Programmation de spécialité (python)
TD 4 : première structuration du code

Julien Velcin

2022-2023

L'objectif de ce TD est de commencer à structurer le code à l'aide de classes et de modules.

Partie 1 : première classe : le Document

Le contenu textuel n'est pas la seule ressource importante du corpus que vous avez constitué. Les documents sont souvent associés à de nombreuses méta-données comme le nom de l'auteur, l'année de publication, des tags, etc. Pour les gérer, nous allons utiliser le paradigme objet.

1.1 Dans un fichier (module) à part, intitulé `Document.py`, créez une classe `Document` contenant les attributs suivants :

- *titre* : le titre du document
- *auteur* : le nom de l'auteur
- *date* : la date de publication
- *url* : l'url source
- *texte* : le contenu textuel du document

1.2 Créez une première méthode, qui va afficher toutes les informations d'une instance. Implémentez ensuite la méthode `__str__(self)` qui affichera une version "digeste" de l'instance (par exemple son titre seulement). Cette méthode sera appelée quand vous exécuterez la fonction `print(doc)`, où `doc` est une instance de document.

1.3 Modifiez le code du TD précédent afin de pouvoir instancier des objets `Document`. Attention, il faut bien penser à **importer** le module dans le fichier principal. Pour la date de publication, vous pouvez la considérer comme une chaîne de caractère `str` mais le mieux serait de la traiter à l'aide de la librairie `datetime`.

Il faut créer une collection qui doit contenir toutes les instances que vous aurez créées, quelle que soit leur origine. Pour cela, à la place de la liste, vous utiliserez un *dictionnaire* qui va **indexer** les documents, c'est-à-dire les associer à un identifiant unique (leur *clef*) que vous générez au fur et à mesure. Vous appellerez ce dictionnaire d'indices *id2doc* dont les clés sont les identifiants et les valeurs les objets qui représentent les documents.

Partie 2 : Prise en compte des auteurs

Les auteurs aussi sont associés à des méta-données intéressantes. Vous allez implémenter une classe pour gérer ces informations.

2.1 Créez une classe *Author* contenant les attributs suivants:

- `name` : son nom
- `ndoc` : nombre de documents publiés
- `production` : un dictionnaire des documents écrits par l'auteur

Mettez cette nouvelle classe dans un fichier (module) à part, intitulé `Author.py`.

2.2 Créez une première méthode `add` propre à la classe `Author`, qui va permettre d'alimenter l'attribut `production`. Cet attribut permet de compiler l'ensemble des documents rédigés par un auteur. Implémentez ensuite les méthodes `__str__(self)`

2.3 Dans le fichier principal, créez un dictionnaire *id2aut* contenant les auteurs avec comme clef leur nom que l'on considèrera unique. Pour l'alimenter, vous devrez ajouter une instance d'auteur à chaque fois que vous observerez un nouvel auteur au moment d'alimenter le dictionnaire *id2doc*. Vous devrez vérifier si l'auteur est connu, c'est-à-dire si son nom ne fait pas déjà partie des clefs du dictionnaire.

2.4 Ajoutez le code nécessaire pour pouvoir demander quelques statistiques concernant un auteur entré par l'utilisateur : nombre de documents produits et taille moyenne des documents.

Partie 3 : Création du corpus

Il s'agit à présent de structurer le code du fichier principal à l'intérieur d'une classe dédiée à la gestion d'un corpus. Cela peut permettre de gérer *plusieurs* corpus ou un corpus qui évolue dans le temps.

3.1 Créez une classe `Corpus`, qui sera implémentée dans son propre fichier `Corpus.py`, à partir des méthode d'ajout des questions précédentes et contenant les attributs suivants :

- *nom* : le nom du corpus
- *authors* : le dictionnaire des auteurs
- *id2doc* : le dictionnaire des documents
- *ndoc* : comptage des documents
- *naut* : comptage des auteurs

3.2 Implémentez les méthodes permettant d'afficher les élément du corpus triés selon la date et le titre. Les méthodes prennent en paramètre le nombre de documents que vous voulez afficher. Implémentez la méthode `__repr__(self)`, qui permet un affichage plus digeste.

3.3 Créez enfin des méthodes *save* et *load* qui permettent d'enregistrer et charger le corpus sur le disque dur. Pour cela, enrichissez le code que vous avez réalisé dans le TD précédent afin de passer par une table `DataFrame`.

Il est bien sûr possible d'utiliser d'autres formats, comme par exemple `pickle` ou `json`. N'hésitez pas à les tester (optionnel).

3.4 Essayez de sauvegarder et charger vos données pour vous assurer que toute la chaîne de traitement est opérationnelle.