TP Lead Developer - Modules, Fichiers & POO

Le but de ce TP est de vous faire manipuler tous les concepts de la programmation impérative et orienté objet à travers plusieurs exercices.

Une archive compressé *(zip, tar.gz, ...)* sera demandé de rendre à la fin du TP, contenant vos scripts python, ainsi que les fichiers annexes que vous aurez pu créer à rendre sur Teams dans l'onglet **'Devoir'**.

Sommaire

Exercice 1 : Titanic

À l'aide du module `csv` uniquement, exploiter les données du fichier `titanic_survival.csv` situé dans le dossier "Annexes". Itérer sur les données et montrer les résultats suivants.

1. Moyenne d'âge des passagers *(~30 ans)*
2. Pourcentage de survie par classe de passager *(~62% pour la première classe, ~43% pour la deuxième classe, ~26% pour la troisième)*
3. Le bateau de sauvetage ayant sauvé le plus de passagers *(bateau n°13 avec 39 passagers sauvés)*

Exercice 2 : Citations d'écrivains

Auteurs 1	Auteurs 2	Auteurs 3
Edgar Allan Poe	Arthur Schopenhauer	Georges Orwell
Victor Hugo	Simone De Beauvoir	Frank Herbert
Gustave Flaubert	Guy De Maupassant	Isaac Asimov
Ernest Hemingway	Voltaire	Tolkien
Agatha Christie	Emile Zola	William Shakespeare
Friedrich Nietzsche		

À l'aide du package `quote`, obtenable avec `pip`, réalisez les questions suivantes.

Si vous avez des soucis de pour utiliser le module `quote`, alors installez `certifi`, puis ajoutez ce code avant de réaliser l'exercice :

```
import os
import certifi
os.environ['SSL_CERT_FILE'] = certifi.where()
```

Si Quote est indisponible, vous pouvez utiliser l'export de citations statiques `citations.zip` pour réaliser l'exercice.

QUESTIONS

1. Concevoir un inventaire des titres de livres classés par fréquence d'apparition dans le résultat en ordre décroissant de Edgar Allan Poe.
2. Dresser un inventaire des mots classés par ordre décroissant de présence dans les citations de Edgar Allan Poe. Ne pas afficher les mots présents moins de 5 fois.
3. À partir de la liste des auteurs, générer 30 citations pour chacun de ces auteurs et générer un set des mots en commun à tous ces auteurs. *(10 citations si le pc est lent)*

   Exemple : si le mot "concombre" est utilisé dans une des citations de Victor Hugo, alors si "concombre" est présent dans une des citations de TOUS les auteurs, il doit être présent dans ce set.
4. Faire un classement décroissant des auteurs par le nombre de fois qu'ils utilisent le mot "the" en affichant l'auteur et le nombre de fois que "the" est présent dans ses 30 citations.
5. Écrire ce classement dans un fichier csv avec pour noms de colonne "auteur" et "nombre d'occurrences du mot 'the'".

Exercice 3 : Gestion de films (CRUD)

1. Recommandation d'Architecture

- Un fichier `data/movies.json` pour stocker le fichier JSON, créé automatiquement s'il n'existe pas
- Une classe `Movie` pour représenter un film
- Une classe `MovieManager` pour gérer la collection et la persistance
- Une classe `MovieCLI` pour interagir avec l'utilisateur
- Un fichier `main.py` comme point d'entrée pour lancer la CLI

2. Classe `Movie`

Attributs :

- `titre` : string
- `date_de_sortie` : string au format `DD/MM/YYYY`
- `description` : string

La classe doit :

- Valider que la date est au format `DD/MM/YYYY`
- Normaliser le titre avec une majuscule à chaque mot
- Spécifier la méthode `__str__()` qui affiche toutes les informations du film

Optionnellement :

- Avoir une méthode `to_dict()` pour convertir le film en dictionnaire (pour JSON)
- Avoir une méthode `from_dict(data)` pour créer un film depuis un dictionnaire

3. Classe `MovieManager`

Attributs :

- Une liste de films en mémoire
- Le chemin du fichier JSON

Méthodes à implémenter :

a) Persistance

- `load_from_json` : Charger les films depuis le fichier JSON. Si le fichier n'existe pas, ne rien faire.
- `save_to_json` : Sauvegarder tous les films dans le fichier JSON au format :

```
[
    {
      "id": 1,
      "titre": "Le Titre Du Film",
      "date_de_sortie": "19/12/2001",
      "description": "Description du film"
    }
]
```

b) CRUD

- `create_movie` : Créer un nouveau film avec un ID auto-généré, normaliser le titre, valider la date, sauvegarder
- `get_all_movies` : Retourner tous les films
- `get_movie_by_title` : Rechercher un film par titre (insensible à la casse)
- `get_movies_sorted_by_date` : Retourner les films triés par date de sortie croissante
- `update_movie_by_title` : Modifier un ou plusieurs champs d'un film trouvé par titre
- `delete_movie_by_title` : Supprimer un film par titre

4. Interface CLI

Créer une classe `MovieCLI` avec un menu principal proposant :

```
1. Créer un film (CREATE)
2. Afficher les films (READ)
3. Modifier un film (UPDATE)
4. Supprimer un film (DELETE)
5. Quitter
```

CREATE

- Demander le titre, la date (DD/MM/YYYY) et la description
- Valider la date et redemander si invalide
- Afficher le film créé
- Afficher la liste complète des films

READ

- Proposer un sous-menu :
  - Afficher tous les films (triés par date)
  - Rechercher un film par titre
  - Retour au menu principal

UPDATE

- Demander le titre du film à modifier
- Afficher un sous-menu pour choisir quoi modifier (titre / date / description)
- Demander la nouvelle valeur
- Afficher le film après modification

DELETE

- Demander le titre du film à supprimer
- Demander confirmation
- Afficher la liste des films restants