TP6 : Lecture/Ecriture

Nous souhaitons dans cet exercice travailler en lecture et en écriture sur un fichier de clients.

Soit le fichier clients.txt contenant une liste de clients:

```
2823
Paul Henriot, 59 rue de l'Abbaye, 51100 Reims, France
Daniel Da Cunha, 27 rue du Colonel Pierre Avia, 75508 Paris, France
Julie Young, 78934 Hillside Dr., 90003 Pasadena, USA
Julie Brown, 7734 Strong St.,  San Francisco, United States
...
```

La première ligne contient le nombre de clients référencés. Chaque ligne suivante contient la description d'un client, ici le prénom, le nom ainsi que l'adresse du client (et le pays).

Le but de l'exercice est d'écrire des fonctions de recherche, d'insertion et de suppression permettant de librement consulter et mettre à jour le fichier de clients.

Dans un premier temps, regardez son contenu avec un éditeur de texte (geany, gedit ou autre...). Les éditeurs de textes permettent de librement transformer le contenu d'un fichier texte. Il est cependant préférable ici de ne pas modifier le fichier.

1. Préambule

1.1 Tests unitaires

Lancez le programme pycharm et créez un nouveau projet (TP6).

- Ouvrez un terminal et executez la commande `pycharm`
- Lors de la première ouverture, pensez à bien sélectionner python 3 comme interpréteur par défaut.
- Tutoriel Pycharm

Toutes les fonctions que vous écrirez doivent être correctement testées avant leur mise en œuvre. Commencez donc par lire et exécutez les exercices du **Tutoriel sur les tests unitaires**

1.2 Structure du projet

Vous devez maintenant dans votre projet créer trois onglets :

- L'onglet `main.py` sera le programme principal. Mettez-y uniquement:

```
from gestion_clients import *
```

- L'onglet `gestion_clients.py` est le fichier contiendra toutes les fonctions servant à gérer le fichier de clients. Mettez-y pour l'instant:

```
import os, sys
```

- L'onglet `tests.py` contiendra les fonctions de test (servant à tester les fonctions que vous écrirez). Mettez-y pour l'instant:

```
from gestion_clients import *
```

Demandez à Pycharm d'exécuter `tests.py` à l'aide de l'environnement de test (voir tuto).

Copiez le fichier clients.txt dans votre projet (glissez-déplacez dans le panneau de gauche). Vous devez le voir apparaître dans un onglet supplémentaire.

1.3 Premières fonctions

Commencez par copier les fonctions suivantes dans `gestion_clients.py`:

```
def ouvre_fichier(nom_fic, verbose = True):
    '''Cette fonction ouvre un fichier en effectuant quelques test (vérifie
l'existence du fichier demandé)
    Attention : le fichier est ouvert en lecture/écriture
    argument :
      - le chemin d'accès nom_fic
    retourne :
      - le descripteur du fichier ouvert'''
    try:
        assert os.path.isfile(nom_fic)
        f = open(nom_fic, 'r+')
        if verbose :
            print("Connexion au fichier ", nom_fic, "OK.")
        return f
    except:
        if verbose:
            print("Erreur de connexion : le fichier n'existe pas!")
        sys.exit()

def lire_a_la_position(i, nom_fic):
    '''Cette fonction positionne la tête de lecture sur la kième ligne du
fichier
    et retourne le contenu de cette ligne
    arguments :
      - i : position des données dans la liste
      - nom_fic : chemin d'accès au fichier
    retourne :
      - une chaîne de caractères (le contenu de la ligne)'''
    with ouvre_fichier(nom_fic, verbose=False) as f:
        f.seek(i * 91)
        return f.readline()[:-1]
```

```
def ecrire_a_la_position(i, data, nom_fic):
    '''Cette fonction sert à écrire dans le fichier les données par trames
de 90 caractères (+ retour chariot)
    argument :
      - i : position dans la liste
      - data : données à écrire
      - nom_fic : chemin d'accès au fichier '''
    with ouvre_fichier(nom_fic, verbose=False) as f:
        f.seek(i * 91)
        if len(data) < 90:
            f.write(data + ' ' * (90 - len(data)) + '\n')
        else:
            f.write(data[:90] + '\n')
        f.flush()
```

A faire :

- La première fonction sert simplement à ouvrir un fichier. Ajoutez la ligne suivante dans le programme principal:

```
f = ouvre_fichier('clients.txt')
```

Exécutez le programme principal et vérifiez que la phrase

```
Connexion au fichier clients.txt OK.
```

s'affiche

- le fichier possède en tout 2824 lignes. Utilisez la fonction `lire_a_la_position` pour lire et afficher les informations du client situé à la ligne 47.
- Utilisez maintenant la fonction `ecrire_a_la_position` pour écrire l'information suivante : "Laurence Lebihan, 12, rue des Bouchers, 13008 Marseille, France" à la position 2. Vérifiez dans le fichier `clients.txt` que la nouvelle cliente a bien été enregistrée.

2. Fonctions de recherche élémentaires

2.1 Lire le nombre de clients

Écrire une fonction qui lit le nombre de clients dans le fichier. Le nombre de clients référencés est inscrit sur la première ligne du fichier (d'index 0). Nous allons ici écrire une notre première fonction de lecture :

```
def nombre_clients(nom_fic):
  ...
```

Cette fonction prend en argument le nom du fichier, ouvre le fichier, lit le contenu de la première

ligne et retourne la valeur entière qui y est écrite.

- La fonction ne lit que la première ligne et ne vérifie pas le nombre réel de clients enregistrés. Ce nombre doit donc être soigneusement maintenu lors de toute modification du fichier de clients.
- Faites appel à la fonction `lire_a_la_position` pour lire la première ligne. Cette fonction retourne une *chaîne de caractères*. Vous devez donc convertir cette chaîne de caractères en entier avec la fonction `int`.

Pour tester le bon fonctionnement de cette fonction, vous utiliserez la fonction de test suivante:

```
def test_nombre_clients():
    assert type(nombre_clients(nom_fic)) is int
    assert nombre_clients(nom_fic) >= 0
```

Cette fonction teste uniquement si la variable retournée est un entier positif. Elle ne vérifie pas le nombre *effectif* de clients inscrits dans le fichier.

2.2 Recherche

Comme dans le TD6, nous allons maintenant écrire une fonction de recherche qui prend en argument le nom d'un client et le nom du fichier et retourne:

- -1 si le client n'est pas dans le fichier
- La position de sa *première* occurrence dans la liste s'il est présent.

```
def recherche(client, nom_fic):
  ...
```

La fonction ouvre le fichier, lit le nombre de clients enregistrés et initialise la réponse à -1. Puis, à l'aide d'un boucle sur le nombre de clients, la fonction lit les lignes une à une et copie le numéro de ligne dans la variable réponse si le client est présent dans cette ligne. La fonction retourne la réponse en sortie de boucle

- Pour tester si `client` est présent dans `ligne`, il suffit d'écrire

```
  if client in ligne:
     ...
```

- il est possible de faire la recherche uniquement avec le prénom et le nom
- Pour retourner la première occurrence du client (et non la dernière), il est important de sortir de la boucle dès que client est trouvé

Pour tester le bon fonctionnement de cette fonction, vous utiliserez la fonction de test suivante:

```
def test_recherche():
    assert type(recherche('', nom_fic)) is int
        assert  -1  <  recherche('',  nom_fic)  <=
nombre_clients(nom_fic)
```

La fonction vérifie uniquement que l'entier retourné et bien compris entre -1 et le nombre de clients.

2.3 Ajout d'un client

Écrire une fonction qui ajoute un client dans le fichier. Pour insérer un nouveau client, il suffit d'écrire les données dans le premier emplacement non utilisé. Si N est le nombre de clients avant insertion, le premier emplacement libre est à la position N+1 (Une fois l'insertion effectuée, penser à mettre à jour la valeur de N).

Écrire une fonction:

```
def ajoute(client, nom_fic):
  ...
```

qui prend en argument la chaîne de caractère client à insérer ainsi que le nom du fichier. La fonction ouvre le fichier, lit le contenu de la première ligne pour connaître le nombre de clients, puis

- inscrit le nouveau client à la ligne N + 1
- met à jour la valeur de N à la ligne 0

La fonction ne retourne aucun résultat. Son effet est uniquement visible dans le fichier `clients.txt`. Pensez à vérifier après chaque utilisation que le fichier `clients.txt` a bien été modifié.

Pour tester le bon fonctionnement de cette fonction, vous utiliserez la fonction de test suivante:

```
def test_ajoute():
    N_avant_ajout = nombre_clients(nom_fic)
    client_test = 'TEST'
    ajoute(client_test, nom_fic)
    N_apres_ajout = nombre_clients(nom_fic)
    assert N_apres_ajout == N_avant_ajout + 1
    assert recherche(client_test, nom_fic) > 0
```

2.4 Suppression d'un client

Écrire une fonction qui supprime un client du fichier. Pour supprimer un client de la liste, il faut commencer par vérifier que celui-ci est bien présent dans le fichier avec une recherche. Si le client est présent à la position `i`, la fonction lit le nombre de clients `N` sur la ligne 0 puis effectue les deux opérations suivantes :

- Copie le contenu de la ligne `N` à la position `i`
- Écrit la valeur `N - 1` à la position 0

```
def supprime(client, nom_fic):
  ...
```

La fonction ne retourne aucun résultat. Son effet est uniquement visible dans le fichier `clients.txt`. Pensez à vérifier après chaque utilisation que le fichier `clients.txt` a bien été modifié.

Pour tester le bon fonctionnement de cette fonction, vous utiliserez la fonction de test suivante:

```
def test_supprime():
    N_avant_suppression = nombre_clients(nom_fic)
    client_test = 'TEST'
    supprime(client_test, nom_fic)
    N_apres_suppression = nombre_clients(nom_fic)
     assert N_avant_suppression - 1 <= N_apres_suppression <=
N_avant_suppression
```

3. Gestion des doublons

3.1 Recherche multiple

Une fonction de recherche multiple recherche toutes les occurrences d'un certain client dans le fichier.

- si le client n'est pas présent, la recherche retourne une liste vide
- si le client est présent au moins une fois, la recherche retourne la liste de tous les numéros de ligne où le client est présent.

```
def recherche_multiple(client, nom_fic):
   ...
```

Pour tester le bon fonctionnement de cette fonction, vous utiliserez la fonction de test suivante:

```
def test_recherche_multiple():
```

```
    assert type(recherche_multiple('', nom_fic)) is list
    assert min(recherche_multiple('', nom_fic)) > 0
       assert  max(recherche_multiple('',  nom_fic))  <=
nombre_clients(nom_fic)
       assert  len(recherche_multiple('',  nom_fic))  <=
nombre_clients(nom_fic)
```

3.2 Ajout sans doublon

Une fonction d'ajout sans doublon vérifie avant l'insertion d'un nouveau client que celui-ci n'est pas déjà présent dans la liste.

- si le client n'est pas présent, faire une insertion normale
- sinon ne rien faire

```
def ajoute_sans_doublons(client, nom_fic):
   ...
```

Pour tester le bon fonctionnement de cette fonction, vous utiliserez la fonction de test suivante:

```
def test_ajoute_sans_doublons():
    N_avant_ajout = nombre_clients(nom_fic)
    client_test = 'TEST_SANS_DOUBLON'
    ajoute_sans_doublons(client_test, nom_fic)
    N_apres_ajout = nombre_clients(nom_fic)
    assert N_avant_ajout <= N_apres_ajout <= N_avant_ajout + 1
    assert recherche(client_test, nom_fic) > 0
    assert len(recherche_multiple(client_test, nom_fic)) == 1
```

3.3 Suppression multiple

Une fonction de suppression multiple supprime toutes les occurrences du client recherché dans le fichier.

Effectuer une recherche multiple et:

- si le client n'est pas présent, ne rien faire
- sinon supprimer chaque occurrence

```
def supprime_multiple(client, nom_fic):
   ...
```

Pour tester le bon fonctionnement de cette fonction, vous utiliserez la fonction de test

suivante:

```
def test_supprime_multiple():
    N_avant_suppression = nombre_clients(nom_fic)
    client_test = 'TEST_SANS_DOUBLON'
    nb_doublons = len(recherche_multiple(client_test, nom_fic))
    supprime_multiple(client_test, nom_fic)
    N_apres_suppression = nombre_clients(nom_fic)
      assert N_apres_suppression == N_avant_suppression -
nb_doublons
    assert len(recherche_multiple(client_test, nom_fic)) == 0
```

4. Suppression des doublons

Ecrire une fonction qui supprime tous les doublons du fichier de clients.

```
def supprime_doublons(nom_fic):
    ...
```

Pour tester le bon fonctionnement de cette fonction, vous utiliserez la fonction de test suivante:

```
def test_supprime_doublons():
    supprime_doublons(nom_fic)
    N = nombre_clients(nom_fic)
    for i in range(1, N + 1):
        client = lire_a_la_position(i, nom_fic)
        assert len(recherche_multiple(client, nom_fic)) == 1
```

Après utilisation de cette fonction, regardez dans quelle proportion le nombre de clients a diminué.