2. Functional programming part (Scheme) [8% of your final mark]

In this part of the assignment, we simply ask you to write a function that computes the similarity score between two users. This function will simply be as follows:

```
> (similarity 33 88 list-of-users)
0.02912621359223301
```

The list of users is a list that contains all the users and their liked/not-liked movies. Here is a list with the two users 1 and 31.

```
((1  (260 235 231 216 163 157 151 110 101 50 47 6 3 1) (223 70))
 (31 (367 362 356 349 333 260 235 231) (316 296 223)))
```

As for the other part of the assignment, we provide you with the functions that reads the ratings file. A variable is defined as shown below. It points to the list of ratings.

```
(define Ratings (map convert-rating (read-f "Documents\\test.csv")))
```

```
> Ratings
((1 1 #t)
 (1 3 #t)
 (1 6 #t)
 (1 47 #t) …
```

Note that you should initially test your functions on the small file provided to avoid complex processing time during the development phase.

To represent a user and its movies, a 3-element list is used:

```
(user-ID (list of liked movies) (list of not-liked-movies))
(31 (367 362 356 349 333 260 235 231) (316 296 223))
```

All these users are placed into a list.

In order to define your similarity function, you will need the following functions.

- `(add-rating rating list-of users)` : this function adds a new movie to the current list of users. If it is a rating for a new user, then this user is created.

```
> (add-rating '(31 316 #f) (add-rating '(31 333 #t) '()))
((31 (333) (316)))
> (add-rating '(31 362 #t) (add-rating '(31 316 #f) (add-rating '(31
333 #t) '())))
((31 (362 333) (316)))
```

- `(add-ratings list-of-ratings list-of-users)` : this function adds all the ratings in the list to the current list of users.

```
> (add-ratings '((3 44 #f) (3 55 #f) (3 66 #t) (7 44 #f) (3 11 #t) (7 88 #t)) '())
((3 (11 66) (55 44)) (7 (88) (44)))
> (add-ratings Ratings '())
((1 (260 235 231 216 163 157 151 110 101 50 47 6 3 1) (223 70))
 (31 (367 362 356 349 333 260 235 231) (316 296 223)))
```

- `(get-user ID list-of-users)` : this functions returns the user and its movies.

```
> (get-user 31 (add-ratings Ratings '()))
(31 (367 362 356 349 333 260 235 231) (316 296 223))
```

Submit your project in a zip file containing the scheme functions file.

All your Scheme functions must have a header describing what the function does, the input parameters and the output.

Your program must adhere to the functional paradigm principles. In particular you must not use the functions terminating by ! (such as the set! function) and you must not use iterative loops, always use recursion.

2. Partie Fonctionnelle (Scheme) [8% de votre note finale]

Dans cette dernière partie de votre projet, nous vous demandons simplement d'écrire une fonction calculant le score de similarité entre deux utilisateurs. Cette fonction sera simplement comme suit:

```
> (similarity 33 88 list-of-users)
0.02912621359223301
```

La liste des utilisateurs contient tous les utilisateurs et les films qu'ils ont aimés ou non-aimés. Voici une liste avec seulement deux utilisateurs, 1 et 31.

```
((1  (260 235 231 216 163 157 151 110 101 50 47 6 3 1) (223 70))
 (31 (367 362 356 349 333 260 235 231) (316 296 223)))
```

Comme pour les autres parties du projet, nous vous donnons les fonctions permettant de lire le fichier csv des évaluations de films. Une variable `Ratings` est définie et pointe sur la liste des évaluations de films.

```
(define Ratings (map convert-rating (read-f "Documents\\test.csv")))
```

```
> Ratings
((1 1 #t)
 (1 3 #t)
 (1 6 #t)
 (1 47 #t) …
```

Notez que pour vos tests en cours de développement, vous devriez utiliser le petit fichier fourni afin d'éviter une trop grande complexité de traitement.

Afin de représenter un utilisateur, une liste à trois éléments est utilisée :

```
(user-ID (list of liked movies) (list of not-liked-movies))
(31 (367 362 356 349 333 260 235 231) (316 296 223))
```

Tous les utilisateurs sont ainsi placés dans une longue liste,

Afin de définir votre fonction de similarité, vous devrez définir les fonctions suivantes :

- `(add-rating rating list-of users)` : cette fonction ajoute un nouveau film à la liste courante d'utilisateurs. Si l'entrée est l'évaluation d'un nouvel utilisateur, alors cet utilisateur est créé et ajouté à la liste.

```
> (add-rating '(31 316 #f) (add-rating '(31 333 #t) '()))
((31 (333) (316)))
> (add-rating '(31 362 #t) (add-rating '(31 316 #f) (add-rating '(31
333 #t) '())))
((31 (362 333) (316)))
```

- `(add-ratings list-of-ratings list-of-users)` : cette fonction ajoute toutes les évaluations dans une liste à la liste courante des utilisateurs.

```
> (add-ratings '((3 44 #f) (3 55 #f) (3 66 #t) (7 44 #f) (3 11 #t) (7 88 #t)) '())
((3 (11 66) (55 44)) (7 (88) (44)))
> (add-ratings Ratings '())
((1 (260 235 231 216 163 157 151 110 101 50 47 6 3 1) (223 70))
 (31 (367 362 356 349 333 260 235 231) (316 296 223)))
```

- `(get-user ID list-of-users)` : cette fonction retourne un utilisateur et ses films.

```
> (get-user 31 (add-ratings Ratings '()))
(31 (367 362 356 349 333 260 235 231) (316 296 223))
```

Vous devez soumettre un fichier zip contenant vos fonctions Scheme. Chaque fonction doit être précédée d'un entête expliquant ce que fait cette fonction, ses paramètres et ce qu'elle retourne.

Votre solution doit se conformer au paradigme de programmation fonctionnelle. En particulier vous ne devez pas utiliser les fonctions se terminant par `!` (par exemple le `set!`) et vous ne devez pas utiliser de boucles itératives, utiliser toujours la récursivité.