TP: paradigme fonctionnel et map/reduce

M2 MEEF SD NSI

Table 1: Extrait du BO (programme de NSI pour la terminale).

Contenus	Capacités attendues	Commentaires
Paradigmes de programmation.	Distinguer sur des exemples les paradigmes impératif, fonctionnel et objet. Choisir le paradigme de programmation selon le champ d'application d'un programme.	Avec un même langage de programmation, on peut utiliser des paradigmes différents. Dans un même programme, on peut utiliser des paradigmes différents.

Introduction

Le but du TP est de pratiquer un paradigme *fonctionnel* dans un contexte de traitement de données. À partir d'une base de données de films, stockée dans un fichier CSV, on crée une table sur laquelle on va effectuer différentes requêtes au moyen d'opérateurs génériques.

Dans les questions suivantes, on impose de n'utiliser *aucune boucle* (explicite) pour réaliser les opérations demandées mais d'utiliser uniquement les opérations de combinaison sur les listes et générateurs:

- `map` - application d'une fonction à chaque élément,
- `filter` - sélection des éléments qui satisfont une certaine condition,
- `reduce` du module `functools` - combinaison des éléments avec une fonction binaire,
- et les fonctions du même genre que vous jugerez utiles.

```
from functools import reduce
```

L'utilisation des expressions en `lambda` pour créer des fonctions anonymes est bienvenue mais pas obligatoire.

Lecture et affichage des données

Le fichier `movies.csv` fourni contient une liste de films, chacun sur une ligne. Chaque film est composé de champs séparés par le caractère « ; », dans l'ordre suivant:

- un identifiant pour le film (un entier),
- le titre du film (chaîne de caractères ne contenant pas de « ; »),
- l'année de sortie du film (un entier),
- la durée du film en minutes (un entier),
- le rang de ce film dans un classement des meilleurs films (un entier).

On va représenter un film par un objet d'une classe `Movie` avec les attributs `id`, `title`, `year`, `duration` et `rank` pour représenter ces différentes informations.

Question 1. Définir une classe `Movie` avec

- un constructeur qui prend en entrée une liste de chaînes de caractères représentant les différents champs dans l'ordre indiqué ci-dessus,
- une méthode `__repr__` qui présente un film sous forme de chaîne de caractères dans le format suivant, où les champs dans le même ordre que dans le fichier d'entrée : « `[identifiant] Titre (année, durée) #rang` ».

```
class Movie:
    def __init__(self, row):
        ...
    def __repr__(self):
        ...
```

Exemple:

```
Movie("18267", "La Reine des neiges", "2013", "102", "5")
```

```
[18267] La Reine des neiges (2013, 102min) #5
```

Question 2. Lire le contenu du fichier `movies.csv` et en faire une liste `db` dont les éléments sont des instances de la classe `Movie`. On a le droit d'utiliser le module `csv` de Python.

```
import csv
db = ...
```

```
db
```

```
[[2065] The Treasure of the Sierra Madre (1948, 126min) #66,
 [2115] Million Dollar Baby (2004, 132min) #116,
 [2089] No Country for Old Men (2007, 122min) #90,
 [2009] Star Wars : Episode V - The Empire Strikes Back (1980, 190min) #10,
 [2070] Singin' in the Rain (1952, 103min) #71,
 ...
```

Filtrage, extraction de données

Chaque fonction demandée dans la suite doit prendre en argument un itérable (on peut utiliser la variable globale `db` définie au dessus pour tester) et renvoyer un itérable (pas forcément une liste). On rappelle que le code ne doit contenir **aucune boucle**.

Question 3. Écrire une fonction `top10` qui renvoie l'ensemble des films dont le rang est inférieur ou égal à 10.

```
def top10(L):
    ...
```

```
list(top10(db))
```

```
[[2009] Star Wars : Episode V - The Empire Strikes Back (1980, 190min) #10,
 [2006] Schindler's List (1993, 190min) #7,
 [2004] The Good, the Bad and the Ugly (1966, 190min) #5,
 [2001] The Godfather (1972, 175min) #2,
 [2002] The Godfather Part II (1974, 200min) #3,
 [2008] One Flew Over the Cuckoo's Nest (1975, 133min) #9,
 [2007] 12 Angry Men (1957, 96min) #8,
 [2005] Pulp Fiction (1994, 190min) #6,
 [2000] The Shawshank Redemption (1994, 142min) #1,
 [2003] Inception (2010, 148min) #4]
```

Question 4. Écrire une fonction `eighties` qui renvoie l'ensemble des films de la décennie 1980.

```
def eighties(L):
    ...
```

```
len(list(eighties(db)))
```

22

Question 5. Écrire une fonction `titles` qui renvoie l'ensemble des titres des films passés en argument.

```
def titles(L):
    ...
```

```
list(titles(top10(db)))
```

```
['Star Wars : Episode V - The Empire Strikes Back',
 "Schindler's List",
 'The Good, the Bad and the Ugly',
 'The Godfather',
 'The Godfather Part II',
 "One Flew Over the Cuckoo's Nest",
 '12 Angry Men',
 'Pulp Fiction',
 'The Shawshank Redemption',
 'Inception']
```

Réductions

Question 6. Écrire une fonction `max_id` qui renvoie le plus grand identifiant de film dans l'argument, ou 0 si l'ensemble donné en entrée est vide.

```
def max_id(L):
    ...
```

```
max_id(eighties(db))
```

2207

Question 7. Écrire une fonction `average_duration` qui renvoie la moyenne des durées des films. Préciser ce qui se passe si l'ensemble donné en argument est vide.

Attention: pour fonctionner sur n'importe quel itérable, la fonction ne doit faire qu'un seul parcours de l'ensemble passé en argument, en particulier elle ne doit pas utiliser la fonction `len` puisque celle-ci fait un parcours.

```
def average_duration(L):
    ...
```

Quelques exemples :

```
average_duration(db)
```

140.4218009478673

```
average_duration(eighties(db))
```

150.13636363636363

```
average_duration(top10(db))
```

165.4

Question 8. Écrire une fonction `oldest` qui renvoie les trois plus anciens films de l'ensemble passé en argument. À nouveau, on demande de ne faire qu'un suel parcours.

Groupements

Question 9. Écrire une fonction `increasing_years` qui renvoie la liste passée en argument triée par années croissantes. On pourra utiliser la fonction `sorted`, le but n'est pas d'écrire un algorithme de tri.

Question 10. Écrire une fonction `by_year` qui renvoie les éléments d'une liste regroupés par année. La fonction doit renvoyer une liste de couples `(year, movies)` où `year` est une année et `movies` est une liste de films pour l'année `year`. Dans la liste de couples obtenue, deux couples consécutifs doivent avoir des années différentes. Les films sont donnés dans le même ordre que dans la base donnée en entrée. Là encore, on impose de ne pas utiliser de boucle mais de se baser sur la fonction `reduce`.

Question 11. Écrire une fonction `average_by_year` qui renvoie la moyenne des durées des films pour chaque année présente dans la base, sous forme de couple `(year, average)`.

Question 12. Écrire une fonction `average_by_decade` qui calcule les durées moyennes par décennies plutôt que par année, en utilisant les mêmes principes.