
>
> 🧩 **- QUESTION 1 -**
>
> Ouvrez `cli_couplage.py` et déclarez l'import de `argparse` en entête du fichier. Ce fichier contiendra tout le code de la *CLI* !

Ma première interface en ligne de commande avec Python 🧸

Qui dit "interface" dit "interaction" : la base d'une *CLI* est donc de permettre à un utilisateur d'interagir avec un logiciel ou un simple programme, via le terminal.

Une première interaction fondamentale est de pouvoir **transmettre des paramètres** depuis le terminal.

Le module `argparse` porte ce nom car il fournit un utilitaire d'analyse syntaxique pour lire et structurer des **arguments** (=paramètres d'une commande) écrits **sur le terminal**.
En anglais on parle d'***argument parsing***, et c'est tout logiquement que l'analyseur syntaxique de `argparse` est une classe nommée `argparse.ArgumentParser` !

> Important
>
> **🧩 - QUESTION 2 - ⭐**
>
> Dans `cli_couplage.py`, instanciez un objet `ArgumentParser` et affectez-le à une variable nommée `cli_parser`.

Cet analyseur d'arguments est le socle de notre *CLI*, c'est lui qui sera chargé de toutes les interactions avec l'utilisateur pour paramétrer le programme de couplage d'enregistrements.

Très bien, mais au fait, c'est quoi un **argument** ?
C'est tout simplement une valeur qui est donnée à la suite du nom de la commande appelée, et qui sera exploité par cette commande lors de son exécution.

Voici un exemple d'un argument, une chaîne de caractères, donné à la commande Linux `echo`, qui l'imprimera sur le terminal.

```
echo "Je suis un argument, la commande `echo` va m'afficher sur le terminal !"
# Je suis un argument, la commande `echo` va m'afficher sur le terminal !
```

Pour que notre analyseur syntaxique puisse lire des arguments, il faut lui dire ce qu'il est sensé lire.
Pour cela, on déclare les attributs attendus auprès de l'analyseur grâce à sa méthode `add_argument()`, qui prend en paramètre un **nom de variable** qui contiendra l'argument lu depuis le terminal.

> Important
>
> **🧩 - QUESTION 3 - ⭐**
>
> Déclarez auprès de `cli_parser` un nouvel argument nommé `fichier1`. Aidez-vous de la documentation !

Pour déclencher l'analyse des arguments, il faut exécuter la méthode `parse_args()` de notre analyseur. Cette méthode va lire le terminal, récupérer les arguments, les traiter, les

organiser et enfin renvoyer un objet de type `argparse.Namespace` qui est une "sorte" de dictionnaire contenant les arguments reconnus.

Si on stocke l'objet `Namespace` retourné par `parse_args()` dans une variable, nommée par exemple `args`, on peut accéder aux arguments par leurs noms avec la syntaxe `args.mon_argument`. Le nom 'mon_argument' est celui donné à l'analyseur via la méthode `add_argument()`

> Important
>
> **🧩 - QUESTION 4- ⭐**
> Après la déclaration des arguments, déclenchez l'analyseur et stockez l'objet `Namespace` dans la variable `args`. Affichez la valeur lue avec `print()`.
> Exécutez le script python depuis un terminal en lui donnant comme argument une chaîne de caractère :
>
> ```
> python cli_couplage.py "Je suis le premier argument"
> ```

> Important
>
> **🧩 - QUESTION 5- ⭐**
> Exécutez `cli_couplage.py` sans arguments, ce qui devrait afficher :
>
> ```
> usage: cli_couplage.py [-h] fichier1
> test.py: error: the following arguments are required: fichier1
> ```
>
> Expliquez (entre vous) la signification de chaque ligne :
> qu'est-ce que `argparse` vous économise de coder par vous-même ?

> Important
>
> **🧩 - QUESTION 6- ⭐⭐**
>
> Observez la première ligne "`usage: cli_couplage.py [-h] fichier1`" : outre l'argument `fichier1`, il y a `[-h]`.
> Exécutez de nouveau le script avec l'argument `-h` :
>
> ```
> python cli_couplage.py -h
> ```
>
> - Que se passe t-il ? À quoi sert le paramètre `-h` ?
> - Notez dans quelle catégorie est "rangé" l'argument `-h` : cherchez dans la documentation de `argparse` ce que cela signifie et quelle est la différence entre ce

> type d'argument et les *positional arguments* comme `fichier1`. Éventuellement, demandez à votre *chatbot* préféré ! 🤖.

Ce serait bien que l'argument `fichier1` serve à donner le **chemin** vers premier fichier CSV à coupler.
Le problème, c'est que pour l'instant il n'y a aucun contrôle sur la valeur lue. L'utilisateur peut passer un nombre, un texte quelconque, aucun problème !

Heureusement, `argparse` peut nous aider grâce au paramètre optionnel `type` de la méthode `add_argument()`.

> Important
>
> 🧩 **- QUESTION 7-** ⭐
> Aidez-vous de la documentation des **types** acceptés par `argparse` (https://docs.python.org/3/library/argparse.html#type) pour ajouter à la déclaration de l'argument `fichier1` qu'il doit être de type **fichier** (*FileType*) à ouvrir en **lecture** (mode 'r').
>
> Vérifiez maintenant que :
>
> ```
> python cli_couplage.py "Je ne suis pas un  fichier !"
> # usage: cli_couplage.py [-h] fichier1
> # test.py: error: argument fichier1: can't open 'Je ne suis pas un  fichier !'
>
> python cli_couplage.py data/didot_1842_small.csv
> # <_io.TextIOWrapper name='data/didot_1842_small.csv' mode='r' encoding='
> ```

> Tip
>
> Un des grands avantages d'utiliser `argparse.FileType()`, c'est que `argparse` se charge d'ouvrir le fichier pour nous en plus de vérifier qu'il existe et qu'il est lisible !

> Important
>
> 🧩 **- QUESTION 8-** ⭐
>
> Puisqu'on a déjà l'argument `fichier1`, il ne reste plus qu'à refaire la même chose pour `fichier2`, l'argument qui doit contenir le second fichier CSV à coupler !

> Tip
>
> ✨ Une bonne *CLI* doit avoir des noms d'arguments concis et les plus explicites possibles, mais aussi une description claire et simple à comprendre.
> On peut donner une description de la *CLI* en paramètre de `ArgumentParser(description=...)`, et des arguments avec `add_argument(help=...)`

> Ces informations seront données dans l'aide générée par `argparse`.
> Par exemple :
>
> ```
> usage: cli_couplage.py [-h] fichier1 fichier2
>
> Un utilitaire de couplage approximatif entre deux tables de données au format
>
> positional arguments:
>  fichier1    Table de données gauche.
>  fichier2    Table de données droit.
>
> options:
>  -h, --help  show this help message and exit
> ```

Nous voilà avec une interface en ligne de commande minimaliste, mais fonctionnelle ! 🥳

Bon, le seul problème c'est qu'elle ne fait pas grand-chose : elle interagit avec l'utilisateur, mais pas encore avec notre programme de couplage !

B/ Lire des tables CSV

Rappelez-vous : dans le programme de couplage d'enregistrements, la fonction finale `couplage()` prenait en paramètres deux listes d'enregistrements, eux-mêmes représentés par des listes.

Pour l'instant, notre *CLI* sait lire deux fichiers et ... c'est tout. Pour faire ingérer le contenu de ces fichiers à la fonction couplage, il faut ajouter un peu de tuyauterie ! En particulier, il faut créer un mécanisme qui va transformer le contenu de ces fichiers en liste d'enregistrements.

Le format CSV sert à représenter des tables de données, qui peuvent être manipulées en Python grâce à au module `csv` inclus dans la bibliothèque standard de Python.
Sa documentation est disponible ici : https://docs.python.org/fr/3.10/library/csv.html

> Tip
>
> 🆘 **Vous ne connaissez pas le format CSV ?**
> Ouvrez le fichiers `data/didot_1842_small.csv` dans un éditeur de texte et demandez de l'aide 🙋

Ce module permet de créer un "lecteur" CSV à partir d'un objet fichier avec la fonction `csv.reader(fichier)`.
Ce lecteur permet de parcourir un fichier CSV comme une simple liste !

> Important

> 🧩 - **QUESTION 9-** ⭐⭐
>
> Importez le module `csv` et utilisez `csv.reader(fichier)` pour récupérer le contenu les tables CSV `fichier1` et `fichier2` sous forme de listes d'enregistrements.
> Stockez ces listes dans deux variables `enregistrements_1` et `enregistrements_2`.

⚠ À partir de maintenant, **testez systématiquement votre *CLI*** avec les fichiers `data/didot_1842_small.csv` et `data/didot_1843_small.csv`

> Important
>
> 🧩 - **QUESTION 10-** ⭐
>
> Si vous affichez le contenu de `enregistrements_1` et `enregistrements_2`, vous pourrez constater que le premier enregistrement contient en fait les entêtes des tables !
> Modifiez votre code pour séparer ces entêtes des données, et stockez-les dans deux autres variables `entêtes_1` et `entêtes_2`.

> Tip
>
> ✨ Pour afficher "joliment" des tableaux dans le terminal avec Python, on peut utiliser le module `tabulate` (à installer).
> Par exemple :

```
table = tabulate.tabulate(enregistrements_1, headers=entête_1, tablefmt="outli
print(table)
# +------------+------------+--------------+
# | per        | act        | loc          |
# +============+============+==============+
# | Abadie     | coiffeur   | 21 Miroménil |
# | Abadie (A. | pharmacien | 10 Ferme     |
# ...
```

C/ Coupler deux tables CSV

On y est presque, encore un peu de courage 🥵

On dispose maintenant de deux listes d'enregistrements, il est temps de rajouter notre méthode de couplage !

> Important
>
> 🧩 - **QUESTION 11-** ⭐

Copiez au même emplacement que `cli_couplage.py` le fichier `couplage.py` fait dans la 1ère séance, **si vous aviez terminé la question 10 de la séance précédente**.
Sinon, renommez le fichier `couplage_correction.py` en `couplage.py`.

Puis dans `cli_couplage.py`, importez la fonction `couplage` depuis `couplage.py` :

```
from couplage import couplage
#    ^-le module     ^-la fonction
```

⚠ Quand on importe un module, Python l'exécute en entier. Donc, si vous importez comme module votre fichier `couplage.py` de la partie 1, chaque exécution de `cli_couplage.py` exécutera tous les "print(...)" de `couplage.py` ! Pas top 🥞 Supprimez donc tous les blocs `🧪 VALIDATION 🧪` et les appels à la fonction `q(...)` dans la copie locale de `couplage.py` pour éliminer les impressions inutiles.

Tip

Oui, on peut utiliser n'importe quel fichier Python local comme un module, et en importer des éléments !
Mais Python exécute le code du module à son import, ce qui n'est pas pratique s'il contient du code qui ne devrait s'exécuter qu'en mode "script" (i.e. quand on l'exécute directement avec `python mon_script.py`).
Heureusement, il est possible d'avoir des comportements différents si le fichier est exécuté comme script ou importé comme module !
C'est en dehors du thème de cette partie, mais une explication très claire est donnée sur https://realpython.com/if-name-main-python/.
Si vous avez le temps, nous pouvons aussi en discuter en fin de séance ! 🙋

Important

🧩 - QUESTION 12- ⭐⭐⭐

Nous voilà au moment de vérité. Dans `cli_couplage.py`, appelez la fonction `couplage` importée en lui passant `enregistrements_1` et `enregistrements_2` et, pour l'instant, un seuil fixe de `0.5`.

Reprenez le code de validation de la `Partie 1 -> Question 10` pour imprimer les *match* identifiés.

Note

🚀 Bonus : pour aller plus loin.

Utilisez la fonction `print_couplage_tables` du module local `util` pour afficher proprement le résultat du couplage !

Dernière touche à notre outil de couplage : sauvegarder le résultat dans un fichier CSV !

Le module `csv` offre une manière d'écrire au format CSV avec `csv.writer(fichier)`, qui retourne on objet "writer" qui fournit la méthode `writer.writerow(liste)`. Cette méthode écrit au format CSV une liste qui lui est donnée en paramètre.

> Important
>
> **🧩 - QUESTION 13- ⭐⭐⭐**
> Ajoutez à la CLI un troisième argument positionnel nommé `sortie`, de type Fichier et ouvert en mode 'w' (*write*). Ici encore, `argparse` va créer le fichier, s'assurer qu'il est inscriptible, etc.
>
> À la suite du couplage, utilisez `csv.writer(args.sortie)` pour exporter un fichier CSV contenant les couplages identifiés entre `fichier1` et `fichier2` !

> Important
>
> **🧩 - QUESTION 14- ⭐**
> Appelez votre commande avec les arguments suivants :
>
> ```
> python cli_couplage.py data/didot_1842_small.csv data/didot_1843_small.csv fic
> ```

Ouvrez le fichier `fichiers_couplés.csv` avec un éditeur de texte ou un tableur (LibreOffice Calc par exemple) et admirez le résultat ! 😎

Ouf, c'est fini ! 🏁

Félicitations🥳🎉
Vous venez de constuire un outil minimaliste mais complet de couplage d'enregistrements, avec une interface en ligne de commande fonctionnelle !

```
usage: cli_couplage.py [-h] fichier1 fichier2 sortie

Un utilitaire de couplage approximatif entre deux tables de données au format CSV.

positional arguments:
  fichier1    Table de données gauche.
  fichier2    Table de données droit.
  sortie      Table de couplage CSV.

options:
  -h, --help  show this help message and exit
```

Pas encore épuisé(e)s ?! 🔥

Vous souhaitez aller plus loin ?
Cette première *CLI* est fonctionnelle, mais on pourrait lui ajouter de nombreuses fonctionnalités. N'hésitez pas à lui en ajouter ! 👍

Quelques exemples :

- Passer le seuil de couplage en argument optionnel, ex. `-s 0.6` [⭐]
- Exporter non pas les couplages, mais les deux tables fusionnées ! [⭐⭐⭐]
- Implémenter un mécanisme de résolution des cas où un enregistrement de la table 1 est couplé à plusieurs de la table 2 (ou vice-versa), par exemple en conservant le couplage avec le score de similarité le plus grand. [⭐⭐⭐⭐]
- Impliquer l'utilisateur dans la décision de couplage dans les cas ambigus, par exemple si la similarité mesurée est proche du seuil ! [⭐⭐⭐⭐] .

Et, bien sûr, vos propres propositions ! 💡🤓